SData 2.0: Expressing metadata in JSON

Version 1.0

1 Introduction

1.1 Background

SData [1] is a standards-based protocol used by many Sage products to share information and promote integration. SData is based on HTTP (Hypertext Transfer Protocol), the protocol that powers most of the internet traffic around the globe, and is suitable for use in Application Programming Interfaces (APIs), mobile applications, and for application integration.

SData is published under a Creative Commons Licence, and may be freely reused as a specification.

SData version 1.0 was published in 2010, and updated in 2011 to the current version, v1.1. This document, along with four others defines the next version of SData, 2.0, which focuses on simplifying the protocol and introduces full support for JavaScript Object Notation (JSON) [2], alongside the XML and Atom support introduced in SData v1.0.

The full set of documents defining SData 2.0 is:

"The underlying approach to evolving SData"	Outlines how the SData Working Group approached the task of updating the SData protocol while ensuring compatibility with implementations of version 1.0 and 1.1 already in use.
"SData 2.0 Core"	Defines the main elements of the SData protocol, explains how these elements are being updated (and in some cases, relaxed) for the 2.0 release, and outlines how JSON is being integrated into these elements.
"JSON formatted SData responses"	Defines the JSON format for SData content, focusing on structural aspects of content and representation.
"SData 2.0 Expressing metadata in JSON"	This document. Builds on the JSON formatted SData responses to define how providers should express metadata in JSON.
"SData 2.0 and Sage ID"	Specifies how Sage ID Authentication is handled in SData 2.0.

1.2 Overview

When designing distributed systems, there is always a balance to be struck between flexibility and simplicity. One particular instance of this balance is in the amount of "hard-coded" or built-in knowledge that a client, or consumer, should have about the server, or provider, that it is using[1]. Historically, consumers have been designed with significant amounts of built-in knowledge about the provider, tipping the balance in favour of simplicity and against flexibility. In this situation the provider needs only to provide raw information, as knowledge necessary to handle this raw information is encoded into the client. Increasingly, however, we are building more versatile clients, capable of communicating with several different providers and able to adapt, dynamically, to data and metadata from any given provider.

SData 2.0 is focused on supporting these more versatile clients and treats the representation, transport, and handling of metadata as an important protocol element. This document defines how SData providers should expose JSON metadata at the feed, entry and property level.

[1] "Provider" and "consumer" are used in SData broadly interchangeably with the terms "server" and "client".

2 Required attributes of the JSON metadata representation

Practical experience applying SData, along with feedback from teams using JSON, clearly demonstrates that the capabilities of SData v2.0 for expressing metadata should possess at least the following attributes:

- Support for JSON as a first-class representation (i.e., no requirement for XML or Atom support when using JSON as the metadata representation).
- Available at the resource and property level.
- Retrievable:
  - alongside a regular feed (similar to the `includeSchema` URL parameter in SData v1.1).
  - from a predefined location (similar to the `$schema` URL segment in SData v1.1).
- Support of the relevant metadata elements defined in the SData v1.1 specification.
- Standards-based where appropriate.
- As concise and unobtrusive as possible.
- Human and machine readable.
- Suitable for validation and automated testing as well as the main use case of enabling smart clients to respond to metadata.

3 JSON metadata in SData

JSON Metadata in SData appears in the following structures:

- **Type definitions:** defines the SData-recognized set of types.
- **Links (hypermedia controls):** are JSON objects describing operations (Create/Read/Update/Delete and also SData queries and services) on resources.
- **Embedded metadata:** covers metadata provided within a resource representation. It is optionally attached to resources and their properties.
- **Prototype:** prototypes are the JSON counterpart of schemas. In contrast to the detailed and restrictive nature of an XSD document, the prototype is:
  - Expressed in JSON.
  - Simple and compact, making use of string composition to bind a generic template to the information delivered at the resource level.

The JSON characteristics are:

- Metadata is expressed in JSON and relates to resource kind representations and their properties; there is no overarching prototype that describes an entire contract (in contrast to the schema in SData v1.1).
- The static/structural metadata of a representation SHOULD be collected in a JSON object called **$prototype**.
  - Prototypes describe, by means of metadata, the characteristics of a representation (e.g., structure, type, links).
  - A complete resource (i.e. containing both the data and the metadata) is obtained by a merge and substitution process combining the raw data and the metadata contained in the prototype.

    - Representations and prototypes are ideally in a 1-1 relationship.

- Metadata MAY be embedded in responses, but is easy to separate from the raw payload. Embedded metadata extends/overrides the metadata of the associated prototype if one such exists. If no prototype is present, the embedded metadata is the only kind available to the consumer.
- The SData-defined metadata set (presented in this document) can be augmented by contract-specific metadata elements.
- All SData entities expressed in JSON MUST be valid JSON documents, whether containing metadata extensions or not.

4 Requesting metadata

Provision of metadata is OPTIONAL for SData Providers. If a Provider does support metadata, it MAY choose to provide consumers with metadata by default.

A consumer can specifically request metadata-enhanced responses through URL parameters as shown below:

- `GET http://www.example.com/sdata/myApp/-/-/products?`**`includeMetadata`**`=true`
- `GET http://www.example.com/sdata/myApp/-/-/products?`**`includePrototype`**`=true`

In both examples, the response will be supplemented with metadata. In the second case, the metadata will take the form of an embedded $prototype object as explained in more detail in section 10.

The "include" parameters provide an efficient[2] manner for consumers to retrieve metadata.

In the case of prototypes, these may also be directly retrieved from links specified by the Provider, for example:

- `GET http://www.example.com/sdata/MyApp/-/-/$prototypes/addresses('detail')`

The prototypes of an application are resources exposed under a `$prototypes` URL segment. While the complete URL delivering prototypes is a matter for the application, it is suggested that the `$prototypes` be located at the same level as resource kinds.

As discussed in section 10, a resource kind may expose several prototypes. These are returned as a JSON feed by a GET operation on the URL ending in `$prototypes/`*`resourceKindName`*.

For example, the following will retrieve a feed of `$prototypes` for the addresses resource kind:

- `GET http://www.example.com/sdata/MyApp/-/-/$prototypes/addresses`

[2] In this manner an additional round-trip to retrieve the prototype separately is avoided.

5 Conventions for metadata

In SData, metadata elements and payload properties may co-exist in the same JSON object. In order to distinguish between the payload and metadata properties, SData requires that metadata elements have names prefixed by a dollar sign ('**$**'). SData-defined metadata elements are presented in Appendix A.

A metadata property with a value of `null` is ignored.

All complete examples in this document are presented in gray boxes. These complete examples are all valid JSON documents. The convention of a property name of `"..."` with a property vaue of `"..."` is used to indicate that some detail which is not relevant to the topic being explained has been omitted.

6 Substitution formalism

Metadata associated with the natural payload of a resource can be quite extensive and the values they carry (often URLs) are quite verbose. To reduce bandwidth consumption and to ease readability, SData 2.0's JSON format defines a formalism that allows a compressed representation of string values.

The substitution formalism is simple: the string values of metadata elements may contain a property name (metadata or payload) enclosed in curly brackets (`"{"` and `"}"`). Once a payload is received, the above specified string portions are substituted with the corresponding values. The syntax is a simplified version of RFC 6570, "URI Template" [3], applicable to any information, not just URIs.

The substitution process is described as follows:

- For every complete resource (payload + all associated metadata), for every metadata string property in the object, perform the following:
- If the string contains an un-escaped[3] `"{`*`identifierString`*`}"` substring then
  - If the *identifierString* is a defined property name determined according to the scoping rules below, then:
    - Replace `"{`*`identifierString`*`}"` with its corresponding value in string format.
  - Otherwise, a formal error has occurred.

Scoping rules: using the specification "X" : "{Y}"

- If the values of X and Y are different:
  - The initial search scope is the scope containing the definition of the property X.
- Otherwise:
  - The initial search scope is the scope immediately enclosing (logically "above") the scope containing the definition of the property X.
- If a property Y can be found in this scope, its value is used in the substitution.
- Otherwise, the enclosing scopes are searched for property Y, from the initial scope "upwards".

The reader will encounter specifications of the form "X": "X" in the section on Links, where they are used to indicate that the URL for a link should be obtained by substitution of the URL in the containing object.

[3] The substitution mechanism can be escaped by using "{{" and "}}" instead of the single curly brackets.

The substitution formalism defines a recursive process with a fixed recursion depth set as default to 5; the value of the recursion depth can be overridden if necessary in the contract.

Example: Consider the following simple SData entry expressed in JSON:

```
{
   "$baseUrl": "http://www.example.com/sdata/MyApp/-/-",
   "$url": "{$baseUrl}/addresses?CreditExceeded=true",
   "$title": "Account {accountId} of {companyName} has exceeded credit limit",
   "companyName": "ACME Inc.",
   "accountId": "A-1322",
   "ID": "7123a",
   "Street": "Lerchenweg",
   "StreetNumber": 11,
   "PostalCode": 71711,
   "City": "Marbach am Neckar",
   "Country": {
        "$url": "{$baseUrl}/countries('{ISOCode}')",
        "Name": "Germany",
        "ISOCode": "DE"
   }
}
```

In this case, the metadata properties highlighted in red (two instances of `$url` and one of `$title`) have values that require substitution. This is applied as follows:

- `$baseUrl` value into:
  - `$url` at the entity level.
  - `$url` within the `Country` property.
- `accountId` and `companyName` in the $title at the feed level.
- `ISOCode` value into the respective `$url` of the `Country` object.

The substitutions and the resulting logical JSON object (final) are shown below:

```
{
   "$baseUrl": "http://www.example.com/sdata/MyApp/-/-",
   "$url": " http://www.example.com/sdata/MyApp/-/-/addresses?CreditExceeded=true",
   "$title": "Account  A-1322 of  ACME Inc. has exceeded credit limit",
   "companyName": "ACME Inc.",
   "accountId": "A-1322",
   "ID": "7123a",
   "Street": "Lerchenweg",
   "StreetNumber": 11,
   "PostalCode": 71711,
   "City": "Marbach am Neckar",
   "Country": {
        "$url": " http://www.example.com/sdata/MyApp/-/-/countries('DE')",
        "Name": "Germany",
        "ISOCode": "DE"
   }
}
```

7 SData JSON Types

The type of an SData JSON property or object is defined by its associated `$type` property. The possible values of the `$type` property follow the MIME type formalism; the process of registering the `"sdata/"` prefix with the Internet Assigned Numbers Authority (IANA) is underway. The types are presented in the following sub-sections, divided into basic and complex types.

The approach of defining explicit SData basic types for `date`, `time` and `datetime` (rather than representing these types as SData `string` types with associated formats) was taken in the interests of familiarity to application designers and programmers.

7.1 Basic types

7.1.1 sdata/boolean

The `sdata/boolean` type conforms to the JSON Boolean type with values that can be either `true` or `false.`

7.1.2 sdata/string

The `sdata/string` type conforms to the JSON string type.

An `sdata/string` type specification can be complemented through an additional format definition. This defines an underlying pattern for the string. The following `$format` specifications are defined:

- **`email:`** the string MUST consist of an email address conforming to RFC 5322, "Internet Message Format" [4].
- **`currency`**: the string MUST consist of an alphabetic 3 letter code conforming to ISO 4217, "*Codes for the representation of currencies and funds*" [5].
- **`locale`**: the string MUST conform to the format specified in the HTTP Accept-Language header and detailed in the language tags specification of section 3.10 of RFC 2616, "Hypertext Transfer Protocol – HTTP/1.1" [6].
- **`country`**: the string must consist of an alphabetic 2 letter code conforming to ISO 3166-1, "Codes for the representation of names of countries and their subdivisions" [7].
- **`phone`**: the string SHOULD consist of a valid dialling sequence. Due to the lack of a widely accepted standard, no specific recommendation is made on conformance, but restriction of characters to `"'0'..'9', '+', '-', ' ', '.', '(', ')'"` (i.e., numbers, the plus sign conventionally used to indicate a nationally defined prefix for International Direct Dialling, and the dash, space, period and brackets commonly used to separate and group elements of a dialling sequence) is encouraged to aid interoperability.

Other formats can be defined as deemed necessary through contracts.

Example: Some properties of a fictional `Contact` resource could be expressed as follows:

```
{
      "..." : "...",
      "countryOfResidence" : {
          "$type" : "sdata/string",
          "$format" : "country"
      },
      "preferredCurrency" : {
          "$type" : "sdata/string",
          "$format" : "currency"
      },
      "displayLanguage" : {
          "$type" : "sdata/string",
          "$format" : "locale"
      },
      "emailAddress" : {
          "$type" : "sdata/string",
          "$format" : "email"
      },
      "telephone" : {
          "$type" : "sdata/string",
          "$format" : "phone"
      }
}
```

The payload for a resource conforming to this definition could look like:

```
{
      "..." : "...",
      "countryOfResidence" : "GB",
      "preferredCurrency" : "GBP",
      "displayLanguage" : "en-GB",
      "emailAddress" : "john.doe@example.org",
      "telephone" : "+44 191 294 3000"
}
```

7.1.3 sdata/number

The `sdata/number` type is equivalent to the JSON `number` type. Both specifications can be used in an SData context.

Example:

- `"avogadroConstant": 6.0221413e+23`

7.1.4 sdata/integer

An `sdata/integer` property contains a (possibly signed) integer value.

Examples:

- `"kilo": 1024`
- `"minusOne": -1`

7.1.5 sdata/decimal

The `sdata/decimal` type is a `string` representing a (possibly signed) rational number with a finite number of decimal places. The decimal point is a period. The string type is necessary since some languages, such as JavaScript, have difficulties in representing decimal numbers (for example, 0.1 + 0.2 === 0.3 is false in JavaScript).

The metadata elements `$totalDigits` and `$fractionDigits` allow a more precise description if it is required.

Example:

- `"exchangeRate": "1.2990"`

7.1.6 sdata/date

The `sdata/date` type is a `string` representing a date. The format of the contents MUST correspond to the "YYYY-MM-DD" representation as defined in the ISO8601 specification [8].

Example:

- `"creationDate": "2014-07-16"`

7.1.7 sdata/time

The `sdata/time` type is a `string` representing the time of day. The format of the contents MUST correspond to the "*extended format*" of "hh:mm:ss" of the ISO8601 specification [8], with an optional fractional part to the seconds. In addition, it is RECOMMENDED that the value contain the time-zone information as recommended by [8], although this may not be appropriate in some cases (for example, times for recurring events in locales using Daylight Savings).

Example:

- `"lastUpdatedTime": "20:30Z"`
- `"lastUpdatedTime": "20:30:12+02:00"`
- `"lastUpdatedTime": "20:30:12.435-01:00"`

7.1.8 sdata/datetime

The `sdata/datetime` type is a `string` representing a time on a particular date. The format of the contents MUST correspond to the ISO8601 specification [8], with the date and time parts following the SData requirements above. In addition, SData REQUIRES that the value contain the time-zone information as recommended by the standard.

Examples:

- `"invoicePrintedAt": "2014-07-16T19:20:30+1:00"`
- `"invoicePrintedAt": "2014-07-16T19:20:30Z"`

7.2 Complex types

Complex types describe properties that are either JSON objects or containers of objects and properties.

The `$item` property of a complex type is a JSON object of the type that comprises the metadata (including the type) for the contained object. The `$item` property MUST be present for all complex types.

7.2.1 sdata/choice

The `sdata/choice` type describes an enumeration. The `$item` property MUST contain a `$type` and a `$enum` property. The `$enum` property is an array of JSON objects delivering the enumeration elements. Each JSON object in `$enum` MUST contain a `$value` property which defines the value of the enumeration elements.

Example: The property `status` taking on values "`ready`", "`pending`" or "`done`" would be expressed as:

```
{
      "status" : {
            "$type" : "sdata/choice",
            "$item" : {
                  "$type" : "sdata/string",
                  "$enum" : [
                        { "$value" : "ready", "$title" : "READY" },
                        { "$value" : "pending", "$title" : "PENDING" },
                        { "$value" : "done", "$title" : "DONE" }
                  ]
            }
      }
}
```

The payload for a valid resource could look like:

```
{
     "status" : "ready"
}
```

7.2.2 sdata/array

The `sdata/array` type specifies that the property is a JSON array. The elements of the array are specified within `$item` and can be of any type. The `$item` property MUST be present.

Example: A simple object containing an array of tags can be expressed as:

```
{
    "firstName": { "$type": "sdata/string" },
    "lastName":  { "$type": "sdata/string" },
    "...":"...",
    "tags": {
        "$type": "sdata/array",
        "$item": {
            "type": "sdata/string"
        }
    }
}
```

A valid resource payload could look like:

```
{
    "firstName": "John",
    "lastName": "Doe",
    "...":"...",
    "tags": [ "C#", "Java", "Programming" ]
}
```

7.2.3 sdata/reference

The `sdata/reference` type identifies a property that contains a reference to another SData resource. The properties of the referenced resource MAY be included – fully or partially - in the payload. These properties are to be viewed as **read-only**, and consequently any modification on the consumer side to these properties will be ignored by the provider.

The metadata is presented as follows:

- metadata pertaining to the property is defined within the object.
- metadata pertaining to the `sdata/reference` object is defined within the `$item` object.

The referenced resource is pointed to by a `$url` specification located inside the `$item` block. The presence of:

- `$item` is REQUIRED (in the object).
- `$url` is REQUIRED (in the `$item`).
- `$properties` is OPTIONAL.
- `$links` is OPTIONAL but RECOMMENDED (see the next chapter for a discussion on links).

Example: The manager of an `Employee` resource could be represented as follows:

```
{
    "id": { "$type": "sdata/string" },
    "...":"...",
    "manager": {
        "$type": "sdata/reference",
        "$title": "Manager Link",
        "$key": "{$uuid}",
        "$item" : {
            "$url": "{$baseUrl}/users('{$key}')",
            "$title": "Manager Details",
            "$properties": {
                "firstName": { "$type": "sdata/string" },
                "lastName": { "$type": "sdata/string" }
            }
        }
    }
}
```

A valid resource payload could look like:

```
{
    "id": "967-1111",
    "...":"...",
    "manager": {
        "firstName": "John",
        "lastName":  "Doe"
     }
}
```

IMPORTANT:

A consumer can only operate on the relationship with the referenced resources. This means that relationships between enclosing and referenced resources can be created or deleted and then persisted via a PUT/PATCH on the enclosing resource. It is not possible to create a new referenced resource in this context. This is achieved by operating directly (via POST, DELETE) on the resource kind of the reference.[4]

7.2.4 sdata/object

The `sdata/object` type identifies an embedded resource. Please note that the payload of the embedded resource MUST be presented in its entirety within the payload of the enclosing resource.

The metadata is presented as follows:

- metadata pertaining to the property is defined within the object.
- metadata pertaining to the `sdata/object` is defined within the contained `$item` object.

[4] Or by operating via the appropriate links: $create, $delete, $updateFull, $updatePartial as described in the next section.

The presence of:

- `$item` is REQUIRED.
- `$url` is OPTIONAL.
- `$properties` is OPTIONAL.
- `$links` is OPTIONAL but RECOMMENDED (see the next section for a discussion on links).

Example: An `Employee` with an embedded `Address` resource could be represented as follows:

```
{
    "firstName": { "$type": "sdata/string" },
    "lastName": { "$type": "sdata/string" },
    "...":"...",
    "address": {
        "$type": "sdata/object",
        "$title": "Address",
        "$item": {
            "$properties": {
                "street":  { "$type": "sdata/string" },
                "zip":     { "$type": "sdata/string" },
                "city":    { "$type": "sdata/string" },
                "state":   { "$type": "sdata/string" },
                "country": { "$type": "sdata/string",
                             "$format": "country"}
            }
        }
    }
}
```

A valid resource payload could look like:

```
{
    "firstName": "John",
    "lastName":  "Doe",
    "...":"...",
    "address": {
        "street":  "444 High Street",
        "zip":     "92301",
        "city":    "Palo Alto",
        "state":   "California",
        "country": "US"
     }
}
```

IMPORTANT:

Unlike the `sdata/reference` case, the consumer can create, delete, and update the resources embedded as an `sdata/object` in the context of the object. A PUT operation on a resource containing an `sdata/object` property will have the effect of persisting the resource **and** the embedded `sdata/object` resources as they stand (barring, of course, the case of constraint violations on the provider side).

7.3 Other internet media types

Any internet media type outside the ones described in the previous sections MAY be attached to an SData JSON property. The associated value (or object) in the payload is transparent to SData, although it could be meaningful in the context of a particular contract.

Note: if a JSON object is specified using "application/JSON", there is no requirement for the attached JSON object to conform to the SData standard.

Example: An Employee resource referencing a photograph resource could be represented as follows:

```
{

     "$baseUrl": "http://www.example.com/sdata/MyApp/-/-",
     "...":"...",
     "$properties": {
          "firstName": {
              "$type": "sdata/string"
          },
          "lastName": {
              "$type": "sdata/string"
          },
          "photoKey": {
              "$type": "sdata/string",
              "$isHidden": true
          },
          "photograph": {
              "$type": "image/jpeg",
              "$url": "{$baseUrl}/pictures('{photoKey}')"
          }
     },
     "...": "...",
     "firstName": "John",
     "lastName": "Doe",
     "photoKey": "445-C...",
     "photograph": "http://www.example.com/sdata/MyApp/-/-/pictures('445-C...')"
}
```

In the example above, the consumer would be able to retrieve the image associated with an employee by combining the information from the photograph object with the information passed in the payload through the photoKey property.

The $isHidden property indicates that the photoKey property SHOULD NOT be rendered on any user interface.

8 Links

SData Links are JSON objects that provide the information necessary to operate on a resource.[5]

SData Links also convey the operational capabilities on a resource. If links are provided for the creation of a resource, the operation-related specifications like `canPost` (see the SData 1.1 standard, sme.xsd) become redundant. Other capabilities (like `canFilter` for example) are gathered in the `$capabilities` metadata element.

This chapter describes in turn:

- The standard SData links; these cover the CRUD operations on a resource.
- The components of an SData link.
- The links to services and queries related to a resource.

The usage of links and the definition of a `$links` object is OPTIONAL.

8.1 Standard links

SData defines the following standard link objects:

	Description
$create	Link used to create a new resource.
$delete	Link used to delete an existing resource.
$updateFull	Link used to update a resource by providing all its contents.
$updatePartial	Link used to update only a selected subset of the properties of a resource.
$details	Link used to retrieve a single resource (the response will contain an entry).
$list	Link used to retrieve a set of resources (the response will contain a feed).
$lookup	Link used to retrieve a set of resources to be used to populate a list of choices. This corresponds to the sdata:lookup defined in the SData 1.1 specification section 6.7.
$prototype	Link used to retrieve the prototype associated with a representation – see Section 9 for information on prototypes.

Any of the above links MAY be present within an SData payload.

[5] SData Links are a specific embodiment of Roy Fielding's hypermedia controls (enabling "Hypermedia as the Engine of Application State" or HATEOAS); Fielding's indication to REST application designers is that operations (state transfer) be driven by hypertext. Link constructs (as known from HTML and ATOM) appear to be an appropriate means to do this.

8.2 Link structure

An SData Link object consists of several properties aiming to decouple the consumer from a provider. These are:

	Compliance	Description
$url	MUST	The URL pointing to a resource.
$method	MAY	HTTP method used to operate on the resource. - **default : GET**
$title	SHOULD	Localized label/caption for the link.
$id	MAY	MAY be used to identify a link in a specific context. This property MUST be set for prototype links **if** several prototypes are available for a resource.
$body	MAY	The object containing the payload transported to the provider alongside the request.
$invocation	MAY	The manner in which the provider should process the request[6]. Allowable values are: **sync** [**default**], async, syncOrAsync.
$batch	MAY	Expresses the manner of invocations supported in batch by the provider. Allowable values are: `false` [**default**]: indicates batching is not supported, `true`.
$request	MAY	Contains either: 1. URL of a prototype (see Section 10) 2. JSON object containing the description of request's individual parameters. The **omission** of `$request` indicates that the request has **no parameters**. Should parameters exist, their description is found in the contained `$properties` object. An individual parameter object has a unique name and contains the following properties: • `$title` [optional]: human-readable parameter name. • **`$type` [mandatory]**: defines the type of the parameter.
$response	MAY	Contains either 1. URL of a prototype (see Section 10) 2. JSON object containing the description of request's expected response. The description of the properties returned by the provider is found in the enclosed `$properties` object
$type	MAY	The SData type(`application/json;vnd.sage=sdata`) or MIME type of the resource. If omitted, the contract default is returned

[6] This should not be confused with client-side AJAX requests. The mechanism corresponds to the sme:invocationMethod described in section 11.4 and 11.5 of the SData 1.1 standard

Example: Consider an SData object that exposes links for deletion and update operations. This would be expressed as:

```
{
   "$links": {
      "$updateFull": {
         "$title": "Update the resource",
         "$type": "application/json;vnd.sage=sdata",
         "$url": "{$url}",
         "$method": "PUT"
      },
      "$delete": {
         "$title": "Delete this resource",
         "$type": "application/json;vnd.sage=sdata",
         "$url": "{$url}",
         "$method": "DELETE"
      }
   }
}
```

The URL used is the same with the URL of the resource in this case. The {$url} notation indicates a replacement with a resource value as explained in section 6 of this document.

8.3 Service and query links

Links are used to express services and queries delivered by a provider, as defined in the SData specification section 11 and section 12. In this case, the name of the link object corresponds to the desired query/service.

Examples: The following is the link accessing the `createBOM` service returning the bill of materials associated with a sales order.

```
{
   "$links": {
      "createBOM": {
         "$title": "Create Bill of Materials",
         "$url": "{$url}/$service/createBOM",
         "$method": "POST",
         "$response": "{$baseUrl}/$prototypes/createBOM",
         "$invocation": "syncOrAsync"
      }
   }
}
```

In addition to the URL and appropriate HTTP method, the link indicates the structure of the response (by means of a prototype which is discussed later in this document) as well as the fact that the server may choose whether the service is to be performed in a synchronous or an asynchronous manner.

The following is an example showing an SData query associated with a Products resource kind (described in the SData documentation Example of Named Query):

```
{
   "$links": {
      "reOrder": {
         "$title": "List of products to be reordered",
         "$url": "{$url}/$queries/reorder",
         "$method": "GET",
         "$request": {
            "$properties": {
               "family": {
                  "$title": "product category",
                  "$type": "sdata/string"
               },
               "threshold": {
                  "$title": "minimal in-stock threshold",
                  "$type": "sdata/integer"
               }
            }
         },
         "$response": {
            "$type": "sdata/array",
            "$item": {
                "$properties": {
                   "productID": {
                      "$title": "Product ID",
                      "$type": "sdata/string"
                   },
                   "description": {
                       "$title": "Description",
                       "$type": "sdata/string"
                   },
                   "inStock": {
                       "$title": "Quantity in stock",
                       "$type": "sdata/integer"
                   }
                }
            }
         }
      }
   }
}
```

The `reOrder` named query provides a list of products with stock below a certain threshold value. The specification of the query contains the `$request` object that contains a description of the two parameters accepted by the query, namely `family` and `threshold`. It also describes the structure of the return with the three properties: `productID`, `description` and `inStock`.

9 Embedded metadata

SData payloads contain embedded metadata elements of two kinds:

- SData-defined elements: used throughout this document and collected in Appendix A.
- Application specific metadata elements.

Metadata inclusion in a JSON response is governed by the following rules:

- SData metadata for an object is presented as properties of the objects, along-side the native properties of the object. Given the leading **$**, it is possible to distinguish between the two.
- SData metadata pertaining to properties of an object is collected in a `$properties` object within the object itself. The structure of the $properties is as follows:
  - properties of the `$properties` JSON object mirror the native properties of the original object (and have the same name).
  - the contents are objects enclosing the metadata elements pertaining to corresponding native properties.
- The capability metadata elements (canRead, canUpdate, canDelete, canCreate) are replaced in their functionality by **links**.

Embedded metadata SHOULD be an exception, meant to override/extend the information contained in the prototype (see section 10 for more details on prototypes). In this manner, it is possible to significantly reduce the volume of information transferred from the provider to the consumer.

Example: Consider a Product object with the following structure and data:

```
Product
• name          -> "iPhone"
• ID            -> "4711"
• unitPrice     ->  459.00
• stock         -> "available"  [read only]
```

In this example, the value of the `stock` property is computed by the underlying application and therefore `readOnly`. Retrieving this product by means of a GET operation on …`/Products('4711')` we should get the following format in JSON for the product in question:

```
{
      "$url" : "https://www.example.com/sdata/myapp/-/-/products('4711')",
      "$key" : "4711",
      "name" : "iPhone",
      "ID" : "4711",
      "unitPrice" : 459.00,
      "stock" : "available",
      "$properties" : {
               "stock" : { "$isReadOnly" : true}
      }
}
```

The result shows the SData required elements `$url` and `$key` embedded at the same level as the native properties of the object. Additionally, the `$properties` contains a `stock` object with the name-value pair passing the `$isReadOnly` attribute value to the caller.

9.1 $properties object

The `$properties` object encapsulates the properties of the resource kind as they would be returned by a GET operation in JSON, and has the following characteristics:

- The object structure is maintained, exactly matching the JSON structure in the payload.
- An individual property is a JSON object whose name does NOT start with $ and contains:
  - The metadata for the object represented as name-value pairs. The name will start with a **$**.
- Each property MUST have an associated `$type` specification.

Example:

Consider the following Address resource kind with metadata enclosed in square brackets:

- `ID: [integer, Hidden, Mandatory]`
- `Street: [string, title="Street", Mandatory]`
- `StreetNumber: [integer, title="Number"]`
- `City: [string, title="City", Mandatory]`
- `PostalCode: [string, title="ZipCode", Mandatory]`
- `Country: [referenceToCountryResourceKind, Mandatory]`
  - `Name: [string]`
  - `ISOCode: [string]`
  - `[url="http://www.example.com/sdata/MyApp/-/-/countries"]`

This would result in the following `$properties` object:

```
{
   "$properties": {
      "ID": {
         "$title": "AddressId",
         "$type": "sdata/integer",
         "$isMandatory": true,
         "$isHidden": true
      },
      "Street": {
         "$title": "Street",
         "$type": "sdata/string",
         "$isMandatory": true
      },
      "StreetNumber": {
         "$title": "Number",
         "$type": "sdata/integer"
      },
      "City": {
         "$title": "City",
         "$type": "sdata/string",
         "$isMandatory": true
      },
      "PostalCode": {
         "$title": "ZipCode",
         "$type": "sdata/string",
         "$isMandatory": true
      },
```

```
      "Country": {
         "$title": "Country",
         "$type": "sdata/reference",
         "$links": {
             "$prototype": {
                 "$id": "lookup",
                 "$url": "{$baseUrl}/$prototypes/countries('{$id}')",
                 "$title": "Prototype of Country thumbnail"
             }
         },
         "$isMandatory": true,
         "$item": {
            "$url": "http://www.example.com/sdata/MyApp/-/-/countries('{ISOCode}')",
            "$properties": {
               "Name": {
                  "$title": "Country name",
                  "$type": "sdata/string",
                  "$isMandatory": true
               },
               "ISOCode": {
                  "$title": "Country code",
                  "$type": "sdata/string",
                  "$format": "country",
                  "$isMandatory": true
               }
            }
         }
      }
   }
}
```

In the above, the scalar properties (`ID`, `Street`, …) are objects containing respectively the `$title`, `$type` and, if appropriate, the `$isMandatory` metadata.

The `Country` property is a reference to the `countries` resource kind. Accordingly, it has a type of `"sdata/reference"` and a `$url` specification. The metadata for the sub-properties of `Country` (i.e. `Name` and `ISOCode`) are defined in the body of the `$item` object.

10 SData prototypes

Prototypes play a role roughly corresponding to schemas in SData v1.1; the advantage is that they are expressed as SData JSON objects and are more flexible and compact than XML Schema Definition documents. Prototypes define the native and metadata properties of a resource kind, specifying the corresponding types and, for metadata, the default values.

The following subsections introduce the SData prototypes by:

- describing the format and function of the `$prototype` object,
- indicating how a `$prototype` object can surface in a response,
- showing how prototypes can be retrieved individually.

A prototype is a resource that bundles the metadata of a resource kind representation. If metadata is provided, the usage of prototypes in the JSON context is **strongly RECOMMENDED** but **not mandatory**.

10.1 $prototype object

A prototype is a JSON object and is formed according to the rules laid out in the document "JSON formatting of SData responses". Metadata elements that appear in a prototype are:

	Compliance	Description
$properties	MUST	contains the metadata for all the individual properties of an object
$links	MAY	contains the elements describing the possible operations for an element (see Section 8 of this document)

Example:

Consider the following Address resource kind with metadata enclosed in square brackets:

- `ID: [integer, Mandatory]`
- `Street: [string, title="Street", Mandatory]`
- `StreetNumber: [integer, title="Number"]`
- `City: [string, title="City", Mandatory]`
- `PostalCode: [string, title="ZipCode", Mandatory]`
- `Country: [referenceToCountryResourceKind, Mandatory]`
  - `Name: [string]`
  - `ISOCode: [string]`
  - `[url="http://www.example.com/sdata/MyApp/-/-/countries('{ISOCode}')"]`

The prototype describing this resource is shown below:

```
{
   "$baseURL" : "http://www.example.com/sdata/MyApp/-/-",
   "$properties": {
      "ID": {
         "$title": "AddressId",
         "$type": "sdata/integer",
         "$isMandatory": true
      },
      "Street": {
```

```
         "$title": "Street",
         "$type": "sdata/string",
         "$isMandatory": true
      },
      "StreetNumber": {
         "$title": "Number",
         "$type": "sdata/integer"
      },
      "City": {
         "$title": "City",
         "$type": "sdata/string",
         "$isMandatory": true
      },
      "PostalCode": {
         "$title": "ZipCode",
         "$type": "sdata/string",
         "$isMandatory": true
      },
      "Country": {
         "$title": "Country",
         "$type": "sdata/reference",
         "$links": {
             "$prototype": {
                 "$title": "Country list prototype",
                 "$id" : "lookup",
                 "$url": "{$baseUrl}/$prototypes/countries('{$id}')"
             }
         },
         "$isMandatory": true,
         "$item": {
            "$url": "http://www.example.com/sdata/MyApp/-/-/countries('{ISOCode}')",
            "$properties": {
               "Name": {
                  "$title": "Country name",
                  "$type": "sdata/string",
                  "$isMandatory": true
               },
               "ISOCode": {
                  "$title": "Country code",
                  "$type": "sdata/string",
                  "$format": "country",
                  "$isMandatory": true
               }
            }
         }
      }
   },
    "$links": {
       "$updateFull": {
            "$title": "Update the resource",
            "$type": "application/json;vnd.sage=sdata",
            "$url": "{$url}",
            "$method": "PUT"
       },
       "$prototype": {
           "$title" : "Customer address prototype",
```

```
            "$id" : "detail",
            "$url" : "{$baseURL}/prototypes/addresses('{$id}')"
        }
    }
}
```

10.2 Prototypes exposed by an application: the $prototypes URL segment

The prototypes of an application are SData JSON resources retrievable by a GET operation. The URL segments of a prototype are formed according to the pattern:
…/`$prototypes/[<resourceKindName>[('<prototypeId>')]]` where:

- `$prototypes`: is a reserved segment located at the resource kind level[8]. This MUST be supported if prototype resources are present.
- *`resourceKindName`*: is the name of the resource whose prototype it is. This SHOULD be supported if at least one prototype is available for a resource kind.
- *`prototypeId`*: MAY be present. It is the identifier for the prototype and relates to the representation of a resource[9].

10.3 Retrieving the prototype of a resource kind

The prototype of a resource kind is intimately related to the representation of a resource. This means that it is possible to have several prototypes, each describing individual representations of a resource. This is easy to see when looking at the differences between a feed of resources and an individual resource: in the first case the information is succinct, while in the second it would be rather extensive. Another example is the prototype for a representation of a resource in a mobile context (where bandwidth and screen area are prime assets) compared with that for a desktop or full-browser client.

Prototypes are reasonably static. This means that they should be retrieved once, cached and then applied many times. Consequently, a versioning mechanism (`eTag` or `modifiedDate`) would greatly benefit the client-side handling of prototype and therefore providers SHOULD support such a mechanism.

Section 4 of this document described the means for retrieving prototypes. The remainder of this section provides more details on retrieving multiple prototypes via links.

[8] For more information, see the discussion in the SData URL chapter of the "SData 2.0 – Core" document
[9] It is important to note that there may be several prototypes associated with a single resource kind. For example, an application could distinguish between information delivered for a feed of a resource kind, that of an individual entry and yet again to that of a resource at creation.

For example:

- `GET` http://www.example.com/sdata/MyApp/myContract/-/$prototypes/addresses

would provide a response similar to:

```
{
   "$baseURL" : "http://www.example.com/sdata/MyApp/-/-",
   "$url" : "{$baseURL}/prototypes/addresses",
   "$title" : "all Address prototypes",
   "$resources" : [
   {
      "$id": "detail",
      "$prototype": {
          "$url" : "{$baseURL}/addresses('{$ID}')",
          "$properties": {
             "ID": {
                "$title": "AddressId",
                "$type": "sdata/integer",
                "$isMandatory": true
             }
          },
          "...":"...",
          "$links": {
              "$prototype": {
                  "$id": "detail",
                  "$url": "{$baseURL}/prototypes/addresses('{$id}')",
                  "$title": "Customer address prototype"
              }
          }
       }
   },
   {
      "$id": "list",
      "$prototype": {
          "$url" : "{$baseURL}/addresses('{$ID}')",
          "$properties": {
             "ID": {
                "$title": "AddressId",
                "$type": "sdata/integer",
                "$isMandatory": true
             }
          },
          "...":"...",
          "$links": {
              "$prototype": {
                  "$id": "list",
                  "$url": "{$baseURL}/prototypes/addresses('{$id}')",
                  "$title": "Customer address feed"
              }
          }
       }
   }
   ]
}
```

The GET operation on the reserved $prototypes segment will return links to all the prototypes exposed by the application. The returned payload contains:

- One array property for every resource kind where prototypes are available;
- The array element contains at least the following properties pertaining to the prototype:
  - `$url`
  - `$resourceKind`
  - `$id`
  - `$title`

Example:

- GET `http://www.example.com/sdata/MyApp/-/-/$prototypes`

Would return a payload similar to:

```
{
   "$baseURL" : "http://www.example.com/sdata/MyApp/-/-",
   "$title" : "Links to all prototypes",
   "$totalResults" : 32,
   "$startIndex" : 1,
   "$itemsPerPage" : 10,
   "$resources" : [
      {
         "$title" : "Address entry prototype",
         "$resourceKind" : "address",
         "$url" : "{$baseURL}/prototypes/addresses('detail')",
         "$id" : "detail"
      },
      {
         "$title" : "Address feed prototype",
         "$resourceKind" : "addresses",
        "$url" : "{$baseURL}/prototypes/addresses('list')",
         "$id" : "list"
      },
      {
         "$title" : "Customer entry prototype",
         "$resourceKind" : "customer",
         "$url" : "{$baseURL}/prototypes/customers('detail')",
         "$id" : "detail"
      },
      {
         "$title" : "Customer feed prototype",
         "$resourceKind" : "customers",
         "$url" : "{$baseURL}/prototypes/customers('list')",
         "$id" : "list"
      }
   ]
}
```

10.4 Merge process

For the consumer of a JSON formatted response that leverages metadata, objects are obtained in their entirety by merging the prototype with the payload information. The information in the payload has precedence and overlays/overrides the prototype definitions.[11]

The interplay between prototype and payload has several key attributes:

- Metadata is expressed in JSON.
- Metadata is available at the resource kind level and ideally even finer-granular levels.
- Ability to override metadata at any level (feed/entry/property).
- Reduce verbosity of transferred information.

The merge process is a conceptual process, meaning that a consumer will likely use a variety of local techniques to efficiently implement it while maintaining the same overall effect.

Example: Consider the following prototype:

```
{
   "$baseUrl": "http://www.example.com/sdata/MyApp/-/-",
   "$url": "{$baseUrl}/addresses",
   "$title": "Address list",
   "$properties": {
      "ID": {
         "$title": "AddressId",
         "$type": "sdata/integer",
         "$isMandatory": true
      },
      "Street": {
         "$title": "Street",
         "$type": "sdata/string",
         "$isMandatory": true
      },
      "StreetNumber": {
         "$title": "Number",
         "$type": "sdata/integer"
      },
      "City": {
         "$title": "City",
         "$type": "sdata/string",
         "$isMandatory": true
      },
      "PostalCode": {
         "$title": "ZipCode",
         "$type": "sdata/string",
         "$isMandatory": true
      },
      "Country": {
         "$title": "Country",
         "$type": "sdata/reference",
         "$links": {
            "$prototype": {
```

[11] To remove a metadata element defined in the prototype, the payload defines the property with `null` value.

```
                "$id": "lookup",
                "$url": "{$baseUrl}/$prototypes/countries('{$id}')",
                "$title": "Country lookup prototype"
             }
         },
         "$url": "http://www.example.com/sdata/MyApp/-/-/countries('{ISOCode}')",
         "$isMandatory": true,
         "$item": {
            "$properties": {
               "Name": {
                  "$title": "Country name",
                  "$type": "sdata/string",
                  "$isMandatory": true
               },
               "ISOCode": {
                  "$title": "Country code",
                  "$type": "sdata/string",
                  "$isMandatory": true
               }
            }
         }
   },
   "$links": {
       "$prototype": {
          "$id": "list",
          "$url": "{$baseUrl}/$prototypes/addresses('{$id}')",
          "$title": "Address feed prototype"
       }
     }
   }
}
```

Matching the previous prototype is the following JSON formatted payload:

```
{
   "$baseUrl": "http://www.example.com/sdata/MyApp/-/-",
   "$url": "{$baseUrl}/addresses?creditLimitExceeded=true",
   "$title": "Addresses of accounts with exceeded credit limit",
   "$resources": [
      {
         "ID": "7123a",
         "Street": "Lerchenweg",
         "StreetNumber": 11,
         "PostalCode": 71711,
         "City": "Marbach am Neckar",
         "Country": {
            "Name": "Germany",
            "ISOCode": "DE"
         },
         "$properties": {
            "PostalCode": {
               "$isMandatory": false
            }
         }
      },
```

```
      {
         "ID": "hw7631",
         "Street": "Fleet Street",
         "StreetNumber": 31,
         "City": "London",
         "PostalCode": "EC4Y 8EQ",
         "Country": {
            "Name": "United Kingdom",
            "ISOCode": "GB"
         }
      }
   ]
}
```

The above payload provides, in addition to the payload (in blue) a series of specific metadata (in yellow background) for:

- The URL of the feed (`$url`)
- The title of the feed (`$title`)
- The type of the first – German – address, that must be numeric according to German rules

After the merge process, the logical JSON object will contain the following:

```
{
   "$baseUrl": "http://www.example.com/sdata/MyApp/-/-",
   "$url": "{$baseUrl}/addresses?creditLimitExceeded=true",
   "$title": "Addresses of accounts with exceeded credit limit",
   "$resources": [
      {
         "ID": "7123a",
         "Street": "Lerchenweg",
         "StreetNumber": 11,
         "PostalCode": 71711,
         "City": "Marbach am Neckar",
         "Country": {
            "Name": "Germany",
            "ISOCode": "DE"
         },
         "$properties": {
          "ID": {
              "$title": "AddressId",
              "$type": "sdata/integer",
              "$isMandatory": true
          },
         "Street": {
             "$title": "Street",
              "$type": "sdata/string",
              "$isMandatory": true
         },
         "StreetNumber": {
              "$title": "Number",
              "$type": "sdata/integer"
         },
         "City": {
```

```
            "$title": "City",
            "$type": "sdata/string",
            "$isMandatory": true
        },
        "PostalCode": {
            "$title": "ZipCode",
            "$type": "sdata/string",
            "$isMandatory": false
        },
        "Country": {
            "$title": "Country",
            "$type": "sdata/reference",
            "$links": {
                "$prototype": {
                    "$id": "lookup",
                    "$url": "{$baseUrl}/$prototypes/countries('{$id}')",
                    "$title": "Country lookup prototype"
                }
            },
            "$url": "http://www.example.com/sdata/MyApp/-/-/countries('{ISOCode}')",
            "$prototype": "{$baseUrl}/$prototypes/countries('lookup')",
            "$isMandatory": true,
            "Name": {
                  "$title": "Country name",
                  "$type": "sdata/string",
                  "$isMandatory": true
            },
            "ISOCode": {
                  "$title": "Country code",
                  "$type": "sdata/string",
                  "$isMandatory": true
            }
        }
      },
      "$links": {
            "$prototype": {
                "$id": "list",
                "$url": "{$baseUrl}/$prototypes/addresses('{$id}')",
                "$title": "Address feed prototype"
            }
       }
    },
    {
     "ID": "hw7631",
     "Street": "Fleet Street",
     "StreetNumber": 31,
     "City": "London",
     "PostalCode": "EC4Y 8EQ",
     "Country": {
         "Name": "United Kingdom",
         "ISOCode": "GB"
     },
     "$properties": {
       "...": "...",
       "PostalCode": {
          "$title": "ZipCode",
```

```
            "$type": "sdata/string",
            "$isMandatory": true
        }
      },
      "$links": {
          "$prototype": {
              "$id": "list",
              "$url": "{$baseUrl}/$prototypes/addresses('{$id}')",
              "$title": "Address feed prototype"
          }
      }
    }
  ]
}
```

Please note that the `$type` of the `PostalCode` property of the first address object is `'sdata/integer'` as overridden by the provider.

11 Compliance

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, “Key words for use in RFCs to Indicate Requirement Levels” [9].

A provider SHOULD consider metadata support in its payloads. This has the advantage of supporting for more dynamic/flexible clients that usually rely on metadata to interact with users. However, if the particular use case does not require metadata and its support would burden the development effort, a provider need not implement it.

If metadata is supported, the provider MAY choose to support prototypes but MUST support embedded metadata .

If a prototype for the targeted resource exists, the provider MUST return it in the payload for a GET request with the `'?includePrototype=true'` specification; if no prototype exists, the specification has no effect.

The amount of metadata returned is a provider specific decision. A reasonable expectation is that, if prototypes are supported, the embedded metadata would consist only of overrides to the prototype.

A consumer MAY leverage the metadata existent in a response. If it does so then, unless otherwise specified in the underlying contract:

- If a prototype exists, then this MUST be retrieved and the merge process MUST be applied.
- The substitution process MUST be applied.

References

Number	Title	Version	Date	Author
1	**SData** *Welcome to SData*	1.1	2011	Sage Group plc
2	**RFC 4627** *The application/json Media Type for JavaScript Object Notation (JSON)*	Informational	July 2006	Internet Engineering Task Force
3	**RFC 6570** *URI Template*	Proposed Standard	March 2012	Internet Engineering Task Force
4	**RFC 5322** *Internet Message Format*	Draft Standard	October 2008	Internet Engineering Task Force
5	**ISO 4217** *Codes for the representation of currencies and funds*	-	2008	International Organization for Standardization
6	**RFC 2616** *Hypertext Transfer Protocol -- HTTP/1.1*	Standard	June 1999	Internet Engineering Task Force
7	**ISO 3166-1** *Codes for the representation of names of countries and their subdivisions – Part 1: Country codes*	-	2006	International Organization for Standardization
8	**ISO 8601** *Data elements and interchange formats – Information interchange – Representation of dates and times*	Third Edition	03-Dec-2004	International Organization for Standardization
9	**RFC 2119** *Key words for use in RFCs to Indicate Requirement Levels*	Best Current Practice		Internet Engineering Task Force

Appendix A: Metadata elements for JSON

SData for JSON recognizes a set of metadata elements presented in the table below. Any application can extend this set with native metadata elements that should be described in their respective contracts.

It should be noted that some metadata elements in the table below are not referenced directly in this document. In these cases, the metadata elements are drawn from SData v1.1, and are applicable to the SData 2.0 JSON format. For further explanation of these elements, the reader is referred to [1].

Name	Description	Applicable to	Point of definition
$applicationCode	JSON diagnosis related: Application specific diagnosis code	Diagnoses object	JSON format doc
$averageLength	Contains the average display length needed	Property	Sme.xsd
$baseUrl	URL leading to the resource kind level of an application. The URL SHOULD NOT end in a "/"; this is to aid readability when $baseUrl is concatenated with other strings to form new URLs. **Examples**: *"$baseUrl": "http://www.example.com/MyApp/Contract/-"* *"$url" : "{baseUrl}/$services/pricingService"*	Feed	JSON format doc
$batch	expresses the manner of invocations supported in batch by the provider. Allowable values are: • **`false`** [**default**]: indicates batching is not supported • `true`	Request	JSON metadata doc (here)
$batchingMode	Batching is supported for the resource kind. Allowable values are: sync, async, syncOrAsync	Links	Sme.xsd
$body	JSON object containing the payload transported to the provider from an SData link	Links	JSON metadata doc (here)
$capabilities	String element containing a comma separated list of the resource's capabilities. These may contain • **`filter`**: corresponds to `canFilter` sme attribute • **`group`**: corresponds to `canGroup` sme attribute • **`search`**: corresponds to `canSearch` sme attribute • **`sort`**: corresponds to the `canSort` sme attribute	Links	JSON metadata doc (here)
$create	Link used to create a new resource.	Links	JSON metadata doc (here)
$delete	Link used to delete an existing resource	Links	JSON metadata doc (here)
$deleteMissing	Indicates that all elements not present in an array must be considered as deleted	Request	sdata.xsd
$details	Link used to retrieve a single resource (response is an entry)	Links	JSON

			metadata doc (here)
$diagnoses	Array containing objects providing detailed indications of errors and warnings encountered by the provider during the execution of a request	Response	JSON format doc
$elapsedSeconds	JSON tracking related: Time elapsed since operation started, in seconds.	Tracking response	JSON format doc
$enum	Contains the allowable values of an enum property (see sdata/choice type)	Property	JSON metadata doc (here)
$etag	opaque identifier assigned by the provider to a version of a resource	Response	JSON format doc
$format	Is associated with sdata/string properties; imposes a particular structure to the string	Property	JSON metadata doc (here)
$fractionDigits	Number of digits after the decimal point	Property	Sme.xsd
$groupName	A group (category) name to group related properties	Property	Sme.xsd
$invocation	The manner in which the provider should process the request[12]. Allowable values are: • **sync** [default], • **async**, • **syncOrAsync**.	Links	JSON metadata doc (here)
$isDeleted	Indicates that a member of an array has been deleted	Response	sdata.xsd
$isLocalized	The property contains localized text	Property	Sme.xsd
$isMandatory	The property cannot have an empty content	Property	Sme.xsd
$isReadOnly	The contents of the property cannot be modified	Property	Sme.xsd
$isUniqueKey	The property contains a unique key	Property	Sme.xsd
$item	Contains the description of an individual element in an sdata/array, sdata/choice, sdata/reference, sdata/object	Entry definition	JSON metadata doc (here)
$key	The native primary key of the resource. Its value can be used to target a single resource (ex: http://www.example.com/MyApp/-/myResource('123'))	Property	JSON format doc
$links	Object containing links that present functional aspects of the resource (ex: edit, lookup, create).	Entry	JSON format doc JSON metadata doc (here)
$list	Link used to retrieve a set of resources (response is a feed)	Links	JSON metadata doc (here)

[12] This should not be confused with client-side AJAX requests. The mechanism corresponds to the sme:invocationMethod described in section 11.4 and 11.5 of the SData 1.1 standard

$lookup	Link used to retrieve a set of resources to be used to populate a list of choices. This corresponds to the `sdata:lookup` defined in the SData specification section 6.7	Links	JSON metadata doc (here)
$maxLength	Contains the maximum length of a string	Property	Sme.xsd
$message	JSON diagnosis related: Friendly message for the diagnosis	Diagnosis object	JSON format doc
$method	HTTP method used to operate on a resource	Links	JSON metadata doc (here)
$payloadPath	JSON diagnosis related: XPath expression that refers to the payload element responsible for the error	Diagnosis object	JSON format doc
$phase	JSON tracking related: End user message describing the current phase of the operation.	Tracking response	JSON format doc
$phaseDetail	JSON tracking related: Detailed message for the progress within the current phase.	Tracking response	JSON format doc
$pluralName	Name of a resource in plural form	Feed	Sme.xsd
$pollingMillis	JSON tracking related: Delay (in milliseconds) that the consumer should use before polling the service again.	Tracking response	JSON format doc
$precedence	Controls the visibility of properties on small screens	Property	Sme.xsd
$progress	JSON tracking related: Percentage of operation completed.	Tracking response	JSON format doc
$properties	JSON object containing the metadata in a per-property manner for a resource	Entry	JSON metadata doc (here)
$protocolFilters	Comma separated list of properties that can be used for filtering within the `where` clause of the URL	Feed	Sme.xsd
$remainingSeconds	JSON tracking related: Expected remaining time, in seconds.	Tracking response	JSON format doc
$request	Contains either 3. URL of a prototype (see Section 9) 4. JSON object containing the description of request's individual parameters. The **omission** of `$request` indicates that the request has **no parameters**. Should parameters exist, their description is found in the contained `$properties` object. An individual parameter object has a unique name and contains the following properties: • `$title` [optional]: human-readable parameter name • **`$type` [mandatory]**: defines the type of the parameter	Links	JSON metadata doc (here)

$resources	Array containing the individual entries	Feed	JSON format doc
$response	Contains either 1. URL of a prototype (see Section 9) 2. JSON object containing the description of request's expected response. The description of the properties returned by the provider is found in the enclosed `$properties` object	Feed/entry	JSON metadata doc (here)
$sdataCode	JSON diagnosis related; The SData diagnosis code	Diagnosis object	JSON format doc
$severity	JSON diagnosis related: Severity of the diagnosis entry. Possible values are: • **`info`** • **`warning`** • **`transient`** • **`error`** • **`fatal`**	Diagnosis object	JSON format doc
$stackTrace	JSON diagnosis related: Stack trace – to be used with care	Diagnosis object	JSON format doc
$tags	Comma separated list of tags		Sme.xsd
$title	Human-readable description of the resource	Feed/entry	JSON format doc
$totalDigits	Maximum overall number of digits for a decimal property	Property	Sme.xsd
$type	SData JSON type of a property	Property	JSON metadata doc (here)
	For a Link, the $type identifies the internet media type of the response.	Links	JSON metadata doc (here)
$unsupported	The element is an unsupported part of a global contract	Feed/entry /property	Sme.xsd
$updated	the time stamp of the last update of the resource, formatted according to ISO 8601 dateTime specification	Entry	JSON format doc
$updateFull	Link used to update a resource by providing all its contents	Links	JSON metadata doc (here)
$updatePartial	Link used to update only a selected subset of the properties of a resource	Links	JSON metadata doc (here)
$url	URL pointing to the resource	Feed/entry	Sdata.xsd, JSON format

			doc
$uuid	UUID identifying the resource. UUIDs may complement the native primary keys; they are frequently encountered in application integration to identify the same logical resource across application boundaries.	Entry	JSON format doc
$value	Introduces the value of a property in an enum context (see sdata/choice type)	Property	JSON metadata doc (here)