JSON formatted SData responses

Version: 1.0

Purpose

This paper describes the JSON format for SData content. The focus is placed on the structural aspects of content and representation of the protocol-mandated information. The details of metadata usage are handled in a separate document.

The document is aimed at a technical audience who has had a prior exposure to the SData protocol.

The information in this document has not been approved by the standard body governing SData. As such, it should be treated as a preview and a snapshot to the upcoming JSON aspect in SData.

Introduction to the JSON formalism

JSON is an open, text-based data exchange format (see RFC 4627). Like XML, it is human-readable, and platform independent. Data formatted according to the JSON standard is lightweight and can be parsed by JavaScript implementations with incredible ease, making it an ideal data exchange format for web applications.

The information transported through JSON is always in form of the following basic types:

- strings: double-quoted Unicode (UTF-8 by default), with backslash escaping
- numbers
- Boolean values: `true` or `false`
- `null`

The JSON building block is the `name-content` tuple. The `name` is always a quote-enclosed string and is separated from the content by a ":" character. The `content` of a JSON tuple can be either:

- A value in one of the basic types
  Example : `"firstName" : "Ted"`
- An **ordered** set of contents, delimited by square brackets ("[" "]")
  Example: `"continents" : [ "Europe" , "Africa", "Asia", "Americas", "Australia", "Antarctica" ]`
- A JSON object. JSON objects are **unordered** sets of name-content pairs. The only restriction is that within a set the names are unique. The name-content pairs are separated by a comma (",") and enclosed by curly brackets ("{" "}")
  Example: `{ "street" : "Augartenstrasse", "number" : 1, "city" : "Karlsruhe", "country" : "Germany"}`

This is it! JSON is simple, yet very expressive and also less verbose than XML.

Formatting native objects in JSON

Representing a native data container in JSON is a straightforward task, guided by the following rules:

- Scalar properties and their values are represented as JSON name-value pairs
- Structural elements and sub-objects are represented as JSON objects

- Collections are represented as JSON arrays

Expressing a simple data object in JSON

Let us consider a simplified `SalesOrderLine` object of the following structure and values:

- `productName -> "iPad"`
- `productID   -> 437`
- `lineNumber  -> 1`
- `orderedQuantity -> 2`
- `unitPrice   -> 399.95`

We note that the object contains scalar properties that translate easily into JSON name-value pairs. Thus, the JSON representation would be:

```
{
    "productName" : "iPad",
     "productID"   : "437",
     "lineNumber"  : 1,
     "orderedQuantity" : 2,
     "unitPrice"   : 399.95
}
```

Expressing a data object with embedded structures in JSON

Let us alter the example above by grouping the product-related properties in an own structure:

- `product`
  - `name         -> "iPad"`
  - `ID           -> "437"`
- `lineNumber      -> 1`
- `orderedQuantity -> 2`
- `unitPrice       -> 399.95`

In this case, the product structure would be represented as a JSON object in its own right. Thus, the JSON representation is:

```
{
     "product" : {
         "productName" : "iPad",
         "productID"   : "437"
      },
      "lineNumber"  : 1,
      "orderedQuantity" : 2,
      "unitPrice"   : 399.95
}
```

Expressing an object with an embedded collection

A usual example of an object containing a collection of data elements is the SalesOrder. It contains a collection of line items for the individual positions in the order itself. A simplified SalesOrder structure is:

- orderDate
- shipDate
- contact
  - firstName -> "John"
  - lastName -> "Doe"
  - email -> "john.doe@acme.com"
- orderLines – *containing*
    - lineNumber -> 1
    - orderedQuantity -> 2
    - unitPrice -> 399.95
    - product
      - name -> "iPad"
      - ID -> 437

  and
    - lineNumber -> 2
    - orderedQuantity -> 1
    - unitPrice -> 323.00
    - product
      - name -> "Samsung Galaxy S2"
      - ID -> 932
  - subtotal -> 1021.95

In the translation we observe the following:

- `orderDate`, `shipDate` and `subtotal` are scalar properties that should be represented as JSON name-value pairs
- `contact` is a structural element, and therefore to be represented as a JSON object. The contact properties `firstName`, `lastName` and `email` are scalar properties expressed as name-value JSON pairs
- `orderLines` is a collection, and should be represented as a JSON array. The array elements, the individual line items are structural elements translating into JSON objects.
- `product` is a structural element, mapping to a JSON object
- All remaining properties are scalar, hence representable as name-value JSON pairs.

The resulting JSON representation is:

```
{
    "orderDate" : "2001-07-01" ,
    "shippedDate" : null,
    "contact" : {
```

```
        "firstName" : "John",
        "lastName"  : "Doe",
        "email"     : "john.doe@acme.com"
    }, // end contact
    "orderLines" :
        [ {
            "lineNumber"      : 1,
            "orderedQuantity" : 2,
            "unitPrice"       : 399.95,
            "product" : {
                  "name" : "iPad",
                  "ID"   : "437"
            } //end product
          }, {
            "lineNumber"      : 2,
            "orderedQuantity" : 1,
            "unitPrice"       : 323.00,
            "product" : {
                  "name" : "Samsung Galaxy S2",
                  "ID"   : "932"
            } //end product
          }
        ], // end orderLines
    "subtotal" : 1021.95
}
```

Requesting JSON formatted SData content

The JSON and atom+xml formats are of equal importance from a protocol perspective - a difference to the SData 1.x . Consequently, the desired format of a response is specified by the consumer in its request.

If a consumer wants to be certain to get a response in the SData JSON format, it will request this using the "`application/json;vnd.sage=sdata`" media type. This is achieved in two ways:

1. through the HTTP Accept header:

```
Accept: application/json;vnd.sage=sdata
```

2. using `format` query parameter on the request URL:

```
http://www.example.com/sdata/myApp/myContract/prod/accounts?format=application/json;vnd.sage=sdata
```

The first mechanism should be used when the consumer (the user agent) will systematically request the JSON format. The second one is more appropriate when the consumer normally uses ATOM but switches to JSON occasionally.

Default formats of a contract

A contract may be particularly suitable to one or the other formats (ex: a contract for mobile devices will very likely return JSON). In such cases, for expedience and ease of programming, a default response format can be defined at the contract level. The default format will be returned when requests without a specific media type are sent to a provider.

The SData JSON format MAY be returned by an application by default when:

- it is the default format for contract
- JSON responses are per default conformant to the SData structural requirements

JSON responses

Provider responses to requests in (and for) JSON are in one of the following forms:

- An **entry**: this is the representation of an individual resource and contains the native, JSON formatted objects
- A **feed**: is a collection of entries
- A **diagnosis** : returned in case something went wrong with the request
- A **tracking object**: returned for an asynchronous call to enable subsequent polling of results

JSON entries

Entries encompass single resources such as a specific `Customer` or `SalesOrder`. Payloads will contain a single entry only when the request operates target an individual resource (ex: GET on a single resource URL or PUT/POST/DELETE operations).

In addition to the native properties of the resource, SData allows within an entry the presence of several protocol-defined properties meant to ease consumption. These are described in the table below:

	Description	Compliance
$url	URL pointing to the resource. The URL should be represented as relative to the value of the `$baseUrl` of the enclosing feed. ***Examples:*** *given the "$baseUrl": "http://ex.com/MyApp/-/-/"* • *"$url": "customers('1234')"* • *"$url": "customers?where=$uuid eq 'ab1C43sd-c1asd-c2sT'"* If a `$baseUrl` property is not specified, then the URL MUST be absolute. ***Examples:*** • *"$url": "http://ex.com/MyApp/-/-/customers('1234')"* • *"$url": "http://ex.com/MyApp/-/-/customers?where=$uuid eq 'ab1C43sd-c1asd-c2sT'"*	MAY

$key	the native primary key identifying the resource	MAY
$uuid	UUID identifying the resource	MAY
$title	humanly readable description of the resource	MAY
$updated	the time stamp of the last update of the resource, formatted according to ISO 8601 dateTime specification	MAY
$etag	opaque identifier assigned by the provider to a version of a resource	MAY
$properties[1]	Object containing metadata for the properties of the resource	MAY
$links[2]	Object containing links that present functional aspects of the resource (ex: edit, lookup, create). They are to be understood as hypermedia controls	MAY
$diagnosis	Object containing a more detailed indication of errors and warnings encountered by the provider during the execution of a request	MAY
native properties	Native properties of the object	MAY

An example is the JSON variant of the Typical SData Entry in the SData documentation:

```
…
$baseUrl : http://www.acme.com/MyApp/-/-/,
…
{
   "$url": "salesOrders('43660')",
   "$updated": "2008-03-31T13:46:45Z",
   "$key": "43660",
   "$title": "Sales Order 43660",
   "$etag": "gJaGtgHyuAwW6jMI4i0njA==",
   "orderDate": "2001-07-01",
   "shipDate": null,
   "contact": {
      "$url": "contacts('216')",
      "$key": "216"
    },
    "subTotal": 1553.10
}
```

JSON feeds

Feeds are collections of entries, returned by operations targeting several resources in parallel such as read or query operations on resource kinds. A feed is a JSON object with all entries contained in the `$resources` array – this is the only required property of a feed.

[1]

[2] will be described in detail in another paper on JSON metadata and its usage.

A feed may also contain a set of SData-defined properties, which are described in the table below:

	Description	Compliance
$resources	Array containing the individual entries	**MUST**
$baseUrl	URL leading to the resource kind level of an application. The URL MUST end in a "/" **Example**: *"$baseUrl": "http://www.acme.com/MyApp/MyContract/-/"*	MAY
$url	URL pointing to the resources returned. The URL should be represented as relative to the `$baseUrl` value. **Examples**: • *"$url": "customers"* • *"$url": "customers?where=name ge 'm'"* If a `$baseUrl` property is not specified, then the URL MUST be absolute. ***Examples***: • *"$url": "http://ex.com/MyApp/-/-/customers"* • *"$url": "http://ex.com/MyApp/-/-/customers?where=name ge 'm'"*	MAY
$title	humanly readable description of the resource	MAY
$updated	the time stamp of the last update of the resource, formatted according to ISO 8601 dateTime specification	MAY
$links[3]	Object containing links that present functional aspects of the feed (ex: refresh, first-page, schema, template, …). They are to be understood as hypermedia controls	MAY
$diagnosis	Object containing a more detailed indication of errors and warnings encountered by the provider during the execution of a request	MAY

An example is the JSON variant of the SData Typical Feed :

```
{
      "$baseUrl": "https://www.example.com/MyApp/-/-/"
      "$url": "salesOrders",
      "$title": "Sage App | Sales Orders",
      "$totalResults": 31465,
      "$startIndex": 1,
      "$itemsPerPage": 10,
      "$resources": [
         {
            "$updated": "2008-03-31T13:46:45Z",
            "$key": "43660",
            "$title": "Sales Order 43660",
            "$etag": "gJaGtgHyuAwW6jMI4i0njA==",
            "orderDate": "2001-07-01",
            "shipDate": null,
            "contact": {
```

[3] These will is described in detail in another paper on JSON metadata and its usage.

```
                "$url": "contacts('216')",
                "$key": "216"
             },
          "subTotal": 1553.10
       },
       {
          "$updated": "2008-03-31T13:46:45Z",
          "$key": "43661",
          "$title": "Sales Order 43660",
          "$etag": "3nqPeQqoGoxQB5xf3NIijw==",
          "orderDate": "2001-07-01",
          "shipDate": null,
          "contact": {
                "$url": "contacts('281')",
                "$key": "281"
          },
          "subTotal": 39422.12
       }
    ]
}
```

JSON diagnosis

The diagnoses is an object that contains information about the status (information, warning, error) of a request's execution. Such an object **MUST** be present in a response **if** errors were encountered during the execution. The xml format of diagnosis is described in the SData protocol in the 3.10: Error payload section.

The JSON format of the diagnosis objects supports the following properties:

	Description	Compliance
$severity	Severity of the diagnosis entry. Possible values are: • Info • Warning • Transient • Error • Fatal	**MUST**
$sdataCode	The SData diagnosis code	**MUST**
$applicationCode	Application specific diagnosis code	MAY
$message	Friendly message for the diagnosis	**SHOULD**
$stackTrace	Stack trace – to be used with care	MAY
$payloadPath	XPath expression that refers to the payload element responsible for the error	MAY

An example of a diagnoses JSON object is shown below:

```
{
  "$diagnoses": [
```

```
      {
        "$severity": "error",
        "$sdataCode": "BadWhereSyntax",
        "$message": "Invalid query syntax",
        "$applicationCode": "2403"
      }
   ]
}
```

JSON tracking

The tracking object **MUST** be returned by a provider in response of an asynchronous operation. SData describes a set of properties that can be provided in a tracking object. For JSON, these are:

	Description	Compliance
$phase	End user message describing the current phase of the operation.	MAY
$phaseDetail	Detailed message for the progress within the current phase.	MAY
$progress	Percentage of operation completed.	MAY
$elapsedSeconds	Time elapsed since operation started, in seconds.	**MUST**
$remainingSeconds	Expected remaining time, in seconds.	MAY
$pollingMillis	Delay (in milliseconds) that the consumer should use before polling the service again.	**MUST**

An example of a tracking JSON object is shown below:

```
{
  "$tracking": {
        "$phase": "Archiving FY 2007",
        "$phaseDetail": "Compressing file archive.dat",
        "$progress": 12.0,
        "$elapsedSeconds": 95,
        "$remainingSeconds": 568,
        "$pollingMillis": 500
  }
}
```

A note to SData ATOM+xml users

If you are intimately aware of the ATOM+xml format of SData as described in the SData 1.x standard, you will have noticed that some ATOM mandated elements are not present in the JSON objects of this document. The reason is that these are meaningful in a syndication context but have little relevance in the general SData application. The elements omitted are:

- Envelope markup `<feed>` : no longer needed due to representation as a JSON object

- Envelope markup `<entry>` : no longer needed due to representation as a JSON object
- `<id>` : information is carried by the `$url` element
- `<link rel='self'>` : information is carried by the `$url` element
- `<author>`
- `<category>`