SData 2.0: Contracts

Version 1.0

Purpose

In previous SData versions, the role and contents of a contract was not clearly specified. Our initial belief was that the intuitive understanding of the term would lead to consistent result; unfortunately, this assumption was in some cases wrong, causing discussions and delays.

SData 2.0 allows more freedom of implementation than any the previous releases. Many previously required aspects of the standard become optional; in addition, there is a choice between the formats (atom+xml or json) used to represent the information exchanged between provider and consumer. In this new setting, it is more than ever necessary to clearly define the components of a business-oriented solution - a duty to be fulfilled by the contract.

In order to clear previous misunderstandings and guide future developments, this document provides a more precise description of the SData contract concept.

What is an SData contract

In its wider meaning, a contract is an (usually binding) understanding between consenting parties. The points agreed are documented in textual form and kept for reference. The important bit is the understanding; the documentation and formalism thereof will vary from contract to contract.

An SData **contract** is to be viewed as a mutual understanding described by documentation that specifies the functional, logical and structural semantics of an SData interface. The contract is a fundamental component, occupying (usually) one of the first segments of the SData URL.

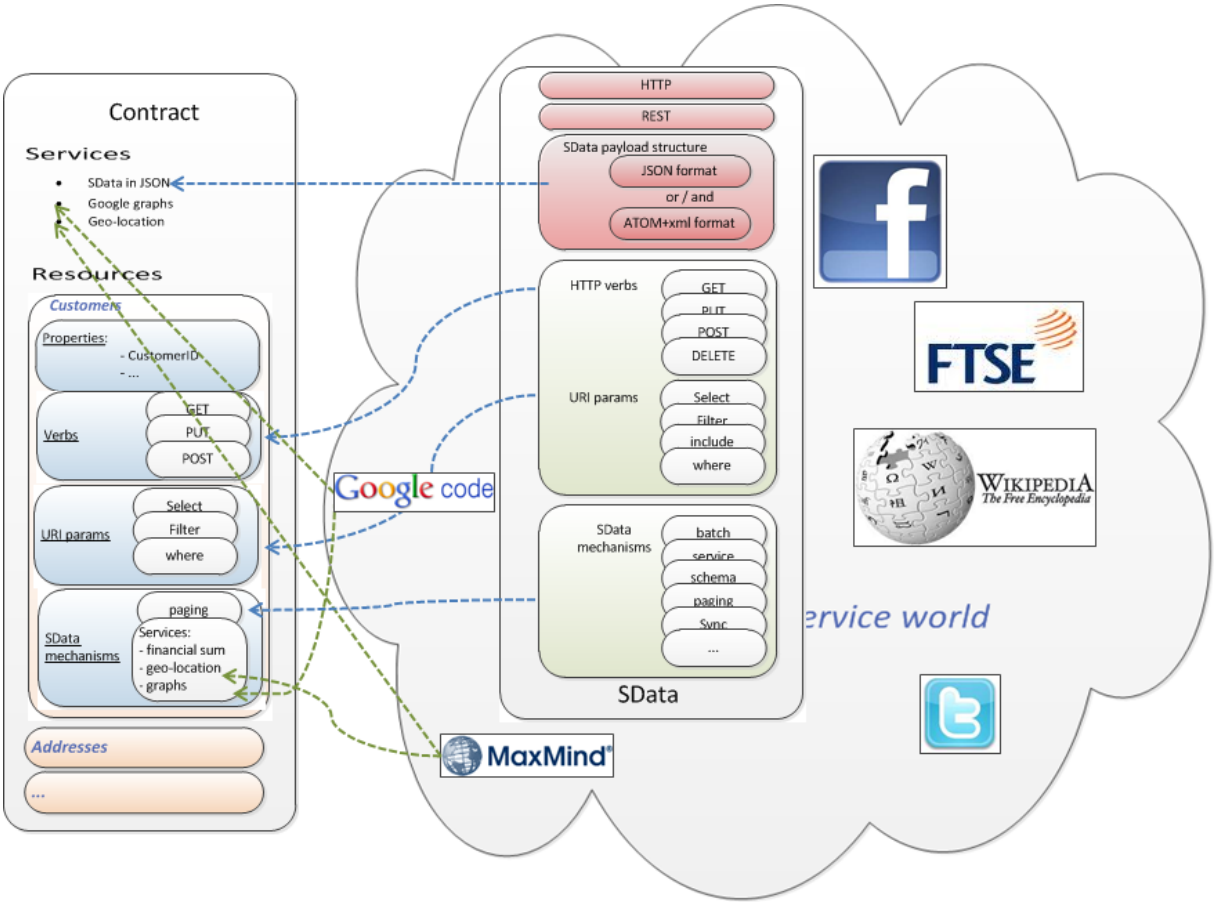
A contract:

- solves a business requirement and is purpose directed. Any application can provide multiple contracts in order to satisfy varying, independent and specific requirements
- uses those SData features deemed appropriate to express the functional aspects of a solution
- uses services from non SData providers as necessary for achieving its business goal
- is **implemented** by an application through the provision on the application's side of an SData interface that conforms to the requirements of the contract

As such, a contract:

- should specify adherence to the syntactic and semantic rules of a specific SData release (i.e. support of SData 1.0, 1.1 or 2.0)
- should define which optional aspects of SData are supported and in what context
- Should describe the interaction:
    - with a single application (typical in a client-server scenario)
    - between several applications (deeper application integration)
- may span a **set of documents of varying types**. Even if a schema (xsd) is one such document, it is not by necessity the embodiment of the contract; as a note: the presence of a schema is not mandatory in a contract and in SData 2.0, it will also no longer be a MUST.
- may impose additional constraints on the usage of SData, (eg: support of specific SData artifacts, constraining URL structure,...)
- cannot relax or ignore the SData protocol requirements (MUST items in SData)

The following diagram shows that a contract chooses from a variety of services and SData artifacts to achieve its goals:



The diagram above presents the major components of a contract:

- the services and formalisms used (in this case SData, Google and MaxMind)
- the resources and their operations. It is important to note that the `customers` resource supports only a subset of the SData verbs (DELETE is missing), a subset of query parameters and only one SData mechanism, the paging. External services are used for graphing and geo-location.

A contract will usually fall in one of the following two categories:

- domain contract, defining the interactions with applications of a business domain. This is the case with the GCRM contract
- native contract, defining an application-specific interface. This is the case with products like Sage50, OfficeLine, X3 and many others

In most cases, a contract documents an API. Therefore, also in most cases, a good API documentation would be a well-defined contract.

Examples

One of the best known examples of a contract is the global CRM (GCRM) integration contract. It provides a common definition of services to be leveraged by CRM applications when integrating with ERP applications. The contract describes:

- a (very large) subset of SData 1.1 to be supported by providers
- the underlying structures and services by means of a schema
- the synchronization mechanism support
- additional documentation for inter-application workflows

Another example is the mobile-device contract of the German OfficeLine. This exposes a functionally slim and narrow API designed specifically to serve mobile OfficeLine client apps. What makes this contract interesting is the fact that it is very purpose directed (mobile clients and not a general-purpose API), that it resides alongside other contracts and is application specific.

What makes a 'good' contract

SData is in effect composed of a small core of requirements and a wide number of artifacts and mechanisms that may be adopted in order to provide viable, re-usable solutions with a lot of commonality.

The contract turns the SData 'may' into the set of 'must' that delivers a solution. A contract should therefore be **clear, precise** and **detailed.** Having said that, we will temper it by adding: **…** but in the amount that the (potential) adopters of the contract require.

Given the high variance (ex: from business partner to internal development teams) in the skills and needs of the audience, there is no absolute manner to assess the quality of a contract: 'good' is relative.

Along the same lines, the documents that describe the contract are difficult to pinpoint. Some team work better using technical formalisms such as schemas or UML diagrams, while others prefer prose descriptions of the functionality. Again, orienting oneself on the target audience will help choose the right artifacts to define a contract.

A couple of points you may want to consider

One thing is constant: among those who will have to deliver to (or work with) an SData contract are technical people. Therefore, a large part of the contract should address this audience and provide the typical level of detail and accuracy required by development teams.

The subsections below provide a list of things you may want to include/describe in your contract. Please view it as a guide and not as a directive.

Goal

This could sound silly, but really stating what business benefit is achieved by complying with a contract can help a lot of people in their understanding (ex: is this relevant for me, what do I get if I comply,…).

Try to describe the functional aspects of the contract - it will certainly be appreciated by the readers and implementers alike.

Provider level

At this level you should consider specifying those aspects that pertain to the provider as a whole. Such aspects are:

- Application **base URL** (i.e. the URL to reach the application): must it have a particular format (ex: for discoverability reasons)? This makes usually good sense to spell it out if the contract is native.
- Is there an implementation of the **SData registry** present to support discoverability?
- Intra-application URL segments:
  - **Contract segment**: is it the default contract (-) or a named contract.
  - Contract **version segment**: it is good to think about the versioning of contracts, even if changes happen in large time intervals.
  - **Datasets/Tenants segment(s)**: are several tenants or datasets supported for a contract by an application? If so, do they need to be structured in specific segments of the URL?
- Capabilities
  - **$system**: is this provided? Are there other discovery mechanisms?
  - **$schema**: is the application wide schema provided? If so and versioning is not done at the URL level, are there any specific ways to retrieve previous versions of a schema?
  - **$batch**: are there any resource-independent batch operations provided?
  - **$services**: are there any resource-independent predefined operations present?
  - **$queries**: are there any resource-independent predefined queries present?
- Provider-wide supported **query parameters** (see section 2.11 of the SData 1.x standard)
- Provider-wide **metadata:** is there a subset of metadata and hypermedia controls present at every level (application and resources)
- **Security aspects**:
  - Encryption
  - Authentication

Resources

It seems a good idea to describe every individual resource covered by the contract. The following points should be considered on a per-resource-kind basis.

URLs

URLs are the addressing portion of a resource or a collection of resources. Stating explicitly what the URLs are will usually help the implementers significantly.

- **Resource collection URL**: it is good to specify this
- **Single resource URL**: it is a good practice to describe this. Ideally, describe this as a pattern (ex: …/`customers('{customerID}')`)

Payload formatting

- Are **JSON** and/or **atom+xml** supported?
- If both are supported:
  - Which one is the **default** format?
  - How can a consumer request the **alternate format**? Through the Accept header, and/or the format query parameter?

Resource structure

The payload structure is a vital portion in the description of a resource; it goes beyond simply listing the properties and their meaning, it describes how properties belong together and the manner in which they should be treated.

It is meaningful to consider, for each property, the inclusion of the following aspects in the contract documentation:

- **Name**
- **Description**
- Kind
  - ***Scalar***: basic type (number, string, date, …), mime type, reference
  - ***Group***: contains a structured set of properties (is per definition immutable)
  - ***Collection*** (to be thought of as an array of several items with identical structure)
    - *Mutability*
      - Immutable (cannot add/delete items from the collection)
      - Mutable (can add/delete items). In this case, the collection can be altered by:
        - passing complete collection
        - per partial update using $isDeleted metadata attribute
    - *Collection Item*
      - Nature
        - Embedded (can change the properties of items in place)
        - Reference
          - Lookup metadata control provided
      - Structure (repeat, recursively as for the payload)

Operations

The operations are essential. They usually go beyond the simple HTTP verbs as the list below shows. Consider documenting each of the below in your contract.

- **Verbs** supported for this resource
  - **Read** operation
  - **Update** operation
    - Support of partial payloads (PUT, PATCH)
    - Concurrency handling?
  - **Create** operation
    - Reliable POST supported?
    - Concurrency handling?
  - **Delete** operation
    - Concurrency handling?
  - **Query** operation
    - Query language support: basic, intermediate, complete) (see section 2.12 of standard)
    - Query parameter support: Include, Select, where, … (see section 2.11 of standard)
- **Hypermedia controls** (links): these augment the logical operational range on a resource by indicating the URL+operation+parameter combination required to perform a certain action; think of the `next` link in a paged set – it is a good example

- **Services** ($services) and **Queries** ($query). For each service consider indicating the following
  - Name
  - Description
  - URL and query parameters
  - Access method (ex: PUT/POST/GET)
  - Request structure, if a payload is relevant
  - Response structure
- **Synchronization** support
  - Synchronization source
  - Synchronization target

Mechanisms supported

- **Batching**
- Sync/async **invocation**
- **Paging**
- **Contract specific mechanisms :** if-and-only-if SData defined mechanisms could not be used

Capabilities

- **$schema** (xsd format)
- **$template**
- **$prototype** (JSON)
- **Resource versioning** support
  - Etag
  - Modified date
  - Version support
- **Metadata** specific to a resource

Contract/provider specific errors

It is a good idea to specify which error codes could be expected for the contract. Generally, these will be provider specific codes for native contracts, but in the context of domain contracts, finer graduations than the HTTP error codes are not uncommon.

The error description should contain:

- Error code
- Context : resource + operation
- Description
- Remedies – if any