Proposed edits to http://code.google.com/apis/opensocial/docs/0.8/restfulspec.html:

1. Overview

Existing Text	New Text
No single data representation is ideal for every client. This protocol defines dual representations for each resource in two widely supported formats, JSON [RFC4627] and Atom/AtomPub [RFC4287][RFC5023], using a set of generic mapping rules. The mapping rules allow a server to write to a single interface rather than implementing the protocol twice.	No single data representation is ideal for every client. This protocol defines representations for each resource in three widely supported formats, JSON [RFC4627], Atom/AtomPub [RFC4287][RFC5023], and XML using a set of generic mapping rules. The mapping rules allow a server to write to a single interface rather than implementing the protocol multiple times.
OpenSocial container servers are free to define additional representations but MUST support at least the JSON and Atom formats defined in this document.	OpenSocial container servers are free to define additional representations but MUST support at least the JSON, Atom, and XML formats defined in this document.

2. Data Representations

Existing Text	New Text
Each resource has a two representations, as JSON and Atom (XML). All data must be representable in both formats, but we do not attempt to map from generic XML or Atom to JSON. Instead, we define an internal data model using English and JSON syntax, and then define the mappings between this and Atom/JSON.	Each resource has three representations, as JSON, Atom (XML), and generic XML. All data must be representable in both formats, but we do not attempt to map from generic XML or Atom to JSON. Instead, we define an internal data model using English and JSON syntax, and then define the mappings between this and Atom/JSON as well as JSON and generic XML.
Mapping consists of converting between the internal hierarchy and the JSON / Atom protocol format.	Mapping consists of converting between the internal hierarchy and the JSON / Atom protocol /generic XML format.
N/A	(insert prior to “Examples of the primary types of data follow. Each example shows both representations, JSON and Atom, with the payload data highlighted for ease of comparison.”)

	The general rules for mapping between the generic XML and JSON formats are as follows. • The default location for all data in the generic XML format is in datatype, where datatype is a root node naming the type of data delivered: <person>, <group>, <activity>, or <appdata>. • The field names are the same as in the JS documentation, in camelCase (the same format as the JS field accessors; e.g, "lastName". • Strings are represented as strings in both formats. • Dates and timestamps are represented as strings containing XML Schema Part 2, section 3.2.7 values (http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/#dateTime). These are also known as "XSD Dates". In cases where only a day-of-the-year is desired, e.g., a birthday, the year SHOULD be specified as 0000. • Enums are represented as objects with "displayvalue" (localizable, customizable string) and "key" (key) fields. • Arrays are represented as arrays in the JSON representation and as repeated fields in the XML representation. • Sub-objects are represented as sub-elements in both formats. • Fields are placed directly in the root object in the JSON format. In the generic XML format, they are by default placed under datatype (e.g., person for person data). Fields are NEVER encoded as attributes on elements. Instead, fields are always included as elements plus text data.

Examples of the primary types of data follow. Each example shows both representations, JSON and Atom, with the payload data highlighted for ease of comparison	Examples of the primary types of data follow. Each example shows representations in JSON, Atom, and generic XML with the payload data highlighted for ease of comparison

2.1 Collections

Existing Text	New Text
Collections are a useful abstraction for dealing generically with multiple things, whether those things are persons, groups, activities, or application data sets. They have both Atom and JSON representations; the Atom representation is simply a standard Atom feed whose entries are one of the entry types specified above. The default JSON collection representation is a JSON object containing an "entry" slot containing a list of JSON objects. Collections use the OpenSearch conventions for reporting totalResults (for complete unpaged feed), startIndex of current page, and itemsPerPage.	Collections are a useful abstraction for dealing generically with multiple things, whether those things are persons, groups, activities, or application data sets. They have the Atom, JSON and generic XML representations; the Atom representation is simply a standard Atom feed whose entries are one of the entry types specified above. The default JSON collection representation is a JSON object containing an "entry" slot containing a list of JSON objects. The default generic XML collection representation is a <collection> node containing an “entry” slot containing a list of generic XML elements. Collections use the OpenSearch conventions for reporting totalResults (for complete unpaged feed), startIndex of current page, and itemsPerPage.
	(insert as generic XML example) application/xml representation: <collection xmlns="http://ns.opensocial.org/2008/opensocial"> <author> example.org:58UIDCSIOP233FDKK3HD44 </author> <link> <rel>next</rel> <href>http://api.example.org/...</href> </link> <totalResults>100</totalResults> <startIndex>1</startIndex> <itemsPerPage>10</itemsPerPage> <entry> <entry>{...first thingie...}</entry> <entry>{...second thingie...}</entry> ... </entry> </collection>

2.2 Person

Existing Text	New Text
	(insert as generic XML example) application/xml representation: <person xmlns="http://ns.opensocial.org/2008/opensocial">> <id>example.org:34KJDCSKJN2HHF0DW20394</id> <name> <unstructured>Jane Doe</unstructured> </name> <gender> <displayvalue>女性</displayvalue> <key>FEMALE</key> </gender> </person>

2.3 Group

Existing Text	New Text
	(insert as generic XML example) application/xml representation: <group xmlns="http://ns.opensocial.org/2008/opensocial"> <id>example.org:34KJDCSKJN2HHF0DW20394/friends</id> <title>Peeps</title> <link> <rel>alternate</rel> <href>http://api.example.org/people/example.org:34KJDCSKJN2HHF0DW20394/@friends</href> </link> </group>

2.4 Activity

Existing Text	New Text
	(insert as generic XML example) application/xml representation: <activity xmlns="http://ns.opensocial.org/2008/opensocial"> <id>http://example.org/activities/example.org:87ead8dead6beef/self/af3778</id> <title> <type>html</type> <value> some activity </value>

	</title> <updated>2008-02-20T23:35:37.266Z</updated> <body>Some details for some activity</body> <bodyId>383777272</bodyId> <url>http://api.example.org/activity/feeds/.../af3778</url> <userId>example.org:34KJDCSKJN2HHF0DW20394</userId> </activity>

2.5 AppData

Existing Text	New Text
	(insert as generic XML example for isolated AppData) application/xml representation: <appdata xmlns="http://ns.opensocial.org/2008/opensocial"> <pokes>3</pokes> <last_poke>2008-02-13T18:30:02Z</last_poke> </appdata>
	(insert as generic XML example for AppData Collection) application/xml representation: <appdata xmlns="http://ns.opensocial.org/2008/opensocial"> <entry> <entry> <id>example.org:34KJDCSKJN2HHF0DW20394</id> <pokes>3</pokes> <last_poke>2008-02-13T18:30:02Z</last_poke> </entry> <entry> <id>example.org:58UIDCSIOP233FDKK3HD44</id> <pokes>2</pokes> <last_poke>2007-12-16T18:30:02Z</last_poke> </entry> </entry> </appdata>