RCP - WEBAPI-025 Lightweight Autofill Schema

Submitter Name	Alexander Likholyot
Submitter Organization	Planitar
Submitter Email	alex@planitar.com
Co-submitter Name	Joshua Darnell (**Currently Editing**)
Co-submitter Organization	RESO
Co-submitter Email	josh@reso.org
Co-submitter Name	Paul Stusiak
Co-submitter Organization	Falcon Technologies Corp.
Co-submitter Email	pstusiak@falcontechnologies.com

Document Name	Data Dictionary Common Schema (JSON)
Document Version	1.7
Date Submitted	2019-02-22
Status	IN REVIEW
Status Change Date	
Related Documents	Lightweight Autofill Object (RCP-022) RESO Common Schema JSON Format Data Dictionary 1.7 Testing Specification

Synopsis

This RCP outlines the schema to accompany the Lightweight Autofill proposal (RCP-022), which describes the business portion of the problem being solved. The schema presented in this document specifies the payload that would be returned in the process of resolving an autofill URI, as outlined in the workflow of RCP-022.

Rationale

When a RESO Web API server uses the Lightweight Autofill workflow to populate a resource with remote data, it will be provided a with a URI that returns an object which contains data to be attached to a given resource record. As such, there is a need to have a standard shape that all RESO Web API servers should be able to understand.

In general, the goal is to create schema that follow the structure contained in the Data Dictionary as closely as possible. This proposal doesn't introduce any new Data Dictionary items. Rather, it presents a normative reference for the data shape expected by the RCP-022 proposal, in which case the resource being populated is a Property record with attached Media and Rooms resource data.

One goal of this proposal is to make the shape of the autofill object resemble that which would be expected during upsert of similar Data Dictionary resource data in order to create predictability when interacting with the RESO Web API.

Proposal

In Step 5 of the Lightweight Autofill proposal, a RESO Web API server is expected to retrieve data from an external source and attach them to a Property record, either existing or in-progress, in a multiple listing service. Alternatively, these data could be provided directly to a Web API that is consuming them, for instance by being pasted into a form within a listing input module or provided through Web API Update, as part of RCP-010. However, in the case of Web API Update, in many cases the people updating these records are professionals working outside the MLS, such as Photographers, who are able to easily create a payload of a certain structure, but not as easily build a Web API Update client. Additionally, at the time of writing, most servers don't support Update, and this proposal adds a way for those systems to take in data to their systems without exposing an update API. As such, the schema of the update object being proposed below mirrors that which would be expected by a Web API Update server.

The data being produced in the initial RCP-022 proposal are those corresponding to Property, Media, and Rooms. In this case, the top-level resource is Property, and the "attached" data are for Media and Rooms. Currently in the RESO Data Dictionary, the relationship between Property and Media as well as Rooms is a "belongs to" relationship, meaning that Media and Rooms *belong to* a given Property. Compare this to a "has a" relationship, such as *Property has Media*, or *Property has Rooms,* as is suggested by the shape of the proposed schema

below. However, it's worth noting that not all target Web API servers processing the data for the Autofill workflow will support the "has a" version of this relationship, and may need to do additional processing behind the scenes to extract the Media and Rooms from the payload and upsert them appropriately. This proposal does not specify the implementation details of that process, which will vary from server to server. It specifies the shape that a server processing Autofill data can expect.

There is a slight nuance to working with the Data Dictionary in that it supports both string and numeric keys. In the former case, the field takes the suffix "Key" and in the latter, "KeyNumeric." In the cases where the autofill payload is supposed to reference an existing record, the autofill provider will be expected to fill in the appropriate keys. In cases where the Media or Rooms records don't already exist, they will only contain data fields and not key fields.

Case 1: Autofill object does not include any keys for existing records

In this case, no keys have been provided as records are expected to be created for each all resource data contained in the autofill payload.

The autofill API consumer would provide a URI to the Web API host, which would resolve that URI and a payload similar to the following:

```
{
  "value": [{
    "Country": "CA",
    "StateOrProvince": "YT",
    "City": "Dawson City",
    "PostalCode": "Y0B 1G0",
    "StreetName": "Bayshore Rd",
    "StreetNumber": "1803",
    "Media": [{
      "ResourceName": "Property",
      "MediaCategory": "Branded Virtual Tour",
      "MediaType": "mov",
      "MediaURL": "https://example.com/vJVDL415WZ7GE1/",
      "ShortDescription": "Example"
    }, {
      "ResourceName": "Property",
      "MediaCategory": "Floor Plan",
      "MediaType": "pdf",
      "MediaURL": "https://example.com/vJVDL415WZ7GE1/doc/floorplan_imperial.pdf",
      "ShortDescription": "imperial"
    }],
    "PropertyRooms": [{
        "RoomType": "Dining",
        "RoomName": "Breakfast",
        "RoomWidth": 4.409,
        "RoomLength": 2.977,
        "RoomLengthWidthUnits": "Meters",
        "RoomLengthWidthSource": "LocalProvider"
      }, {
        "RoomType": "Dining",
        "RoomName": "Dining",
        "RoomWidth": 4.3,
        "RoomLength": 5.998,
        "RoomLengthWidthUnits": "Meters",
        "RoomLengthWidthSource": "LocalProvider"
      },

    ]
  }]
}
```

Note the nested `value` array in the payload. This is to ensure that the proposed schema conforms with the payload conventions specified by OData and the Web API. This allows servers to use the same payload for autofill as they do with upsert.

Case 2: Autofill object includes string keys for existing records

Assume listing data already exists in the Web API host's data store and the goal is to make a call to an autofill provider to update existing data, such as the StreetNumber of the Property record, as well as the dimensions of one of the existing rooms. In this case, the user would provide an autofill URI that provides response similar to that used in Case 1, except now the resulting payload would contain keys for any data that was meant to be updated, as follows:

```
{
  "value": [{
    "StreetNumber": "1804",
    "PropertyRooms": [{
      "RoomKey": "A34535F235",
      "RoomWidth": 5.139,
      "RoomLength": 3.882
     }]
    }
  ]
}
```

Note: in the case of upserts, the autofill provider MUST include the keys for the records to be updated. In the example above, `RoomKeyNumeric` might be used instead, depending on how the Web API autofill consumer represents their keys.

Autofill providers are expected to return a standard HTTP 200 status code when autofill objects are successfully transferred from the provider to the autofill consumer, i.e. the Web API host. Autofill consumers should be prepared to render the appropriate error message in cases where the autofill object fails to transfer. This proposal doesn't specify any additional status codes that might be used in such a scenario and instead relies on existing HTTP protocol conventions for status codes. For example, a 500 should be thrown when there's an internal server error of some kind on the autofill provider.

Impact

Those wanting to implement the Autofill proposal described in RCP-022 and RCP-025 will need to create a mechanism for retrieving autofill objects and attaching them to an existing resource in their system (or use them to create new resources). This proposal doesn't add additional requirements to the Web API specification, as its purpose is to create a data contract that can be used to process external resource data into a Web API host system.

Compatibility

RESO Data Dictionary 1.7+

Certification Impact

RESO Certification is not required at this time. This may be revised given sufficient interest in having these payloads certified.