Alice Protocol

May 31, 2012

Contents

1 Introduction

Alice acts as the connector between one or more Bob clients, a PostgreSQL data store and one ore more PBX'es. Her main task is to get and set data via a HTTP/JSON based protocol.

2 The Protocol

When data is requested (get) the search parameters are always given as a HTTP GET query string and the result is always returned as a JSON document. When data is inserted/updated/deleted (set) the parameters are always given as a HTTP POST request and the confirmation returned from Alice will always be a JSON document.

In order to access any of the interfaces, a client must first authenticate itself. This is done using OpenID. The login interface is the only interface that can be accessed without prior login.

All data handled by the protocol is expected to be UTF-8 encoded. Alice does not check whether this is in fact the case, so when implementing clients, be sure to verify that all data sent to Alice are UTF-8 encoded.

Application level errors and exceptions are returned as JSON documents. How these are handled is entirely up to the client. HTTP level errors are returned as normal for HTTP servers.

3 The Same Origin Policy - CORS and JSONP

Same Origin Policy is a security feature found in the Javascript implementation in most browser. It prevents you from making requests to pages on a different domain, another subdomain or through a different protocol. The consequence of this is that you cannot query Alice from a client on a different domain than the one used by Alice, which in some cases can be a pain.

The solutions to this problem is something called Cross-Origin Resource Sharing (CORS) and the JSONP standard. Both of these are supported out of the box by jQuery and probably also by other Javascript frameworks. The "correct" and preferred method to use for cross domain access is CORS. Only use JSONP if CORS for some reason or another isn't supported by your platform.

4 The Data Model

Instead of using only a relational or document based model, we've opted for a mix. Data that is relational in nature is stored in columns in a relational database, whereas everything else is stored in a JSON document. When a client requests a specific type of data Alice combines these two storage models into a final JSON document.

A benefit of this is that the client software easily can add new fields and data structures, simply by expanding on the JSON document. If we take the */get/contact* interface as an example, it might return a JSON document looking like this:

```
{
"name":"Arthur Dent",
''items'':[''towel'',''heart of gold'',''dressing gown''],
''emailaddress'':''arthurdent42@somewhere.unknown.org'',
''cellphone'':''555-777-888-999'',
```

```
‘‘dislikes’’:’’Vogons’’,
‘‘db_columns’’:
    {
    "ce_id":1,
    "ce_name":"Arthur Dent",
    ‘‘sip_uri’’:’’sip://arthurdent.somewhere’’
    }
}
```

The *db_columns* JSON node is build from the actual columns available in the relational database, in this case *ce_id*, *ce_name* and *sip_uri*. The data found in these columns can be queried, sorted, grouped and/or joined by Alice. She is, so to speak, aware of them.

The other JSON nodes (*name*, *items*, *emailaddress*, *cellphone*, *dislikes*) comes from the JSON document associated with a contact entity. This is controlled 100% by the client software. Alice is completely oblivious as to what this document contains and why. Things put here are only searchable by the client.

The strength of this model is that new fields can be added to a contact entity on the fly, without needing any changes to Alice or the database. Changes to Alice and/or the database itself is only necessary if the new field must be searchable by Alice, for example in the case of a new interface, where some given parameter is being used to search for specific rows of data.

The weakness of this model is of course that a poorly designed/build client can end up with some very bad JSON. We believe the strength outweighs the weakness.

5 Exceptions

When bad things happen an exception is raised and an error JSON document returned. These have the following format:

```
{
"exception_message":"foo",
"exception":"SOME.EXCEPTION",
"message":"bar"
}
```

All exceptions have this format. It is entirely up to the client to decide what to do with an exception.

6 Getting Data

6.1 /get/call[?id=call id]

This interface returns the JSON document associated with a call. If the *id* parameter is given, and the *id* actually exists in the call queue, then the specific call is returned. If the *id* parameter is not given, the call with the highest priority and age is returned. If a call is returned it is also removed from the queue. Example JSON document:

```
{
"id":"08yvc18eIM",
"org_id":3
}
```

If no call is found, an empty JSON document is returned:

```
{}
```

6.2 /get/contact?ce_id=positive integer

This interface returns the JSON document associated with the contact entity identified by *ce_id*. Example JSON document:

```
{
"name":"Arthur Dent",
‘‘db_columns’’:
    {
    "ce_id":1,
    "ce_name":"Arthur Dent",
    ‘‘sip_uri’’:’’sip://arthurdent.somewhere’’
    }
}
```

An empty JSON document is returned if *ce_id* doesn't exist in the database:

```
{}
```

6.3 /get/contact_attributes?ce_id=positive integer

This interface returns the attributes associated with the *ce_id* contact entity. Example JSON document:

```
{
"attributes":[
    {
    "cellphone":"555-777-888-999",
    "db_columns":
        {
        "ce_id":1,
        "org_id":1
        },
    "email":"arthurdent42@somewhere.unknown.org",
    "tags":[
        "Traveller",
        "Towel",
        "Heart"
        ]
```

```
    },
    {
    "phone":"999-000-111",
    "db_columns":
        {
        "ce_id":1,
        "org_id":2
        },
    "email":"some.other@email.address"
    }
    ]

}
```

Note that *attributes* contains an array, meaning that one contact entity can be associated with several attribute sets. This means that the attributes of a contact entity can change depending on the context in which it is seen.

If there are no attributes associated with a contact entity, the following JSON document is returned:

```
{
"attributes":[]
}
```

6.4 /get/contact_full?ce_id=positive integer

This interface returns the full data associated with *ce_id*, meaning it combines the data from */get/contact* and */get/contact_attributes*. Example JSON document:

```
{
"attributes":[
    {
    "cellphone":"555-777-888-999",
    "db_columns":
        {
        "ce_id":1,
        "org_id":1
        },
    "email":"arthurdent42@somewhere.unknown.org",
    "tags":[
        "Traveller",
        "Towel",
        "Heart"
        ]
    },
    {
    "cellphone":"333-666-111-222",
    "db_columns":
```

```
            {
            "ce_id":1,
            "org_id":2
            },
        "email":"some.other@email.address"
        }
        ],
        "name":"Arthur Dent",
        "db_columns":
            {
            "is_human":true,
            "ce_id":1,
            "ce_name":"Arthur Dent"
            },
        "type":"human"
    }
```

An empty JSON document is returned if *ce_id* does not exist:

```
{}
```

6.5 /get/organization?org_id=positive integer

This interface returns the data associated with the *org_id* organization. Example JSON:

```
{
"name":"AdaHeads K/S",
"db_columns":
    {
    "org_name":"AdaHeads K/S",
    "identifier":"sip://adaheads",
    "org_id":1
    }
}
```

An empty JSON document is returned if no *org_id* organization is found in the database:

```
{}
```

6.6 /get/org_contacts?org_id=positive integer

This interface returns all the contact entities associated with *org_id*. Example JSON:

```
{
"contacts":[
```

```
    {
    "name":"Zaphod B.",
    "db_columns":
        {
        "ce_id":4,
        "org_id":1,
        "ce_name":"Zaphod B."
        }
    },
    {
    "name":"Arthur Dent",
    "db_columns":
        {
        "ce_id":1,
        "org_id":1,
        "ce_name":"Arthur Dent"
        }
    }
    ]
}
```

The *contacts* node contains an array, so it can hold multiple contact entities. If no contact entities are associated with the given *org_id* the following JSON document is returned:

```
{
"contacts":[]
}
```

6.7 /get/org_contacts_attributes?org_id=positive integer

This interface returns all the contact entity attribute sets associated with *org_id*. Example JSON document:

```
{
"attributes":[
    {
    "cellphone":"555-777-888-999",
    "db_columns":
        {
        "ce_id":1,
        "org_id":1
        },
    "email":"arthurdent42@somewhere.unknown.org",
    "tags":[
```

```
            "Traveller",
            "Towel",
            "Heart"
            ]
        },
        {
        "cellphone":"444-555-777",
        "db_columns":
            {
            "ce_id":4,
            "org_id":1
            },
        "email":"zb@somewhere.unknown.org"
        }
        ]
}
```

Because there can be several contact entity attribute sets associated an *org_id* the *attributes* JSON node contains an array. If there are no attribute sets, the following JSON document is returned:

```
{
"attributes":[]
}
```

6.8 /get/org_contacts_full?org_id=positive integer

This interface returns the full contact data for all contacts associated with *org_id*, meaning it combines the data from */get/org_contact* and */get/org_contact_attributes*. Example JSON document:

```
{
"contacts":[
    {
    "attributes":
        {
        "cellphone":"555-777-888-999",
        "db_columns":
            {
            "ce_id":1,
            "org_id":1
            },
     "email":"arthurdent42@somewhere.unknown.org",
        "tags":[
            "Traveller",
            "Towel",
            "Heart”
            ]
```

```
        },
    "name":"Arthur Dent",
        "db_columns":
            {
            "is_human":true,
            "ce_id":1,
            "ce_name":"Arthur Dent"
            },
    "type":"human"
        },
        {
        "attributes":
            {
            "cellphone":"444-555-777",
            "db_columns":
                {
                "ce_id":4,
                "org_id":1
                },
         "email":"zb@somewhere.unknown.org"
            },
    "name":"Zaphod B.",
        "db_columns":
            {
            "is_human":true,
            "ce_id":4,
            "ce_name":"Zaphod B."
            },
    "type":"human"
        }
        ]
}
```


If there are no contacts associated with *org_id* the following JSON document is returned:


```
{
"contacts":[]
}
```


6.9 /get/queue

This interface returns the current call queue. This is updated once each second, so polling the /get/queue interface more often than once each second is both wasteful and pointless. Calling for example:

```
/get/queue
```

will return a JSON string like this:

```
{
"normal":[
    {
    "UTC_start_date":"2012-02-22 14:23:30",
    "id":"GDhcf2VBww",
    "unix_timestamp":"1329920610",
    "callee":5,
    "caller":"d7sIp1kR"
    }
    ],
"high":[
    {
    "UTC_start_date":"2012-02-22 14:23:11",
    "id":"bRbYsMUVqx",
    "unix_timestamp":"1329920591",
    "callee":3,
    "caller":"oCgDF7ua"
    }
    ],
"low":[
    {
    "UTC_start_date":"2012-02-22 14:23:17",
    "id":"VV8BFqGpqG",
    "unix_timestamp":"1329920597",
    "callee":9,
    "caller":"ZQsRogwB"
    }
    ],
‘‘length’’:3
}
```

The *normal*, *low* and *high* nodes represent priority in the queue. All these naturally contains arrays of calls. The *length* node contains the totalt length of the queue. The *callee* node maps to an *org_id*. An empty queue returns the following JSON document:

```
{
"normal":[],
"length":0,
"high":[],
"low":[]
}
```

6.10 /get/queue_length

This interface returns the length of the current call queue. Example JSON document:

```
{
"length":7
}
```