Alice Protocol

May 15, 2012

Contents

1 Introduction

Alice acts as the connector between one or more NAME_OF_PRODUCT 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. This document is the final authority on the Alice 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 - 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 solution to this problem is something called the JSONP standard. This is supported transparently and out of the box by jQuery (and probably other Javascript frameworks). You simply change the *get()* or *post()* call to be JSONP based by stating that you would like the return type to be *json*.

If you prefer to build your own query strings by hand, then you must add a *jsoncallback* (preferred) or *callback* (will work) parameter with any value. The value of *jsoncallback* is then used by Alice to wrap the returned JSON string, like this:

```
/get/queue_length?jsoncallback=foo
foo({"length":7})
```

Strictly speaking this is only necessary if the client is running in a browser and is being hosted from another domain than Alice.

4 The Data Model

Instead of using only a relational or document based model, we've opted for a mixture. 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, simply by expanding on the JSON document. If we take the */get/contact* interface as an example, it might return a JSON 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.

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/contact?ce_id=[natural 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:

```
{}
```

6.2 /get/contact_attributes?ce_id=[natural integer]

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

```
{
    "attributes":
    [
        {
            "phone":"555-777-888",
            "db_columns":
              {
                "ce_id":1,
                "org_id":1
              },
            "email":"some@email.address"
        },
        {
            "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":[]
}
```

6.3 /get/contact_tags?ce_id=[natural integer]

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

```
{
    "tags":
    [
        {
            "db_columns":
            {
              "ce_id":1,
              "org_id":2
            },
            "tags":
            [
              "Ada",
              "Slackware",
              "Linux"
            ]
        }
    ]
}
```

As with the attributes interface the *tags* node contains an array so several tag sets can be associated with a contact entity.

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

```
{
    "tags":[]
}
```

6.4 /get/organization?org_id=[natural_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:

```
{}
```

6.5 /get/org_contacts?org_id=[natural 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":2,
              "ce_name":"Zaphod B."
            }
        },
        {
            "name":"Tricia Takanawa",
            "db_columns":
            {
              "ce_id":1,
              "org_id":2,
              "ce_name":"Tricia Takanawa"
            }
        }
    ]
```

```
}
```

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":[]
}
```

6.6 /get/org_contacts_attributes?org_id=[natural integer]

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

```
{
    "attributes":
    [
        {
            "phone":"555-777-888",
            "db_columns":
              {
                "ce_id":1,
                "org_id":1
              },
            "email":"some@email.address"
        },
        {
            "phone":"444-555-777",
            "db_columns":
              {
                "ce_id":7,
                "org_id":1
              },
            "email":"contact@me.here"
        }
    ]
}
```

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":[]
}
```

6.7 /get/org_contacts_tags?org_id=[natural integer]

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

```
{
    "tags":
    [
        {
            "tags":
            [
              "Ada",
              "Slackware",
              "Linux"
            ],
            "db_columns":
              {
                "ce_id":1,
                "org_id":1
              }
        },
        {
            "tags":
            [
              "Support",
              "Accounting",
              "Sales"
            ],
            "db_columns":
              {
                "ce_id":7,
                "org_id":1
              }
        }
    ]
}
```

The *tags* JSON node contains an array because there can be several contact entity tag sets associated with an *org_id*. If there are no tag sets, the following JSON document is returned:

```
{
    "tags":[]
}
```

6.8 /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":[]
}
```

6.9 /get/queue_length

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

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