Proposal for a Common Groupware Interface Standard

1. Scope

As many different opensource and freesoftware groupware systems are being developed, the full realization of the dream of a connected world should be prefaced by an agreement to interoperate. There are limited ways in which cooperation with these and commercial groupware systems may be achecived, the majority if not all of which were derived via the establishment of open standards. These might include email (POP3/IMAP), contacts(LDAP,vcard), or scheduling(ical/vcal). It is felt that while these have proven themselves to be very useful, they are insufficient to satisfy the real needs of a typical business environment.

This document hopes to provide a reasonable, if limited, recommendation for a set of standardized methods to be used for groupware services interaction. More specifically, it hopes to address the need for such a standard as well as to spur discussion about the common service names and methods themselves.

Examples will be given for implementations in XML-RPC, since this standard is relatively fixed and open.

This document does not provide recommendations for the underlying access control system which would allow or deny a particular action.

Also not discussed here is login and authorization to be used for initial access to a service provider.

2. The Services

2.1. Overview

There are a few common services types that will be needed for minimum useability of a groupware server or application. They are:

- Contacts
- Schedule
- Notes

- Todo

These services are represented already in places such as existing groupware client-server applications and also in the PalmOS basic-4 buttons and applications. Different systems may have different names for these services internally, e.g. Contacts - addresses, addressbook, people, Schedule - calendar, agenda, meetings.

Within each of these services are some common methods that would be called to store, retreive, or update data:

- read_list
- read
- save
- delete

2.2. Detail

2.2.1. Contacts

The concept of contacts may encompass local addressbooks, LDAP, and lists stored in other media. The purpose of the contacts service is not to duplicate or attempt to replace these. In some respects, it might do just that. But its goal is more so to provide a common and shareable way for the other core services to create, edit, and read a common user and address list. All of the other services may use the contact service to obtain record owner information to be used in access control. They would also use them when it is required to share this data, as with a meeting where other local and non-local users will be invited to attend.

Contacts may include the local installed user base, users on other cooperative servers, or email addresses used for limited cooperation with other groupware services that are not compliant with this service scheme or implementations thereof. It could also include individuals using web-based or local ISP email services. The scope of this document, however, is to define the service with regard to the common methods to be used for server-server and client-server communications:

- read_list

This method is used to list contacts, with or without limits, filters, or search criteria. In this way it can be used for simple lists or to search for contact records and their identifiers. The optional search criteria includes:

1. start - Start at this identifier (integer: default 0)

2. limit - Limit to this number of records returned(integer: unlimited by default)
3. fieldlist - limit to showing only these fields (array: default to identifier, owner identifier, possibly firstname and lastname)
4. filter - Show records that are public or private only, or other system-specific filters, e.g group or company(string: default ”)
5. query - Search internal fieldlist for a value (string: default ”)

The return for this method includes:

1. count of number of records returned(integer)
2. array consisting of: array: identifier => (array: fieldlist key => value pairs)

- read

Once the identifier for a single contact record is known, the contact may be read for more detail using this method. This takes two parameters:

1. identifier - (integer: no default)
2. fieldlist - limit to showing only these fields (array: default to identifier, owner identifier, possibly firstname and lastname)

And returns:

1. array consisting of: array: identifier => (array: fieldlist key => value pairs)

- save

This is a method used to save an existing record or create a new one. If the identifier for an existing record is not passed, a new entry will be created.

- delete

This will allow deletion of a record by passing its identifier.

2.2.2. Schedule

2.2.3. Notes

2.2.4. Todo

2.3. Examples in XML-RPC

Query the contacts service for read_list, using only start and limit to grab the first 5 records, starting with identifier 1. Additionally, return only the firstname and lastname fields n_given and n_family (firstname and lastname in pseudo vcard format):

```
<methodCall>
<methodName>service.contacts.read_list</methodName>
<params>
<param>
<value><struct>
<member><name>start</name>
<value><string>1</string></value>
</member>
<member><name>limit</name>
<value><string>5</string></value>
</member>
<member><name>fields</name>
<value><struct>
<member><name>n_given</name>
<value><string>n_given</string></value>
</member>
<member><name>n_family</name>
<value><string>n_family</string></value>
</member>
</struct></value>
</member>
<member><name>query</name>
<value><string></string></value>
</member>
<member><name>filter</name>
<value><string></string></value>
</member>
</struct></value>
</param>
</params>
</methodCall>
```

3. Conclusion

This document outlined the following services and methods:

3.1. Contacts:

- service.contacts.read_list([search criteria])
- service.contacts.read(identifier,[fieldlist])
- service.contacts.save(fields)
- service.contacts.delete(identifier)

3.2. Schedule:

- service.schedule.read_list([search criteria])
- service.schedule.read(identifier,[fieldlist])
- service.schedule.save(fields)
- service.schedule.delete(identifier)

3.3. Notes:

- service.notes.read_list([search criteria])
- service.notes.read(identifier,[fieldlist])
- service.notes.save(fields)
- service.notes.delete(identifier)

3.4. Todo:

- service.todo.read_list(search criteria)
- service.todo.read(identifer,[fieldlist])

- service.todo.save(fields)
- service.todo.delete(identifier)