ejabberd: ejabberd integration with XMLRPC API

Description

mod_rostbeef is a module for ejabberd, an XMPP/Jabber server written in Erlang. It starts an XML-RPC server and waits for external requests.

Implemented calls include management for *mod_roster_odbc* and some additional user administration, like sending stanza to a specified user.

This allows external programs written in any language like websites or administrative tools to communicate with ejabberd to get information or to make changes without the need to know ejabberd internals.

Some benefits of interfacing with the Jabber server by XML-RPC instead of modifying directly the database are:

- external programs are more simple and easy to develop and debug
- can communicate with a server in a different machine, and even on Internet

Available methods

Note on return codes

Response code integer contains either 0 or the response code of ejabberd in case of error (Example: 409 returned when creating a user means "conflict"). Error code mapping is defined in JEP-0086

Rostermanagement

add_rosteritem

Add an entry in a user's roster.

```
Integer add_rosteritem(String user, String server, String jid, String group, String
nick, String subs)
```

jid is the JabberID of the user you would like to add in *user* roster on the *server*.
subs is the state of the roster item subscription. It can be either *both*, *to*, *from* or *none*. *none* means that presence packets are not send between parties. *both* means that presence packets are send in both direction. *to* means that the user see the presence of the given JID. *from* means that the JID specified sees the user presence.
Do not forget that roster items should be kept symmetric: when adding a roster item for a user, you have to do the symmetric roster item addition.
The roster is cached on the client and thus is only updated when the user logs again into the server.
group is the name of the group the added jid should belong to
nick the given nickname for the rosteritem

delete_rosteritem

Remove an entry for a user roster.

```
Integer add_rosteritem(String user, String server, String jid, String group, String
nick, String subs)
```

For *jid, subs, user, server, group, nick* see above. Do not forget that roster items should be kept symmetric: when removing a roster item for a user, you have to do the symmetric roster item removal. This mechanism bypass the standard roster approval addition mechanism and should only be used for server administration or server integration purpose.

link_contacts

Add a symmetrical entry in two users roster.

```
Integer link_contacts(String jid1, String nick1, String jid1, String nick2)
```

jid1 is the JabberID of the user1 you would like to add in user2 roster on the server.
nick1 is the nick of user1
jid2 is the JabberID of the user2 you would like to add in user1 roster on the server.
nick2 is the nick of user2

This mechanism bypass the standard roster approval addition mechanism and should only be userd for server administration or server integration purpose.

unlink_contacts

Remove a symmetrical entry in two users roster.

```
Integer link_contacts(String jid1, String jid1)
```

jid1 is the JabberID of the user1
jid2 is the JabberID of the user2

This mechanism bypass the standard roster approval addition mechanism and should only be used for server administration or server integration purpose.

get_roster

Retrieve the roster for a given user.

```
Array of Struct(String jid, String group, String nick, String subscription, String
pending) get_roster(String user, String server)
```

The function returns a list of the contacts in a user roster.

The function also returns the state of the contact subscription. Subscription can be either "none", "from", "to", "both". Pending can be "in", "out" or "none".

get_roster_with_presence

Retrieve the roster for a given user. The retrieved roster includes presence information.

```
Array of Struct(String jid, String group, String nick, String subscription, String
pending, Array of Struct(String resource, String show, String status, Integer priority)
presence)
get_roster_with_presence(String user, String server)
```

resource resource of the user
show this value contains the user presence flag. It can take limited values:
available, chat (Free for chat), away, dnd (Do not disturb),
xa (Not available, extended away), unavailable (Not connected)
status is a free text defined by the user client.

Priority is the resource priority as integer.

For *jid, user, server, group, nick* see above(add_rosteritem). The function also returns the state of the contact subscription. Subscription can be either "none", "from", "to", "both". Pending can be "in", "out" or "none".

Usermanagement

get_presence

Retrieve the resources and it's presence (show, status message and priority) for a given user.

```
Array of Struct(String jid, String show, String status, Integer priority)
get_presence(String user, String server)
```

jid	This value contains the user jid with resource.
resource	resource of the user
show	this value contains the user presence flag. It can take limited values: available, chat (Free for chat), away, dnd (Do not disturb), xa (Not available, extended away), unavailable (Not connected)
status	is a free text defined by the user client.
Priority	is the resource priority as integer.

get_resources

Retrieve all available resources for a given user.

```
Array of String get_resources(String user, String server)
```

send_stanza

Send stanza to a given user

```
Integer send_chat(String from, String to, String stanza)
```

If Stanza contains a "*from*" field, then it overrides the passed *from* argument.

If Stanza contains a "*to*" field, then it overrides the passed *to* argument.

Erlang XML-RPC formalism

For developers willing to test or implement the XML-RPC call in an Erlang client, here is the Erlang XML- RPC structures to use:

Users administration

Call	Arguments	Returns
link_contacts	struct[{jid1, String}, {nick1, String}, {jid2, String}, {nick2, String}]	Integer
unlink_contacts	struct[{jid1, String}, {jid2, String}]	Integer
get_roster	struct[{user, String}, {server, String}]	array[struct[{jid, String}, {group, String}, {nick, String}, {subscription, String}, {pending, String}]]
get_roster_with_presence	struct[{user, String}, {server, String}]	array[struct[{jid, String}, {resource, String}, {group, String}, {nick, String}, {subscription, String}, {pending, String}, {presence, array[struct[{resource, String}, {show, String},{status, String}, {priority, Integer}]]
get_presence	struct[{user, String}, {server, String}]	array[struct[{jid, String}, {show, String}, {status, String}, {priority, Integer}]]
get_resources	struct[{user, String}, {server, String}]	array[String]
send_stanza	struct[{from, String}, {to, String}, {stanza, String}]	Integer
add_rosteritem	struct[{from, String}, {to, String}, {stanza, String}] struct[{user, String}, {server, String}, {jid, String}, {group, String}, {nick, String}, {subs, String}]	Integer
delete_rosteritem	struct[{user, String}, {server, String}, {jid, String}]	Integer