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 roster and some user administration. 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

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 resource, String group, String nick, String
subscription, String pending, String show, String status)
get_roster_with_presence(String user, String server)
```

The 'show' 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)

The 'status' is a free text defined by the user client.

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".

Note: If user is connected several times, only keep the resource with the highest non-negative priority

get_presence

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

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

The '*jid*' value contains the user jid with resource.

The 'show' 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)

The 'status' is a free text defined by the user client.

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.

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

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}, {show, String},{status, String}]]
get_presence	struct[{user, String}, {server, String}]	struct[{jid, String}, {show, String}, {status, String}]
get_resources	struct[{user, String}, {server, String}]	array[String]
send_stanza	struct[{from, String}, {to, String}, {stanza, String}]	Integer