XMPP in KPR

Introduction

The Extensible Messaging and Presence Protocol (XMPP) is an open technology for real- time communication, using the Extensible Markup Language (XML) as the base format for exchanging information.

It provides different services like, Channel encryption, Authentication, Presence, Contact lists, One-to-one-messaging, Multi-party messaging, Notifications, Service discovery, Capabilities advertisement, Structured data forms, Workflow management, Peer-to-peer media session.

The purpose of XMPP in KPR is to provide a simple to use abstraction to build XMPP client. To reach that goal, the implementation hides from the client developer, all the complication of the session setup (including, for example, the channel encryption and the authentication), and the maintenance of the connection with the XMPP Ping protocol.

The user has to deal with application level stanza only. The client can send stanza to the server and receive stanza from the server. All stanza are manipulated as DOM element. This provides an unified way to create valid XML stanza from the point of view of both the incoming and the outgoing XML streams.

API

Constructor

SYNTAX

```
new XMPP(behavior, domain, user, password)
new XMPP(behavior, domain, user, password, resource)
```

PARAMETERS

- `behavior`: The behavior that will receive events specific to the XMPP instance, like on`XMPPConnected` or on`XMPPReceived` (see below);
- `domain`: domain name of the XMPP server;
- `user`: user name used for the XMPP session (case insensitive);
- `password`: password used for the XMPP session;
- `resource`: tentative resource name of the XMPP session. If omitted, the machine name is used.

DESCRIPTION

The constructor function creates an instance object that is used to connect and manage a session with a XMPP server.

Properties

SYNTAX

```
xmpp.document
```

DESCRIPTION

Returns the DOM document of the outgoing XML stream. It allows creating new stanza element for sending to the server in the context of the negotiated stream session (see example below). This property is read only.

SYNTAX

```
xmpp.bareJID
```

DESCRIPTION

Returns the bare JabberID of the session (`user@domain`). This property is read only.

SYNTAX

```
xmpp.fullJID
```

DESCRIPTION

Returns the full JabberID of the session (`user@domain/resource`). This property is read only.

SYNTAX

```
xmpp.nextID
```

DESCRIPTION

Returns a string containing the next ID that should be used in iQ stanza. It ensures that the ID used in iQ stanza is unique. This property is read only.

SYNTAX

```
xmpp.resource
```

DESCRIPTION

Returns the resource of the session. The resource value used when initializing the object can be modified by the server during the session negotiation. This property is read only.

Methods

SYNTAX

```
xmpp.connect(authority)
```

DESCRIPTION

Negotiates the session with the server. The negotiation can involve different steps, like:

- Switching to a secured connection (STARTTLS);
- Authenticating to the server (PLAIN or DIGEST-MD5);
- Retrieving the negotiated resource ID for the session;
- Starting the session.

If the negotiation succeeds, the on`XMPPConnected` callback is called. If the negotiation fails, the on`XMPPError` callback is called with the appropriate error code.

If no authority is provided, the domain is used and the server is selected automatically. Otherwise, the authority specifies the server and port to connect to. For example:

```
xmpp.connect("ec2-184-73-210-3.compute-1.amazonaws.com:443");
```

SYNTAX

```
xmpp.disconnect()
```

DESCRIPTION

Ends the session and disconnect from the XMPP servers.

SYNTAX

```
xmpp.send(stanza)
```

PARAMETERS

- `stanza`: The stanza is a DOM element created from the `xmpp.document`.

DESCRIPTION

Allows to send stanza on a connected session. The `stanza` is a DOM element created from the `xmpp.document`. See example below.

Callbacks

The behavior specified in the XMPP constructor receives the following callbacks:

SYNTAX

```
onXMPPConnected(xmpp)
```

PARAMETERS

- `xmpp`: The XMPP instance.

DESCRIPTION

The callback is called when the session negotiation succeeded.

SYNTAX

```
onXMPPError(xmpp, error)
```

PARAMETERS

- `xmpp`: The XMPP instance.
- `error`: The error code.

DESCRIPTION

The callback is called when there is an error during the negotiation of the session, when the session is interrupted by the server, or when low level error occurs. The error code are for example:

- `XMPP.SocketNotConnectedError`: The socket connection failed;
- `XMPP.SSLHandshakeFailedError`: The TLS negotiation failed;
- `XMPP.AuthFailedError`: The authentication to the XMPP server fails;
- `XMPP.UnimplementedError`: The authentication mechanism is unimplemented. Currently, PLAIN and DIGEST-MD5 are implemented.

SYNTAX

```
onXMPPReceived(xmpp, stanza)
```

PARAMETERS

- `xmpp`: The XMPP instance.
- `stanza`: The stanza is a DOM element created from the incoming XML stream document.

DESCRIPTION

The callback is called when a new stanza is received from the XMPP server.

Example

Here follows a simple example of a script that:

1. Connects to a Google Talk server;
2. Asks for the rosters;
3. Processes the rosters;
4. Initiates the presence mechanism;
5. Disconnects from the server.

The different steps are identified in the comments of the code sample.

```
var behavior = {
    onXMPPConnected: function(xmpp) {
        // 2. Asks for the rosters
        var document = xmpp.document;
        var stanza = document.createElement("iq");
        stanza.setAttribute("from", xmpp.fullJID);
        stanza.setAttribute("id", rosterID);
        stanza.setAttribute("to", xmpp.bareJID);
        stanza.setAttribute("type", "get");
        var query = document.createElement("query");
        query.setAttribute("xmlns", "jabber:iq:roster");
        stanza.appendChild(query);
        xmpp.send(stanza);
    },
    onXMPPError: function(xmpp, error) {
        trace("XMPP " + xmpp.fullJID + " error " + error + "\n");
    },
    onXMPPReceived: function(xmpp, stanza) {
        if (stanza.tagName == "iq") {
            if (stanza.getAttribute("id") == rosterID) {
                if (stanza.getAttribute("type") == "result") {
                    // 3. Processes the rosters
                    var query = stanza.getElementNodeNS("jabber:iq:roster", "query");
                    var items = query.getElementsByTagName("item");
                    var c = items.length;
                    for (var i = 0; i < c; i++) {
                        var item = items.item(i);
                        trace("Roster: " + item.getAttribute("name") + " -> " +
                            item.getAttribute("jid") + "\n");
                    }
                    // 4. Initiates the presence mechanism
                    var document = xmpp.document;
                    stanza = document.createElement("presence");
                    xmpp.send(stanza);
                }
            }
        }
    }
};

// 1. Connects to a Google Talk server
var xmpp = new XMPP(behavior, "gmail.com", "user", "password");
var rosterID = xmpp.nextID;
xmpp.connect();

...
// application processing
...

// 5. Disconnects from the server
xmpp.disconnect();
```