Elvin Presence Protocol

Version 1.0 (18/12/2003)

Author: Matthew Phillips (Defence Science and Technology Organisation)
matthew.phillips@dsto.defence.gov.au

Contributors: David Arnold (DSTC), Bill Segall (DSTC), Martin Wanicki (Boeing), Ian Lister (DSTC), Julian Boot (DSTC), Anna Gerber (DSTC), David Cox (Mincom)

Contents

Notes About This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in IETF RFC 2119.

Changes to the previous draft specification are highlighted in green.

1 Aim

The aim of the presence protocol is to provide a simple and flexible means for Elvin users to communicate their presence online and publish information they may wish to have known about themselves.

2 Requirements

Presence services are already provided to a greater or lesser extent by a number of applications, including ‘finger’, CoffeeBiff and various Instant Messaging (IM)

clients such as Jabber and ICQ. The primary reasons for, and advantages of, providing presence information via the Elvin presence protocol are:

- Does not require centralised, persistent state.
- Simple and thus easy to support.
- Separate, but complementary to, ticker messaging.
- Extensible.

3 Specification

This section describes the protocol, including the operations between clients and the format of the presence notifications.

3.1 User Identity

The presence protocol assumes that users are identified by a combination of *name* and *domain*, both of which are case-insensitive strings containing any character except '@'. The combination of *name@domain* forms a unique *user ID* for the user in 'presence space' in the same manner as an email address. Apart from its role as a namespace, the domain plays no other role in the presence protocol.

3.2 Presence Groups

A group is a namespace containing a set of online users. Every online user MUST be in at least one group and may opt to be in several. All users within a group will be aware of each other's presence by default. A group can be thought of as a public 'buddy list' that is created by people opting to be members. When users want to add people outside their main group to their buddy list, they have two options: explicitly add the other user by name, or join a group that the other person is in.

For example, all the people working in the Frob Testing Division of Bob's Frobs Inc, may elect to be in the 'ftc.bobco' group. The manager of FTC might also elect to be in the 'management.bobco' group so he or she can maintain contact with the managers of other divisions.

As well as providing a way to partition presence information into communities, the group concept is the key to scalability of the presence protocol by making it feasible for clients to go beyond subscribing to a fixed set of online 'buddies' and subscribe to dynamic presence groups smaller than entire online world.

3.3 Operation

The presence protocol operates by exchanging two types of notifications:

- *Presence-Info*, and
- *Presence-Request*

Clients use Presence-Info notifications to publish presence information about themselves, and use Presence-Request to trigger other clients to generate info notifications.

Clients subscribe to Presence-Request notifications, and generate Presence-Info notifications in response. Clients also spontaneously emit Presence-Info when they start up and when any of the presence information previously published has changed.

A number of fields in Presence-Info are optional, meaning they may be left out when their values are the same as they were in the last full notification. In general, clients should never assume that any of the optional fields in a presence notification will be present, and should treat a missing field as indicating the value of that field is the same as when it was specified previously.

3.4 Example Exchange

Note: see sections 3.6 & 3.7 for definitions of the Presence-Request and Presence-Info notifications.

Scenario: Zaphod@hitch-hikers is already online. Frodo@hitch-hikers, starts up a presence-capable client. Both are in the 'hitch-hikers' group.

Frodo initially announces he is online with the following full presence info notification:

```
    Presence-Info: initial
Presence-Protocol: 1000
        Client-Id: ffffff-111111
             User: Frodo@hitch-hikers
           Groups: |hitch-hikers|
           Status: online
      Status-Text: Online
  Status-Duration: 0
      Chat-Groups: |Chat|ticker-dev|
      News-Groups: |BreakingStories|Slashdot|
    Ticker-Client: frodo-ticker 1.0
       User-Agent: frodo-online 1.0
```

Zaphod's client, being subscribed to presence info in for the hitch-hikers group, receives the information about Frodo and updates its registry.

Frodo's client now populates its registry by:

1. Subscribing to presence information for the hitch-hikers group using the expression:

   ```
   require (Presence-Info) && User != "Frodo@hitch-hikers"
     contains (fold-case (Groups), "|hitch-hikers|")
   ```

   Note that, in order to avoid complications that may arise from Frodos' own notifications being echoed back to him, Frodo's client has added an appropriate "User != …" clause to the subscription.

2. Sending a request for presence information about people in the current group. The presence request notification Frodo sends is:

   ```
    Presence-Request: cafebabe12345
   Presence-Protocol: 1000
           Requestor: Frodo@hitch-hikers
              Groups: |hitch-hikers|
               Users:
   ```

All clients in the group, including Zaphod's, receive the request since they have subscribed using an expression like:

```
Require (Presence-Request) &&
  (contains (fold-case (Groups), "|hitch-hikers|") ||
   contains (fold-case (Users), "|zaphod@hitch-hikers|"))
```

Zaphod's client responds with a full presence info notification tagged with the request ID from Frodo's message:

```
    Presence-Info: cafebabe12345
Presence-Protocol: 1000
        Client-Id: zz9pza-222222
             User: Zaphod@hitch-hikers
           Groups: |hitch-hikers|
           Status: online
      Status-Text: Online (going for coffee at 3pm)
  Status-Duration: 42
      Chat-Groups: |Chat|ticker-dev|
      News-Groups: |BreakingStories|Slashdot|
    Ticker-Client: zticker 3.0
x-Number-Of-Heads: 2
```

Frodo's client updates its registry, at which point Zaphod, Frodo, and any other interested clients are in sync.

Later, when Zaphod decides to go for coffee, he hits the "Coffee!" button on his client, which sends this partial presence notification:

```
    Presence-Info: update
Presence-Protocol: 1000
        Client-Id: zz9pza-222222
             User: Zaphod@hitch-hikers
           Groups: |hitch-hikers|
           Status: unavailable
      Status-Text: Coffee!
  Status-Duration: 0
```

This is an example of a partial notification: Zaphod's client has chosen to only include the optional status fields that have changed since the last full notification was emitted. This is an optional, but recommended, client optimisation.

3.5 Example With Multiple Groups

For simplicity, the previous example assumed Frodo was in a single group. The following example (shown from Frodo's end only) illustrates how his client would support Frodo being both in the 'hitch-hikers' group and the 'hobbits' group. The key changes from the previous example are highlighted in bold.

Frodo's client would announce he is online with:

```
    Presence-Info: initial
Presence-Protocol: 1000
        Client-Id: ffffff-111111
             User: Frodo@hitch-hikers
           Groups: |hitch-hikers|hobbits|
           Status: online
      Status-Text: Online
  Status-Duration: 0
      Chat-Groups: |Chat|ticker-dev|
      News-Groups: |BreakingStories|Slashdot|
    Ticker-Client: frodoticker 1.0
```

Frodo's client subscribes to Presence-Request's using this expression:

```
require (Presence-Request) &&
  contains (fold-case (Groups),
     "|hitch-hikers|", "|hobbits|") ||
  contains (fold-case (Users), "|frodo@hitch-hikers|")
```

3.6 Presence Request Fields

Field	Type	Description	Possible values	When Used
Presence-Request	String	Marks the notification as a presence request and carries a request ID.	A random, unique request ID, not longer than 255 characters, with no spaces. The ID MUST NOT be “initial” or “update”.	Always
Presence-Protocol	Int32	Same as Presence-Info		Always
Groups	String	Allows a client to request status info of all users in a set of groups. Clients should respond to requests when this field contains a group the user is a member of.	A set of groups in the same format as Presence-Info. If no groups are being requested, this field should be left empty.	Always
Users	String	Allows a client to request status info of particular users. Clients should respond to requests when this field contains their username.	A set of usernames delimited by “|”. Eg “|frodo@home|bob@bobco”. If no specific users are being requested, this field should be left empty.	Always
Requestor	String	The username of the person requesting the presence information.	A username in *user@domain* form.	Optional

3.7 Presence Info Fields

Field	Type	Description	Possible values	When Used

Field	Type	Description	Possible values	When Used
Presence-Info	String	Marks this as a presence info notification and specifies its subtype	• "initial": Full info, with all fields that the client supports populated. Generated when the client starts and broadcasts initial presence info. • "*request_id*": Full info (all fields that the client supports are populated) in response to a Presence-Request with ID *request_id*. • "update": Partial info, with only the optional fields that have changed since last update included. Generated when presence data (eg Status-Text) changes.	Always
Presence-Protocol	Int32	The version of the presence protocol that is used in this notification.	An integer built using the formula: *major_version* * 1000 + *minor_version*. Eg. 2001 = version 2.1 of the protocol, 3093 = version 3.93 of the protocol. The usual compatibility rules for major/minor versions apply: major version number changes indicate incompatible changes to the protocol, minor number increments indicate backward-compatible changes to the protocol.	Always
Client-Id	String	Identifies the client instance that generated the response. This can be used by receivers to resolve conflicting presence information from users that are running multiple clients - see section 3.8 for a discussion of this issue.	Any string that uniquely identifies the client and which remains constant across the period between client initialisation (sending the first presence response) and client shutdown (sending an "offline" status notification).	Always

Field	Type	Description	Possible values	When Used
User	String	The user's username.	Any username: eg "mpp@dsto", "bob@bobco", etc.	Always
Groups	String	The groups the user is a member of.	A set of group names separated by '|'. For example "|dsto|quake heroes|". Note that the preceding and trailing '|'s allow clients to use the 'contains' function to easily test for particular groups without having to handle three cases (where the group is at the beginning, middle and end of the list).	Always

Field	Type	Description	Possible values	When Used
Status	String	The status of the user.	• "online": the user is online and ready to receive messages. • "unavailable?": the client has made an educated guess that the user is unavailable. This is often the result of the client not seeing any keystroke or mouse activity from the user for a given period of time. • "unavailable": the user is not currently viewing messages (eg away from desk). • "offline": the user has disconnected. This is usually sent when a presence client shuts down, but may be sent whenever the user no longer "exists" (eg if the user changes their name). • "coffee": the user is on a coffee break. This special status, which is functionally equivalent to "unavailable", is intended to support CoffeeBiff-style client interactions. The status settings SHOULD be taken into account by the client when merging conflicting notifications: see section 3.8.	Optional

Field	Type	Description	Possible values	When Used
Status-Text	String	Text describing the status of the user.	Any string intended to be displayed as the current status eg “Online: but will be away this afternoon”.	Optional: required when the Status field is present
Status-Duration	Int32	The amount of time (in seconds) that the current status has been in effect. This is a relative duration rather than an absolute time since the status was set to avoid the perennial problems with inaccurate clocks and time zones.	Any integer >= 0. Undefined behaviour if anyone holds the same status longer than 136 years ;) (2^{32} seconds)	Optional: required when Status field is present
Chat-Groups	String	The set of chat groups the user is subscribed to.	Eg. “|Chat|ticker-dev|”	Optional
News-Groups	String	The set of news groups the user is subscribed to.	Eg. “|BreakingStories|Slashdot|”	Optional
Ticker-Client	String	The ticker client the user is running	A ticker client name and version eg “sticker 2.0.1”, “xtickertape 2.0a”. A missing value indicates the user is not running a ticker client. Users that are running ticker but which don’t wish the type of client to be known should use “generic”.	Optional
User-Agent	String	The name of the presence client that generated this notification.	Any string value identifying a presence client, or “generic” if this information is being hidden.	Optional

3.8 Handling Conflicting Presence-Info Responses

Clients SHOULD handle the situation where multiple "overloaded" Presence-Info responses are received for single user by using the Client-Id and Status fields. For example, in the case where a user is running two clients on two hosts, one of which is listing the user as "unavailable?", the other which lists the user as "online", Presence-Request's will generate two responses with two different Client-Id's. In this case, clients SHOULD give precedence to the information in the "online" notification response over the "unavailable?".

A straightforward way of achieving this is for clients to maintain a registry of users that assigns to each unique User a set of received Presence-Info's, one per unique Client-Id. For each user, the client selects the Presence-Info with the "most active" status as the authoritative info for that user. In the case where some statuses are equal, the client selects the Presence-Info with the smallest Status-Duration. The order of precedence for the Status field (from "most active" to "most inactive") is "online", "unavailable?", "unavailable", "coffee" and "offline", with any other (illegal) statuses being considered to fall between "unavailable" and "offline" (ie offline is the "most inactive" state a user can be in).

3.9 Notes

- Presence-Info: the values of this field have been designed so that clients can easily discern the derivation of updates if they wish (for efficiency or otherwise). However, clients are not required to distinguish between the different types of Presence-Info notification: simply updating a registry with the new data in any Presence-Info notification is sufficient, so long as missing optional fields are handled.

  Note that one possible efficiency enabled by this approach is the ability to opt out of redundant "reply" updates in response to a Presence-Request from other clients, using the expression `Presence-Info != "update" && Presence-Info != "initial"`. The other obvious way to mark replies to status requests -- with a separate "In-Reply-To" field -- makes this difficult, since `!require (In-Reply-To)` can't be used to create the same effect due to the tri-state logic used in the Elvin subscription language.
- Presence-Protocol: this is designed so that clients can subscribe to known protocol sets eg "`Protocol-Version >= 1000 && Protocol-Version < 3000`" picks up any 1.x or 2.x protocol notifications.
- It is assumed that, if the user is running a ticker client, then they are subscribed to their 'personal' chat group with the same name as their user ID ie "user@domain". This convention makes it obvious how to send a ticker message to someone visible via the presence system.
- Non-standard fields should use the "x-" naming convention eg "x-Phone-Number", "x-Organisation", etc. Ideally, clients should provide an option to display the non-standard fields.
- Treating usernames or groups as case-sensitive runs the risk of people not seeing each other because they got the case wrong eg I look for 'bob@bobco' and fail to see him because he is really 'Bob@BobCo'.
- Requestor field: this was initially intended to allow clients to respond differently depending on the requestor, eg subscriptions to some groups might

be hidden except to certain people. But since any client can listen in to the replies, it would not be a particularly useful way of hiding information (using the security API's would make more sense). It is left in the specification but made optional.

4 References

4.1 Other Presence Protocols

Thanks to David Arnold for providing the initial set of links that these references are based on.

4.1.1 Simple General Awareness Protocol (SGAP)

A Lotus-sponsored IETF Internet Draft, which expired on May 98. It's not clear if this is still a live project. Specification talks mainly about client/server interactions and message formats. Seems to offer little wisdom for this specification.

Also see Lotus Research Page for more information.

4.1.2 Instant Messaging and Presence Protocol (IMPP)

Uses presence information messages (MIME type message/cpim) that contain an XML presence info 'payload'.

Example payload (from IMPP internet draft):

```
<presence xmlns="http://www.ietf.org/ns/cpim-pidf-xml-1.0">
 <tuple name="im-1">
   <status>
     <value>open</value>
     <detail type="im"
             schema="http://www.ietf.org/dtd/im-type-im.dtd">
        away
      </detail>
   </status>
   <contact priority="2">im:shingo@jp.fujitsu.com</contact>
   <note>I'll be in Tokyo tomorrow</note>
 </tuple>
 <tuple name="email">
   <status>
     <value>open</value>
   </status>
   <contact priority="1">mailto:shingo@jp.fujitsu.com</contact>
 </tuple>
</presence>
```

The main wisdom to be gained here is that the presence info is divided into one or more contact tuples, which describe the users' status in a particular online medium. In the example, the user is saying they have both Instant Messaging and email contacts, but are not online for IM. The addition of the note to the IM tuple presumably informs interested parties that the person will be available for chat the next day. The Status-Text field of this protocol gives some of this functionality, but we are really assuming that ticker is the main messaging method. Email addresses and other info can be supplied as an "x-" field.

For more info see IMPP charter page, IMPP main page, IMPP working group pages, IMPP Requirements and Lotus Research.

4.1.3 Jabber

Jabber is an IM/presence service built from a series of federated servers. The protocol is open and there are a number of open-source clients. A free trialware server for up to 100 users is available, but a commercial license is needed for larger numbers of users.

The XML-based Jabber protocol includes a presence message type. Example (from protocol page):

```
<presence
    to="romeo@montague.com/orchard"
    from="juliet@capulet.com/balcony">
  <status>Stay but a little, I will come again.</status>
  <show>away</show>
</presence>
```

The distinction between 'show' and 'status' is interesting. Status is user-readable text and is used in the same way the protocol in this document. 'Show' can be:

chat	The client is available for immediate contact.
away	The client is online, but is momentarily away (e.g., at lunch or a meeting).
xa	The client is online, but has been inactive for a long time.
dnd	The client is in Do Not Disturb mode.

The Elvin presence protocol currently overloads *show* and *status* into the 'Status' field eg "Unavailable (at meeting in rm 56)" is the same as Jabber show = "away" + status = "at meeting in rm 56". We may want to consider following the Jabber example.

Jabber also has an "iq" message format can be used to exchange extended information.

See home page, main commercial page, Jabber Central news page

4.1.4 ICQ

The ICQ v5 protocol allows a range of, non-extensible (?) information about a user to be published using the CMD_META_USER message such as name, phone, age, city etc. This sort of information could be accommodated by the Elvin presence protocol using x-Name, x-Phone etc.

See ICQ protocol site.