

YAMI Event Server Documentation

Copyright © 2001-2008 Maciej Sobczak

Contents

1	Introduction	2
2	Command-line options	2
3	Interface	2
4	Example	3

1 Introduction

The *Event Server* is a simple, *subscription*-driven server for forwarding and broadcasting messages.

The *Event Server* allows other objects to subscribe to messages with different levels of genericity and it takes responsibility for delivering incoming messages to all subscribers, if the message matches the forwarding criteria for given subscriber. The *Event Server* is a generic server, which means that it does not interpret incoming messages and puts no constraint on their names or parameters.

The service is not persistent, which means that it does not retain any information considering its subscribers between subsequent runs.

2 Command-line options

The service can be started with the command:

```
$ ./eserver ipport [vq]
```

ipport is the port number on which the service's agent will listen for messages.

v means *verbose* – with this option the service will report its actions on the console.

q means *quit-enable* – this option causes the server to register the special object that allows the clients to shut the server down.

The **v** and **q** options are independent from each other.

3 Interface

When run, the *Event Server* creates its own agent and registers an object with the name `eventserver`.

This object accepts the following messages:

subscribe with the variable-length list of parameters. This message is used by remote objects to subscribe to messages. The first three parameters are mandatory, the fourth and fifth are optional:

1. address of the subscriber (string)
2. port of the subscriber (string or int)
3. level of the subscriber (string or int)
4. (if present) name of object (string)
5. (if present) name of message (string)

The first three parameters define the destination of the message. The last two parameters are *forwarding criteria* – the message will be forwarded only when the object and message names defined in the original incoming message match the two given names. The empty name matches every name.

The server replies with short string, which is a *subscription id*.

unsubscribe with one string parameter – the subscription id given at the time of subscription. The server sends empty reply as a confirmation.

When run in *quit-enable* mode, the server additionally registers an object with the name `eventserver_quit`. Any message sent to this object will cause the server to shut down.

The *Event Server* accepts also messages sent to it with different object names. These messages are relayed to all interested subscribers as one-way messages. The server does not reject nor reply to such messages.

4 Example

Let's say that the *Event Server* was started with the command:

```
$ ./eserver 12340 vq
```

This means that the server runs on port 12340, in *verbose* and *quit-enable* mode.

The following Python (the idea is the same in other languages) commands, executed on the same machine:

```
>>> from YAMI import *
>>> a = Agent(12345)
>>> a.domainRegister('e', '127.0.0.1', 12340, 2)
>>> m = a.send('e', 'eventserver', 'subscribe', \
... ['comp.mycompany.com', 12350, 2])
>>> m.getResponse()
['1']
>>> del m
```

send the new subscription to the server. The subscription is for *all* messages – every message sent to the server will be relayed to the given location. The subscription id is '1'.

The next commands:

```
>>> m = a.send('e', 'eventserver', 'subscribe', \
... ['comp.mycompany.com', 12350, 2, 'myobject'])
>>> m.getResponse()
['2']
>>> del m
```

send the new subscription. The subscription is only for messages sent to object 'myobject' – other messages will not be relayed in this subscription (but *can* be relayed by other subscriptions, even to the same destination). The new subscription id is '2'.

The next commands:

```
>>> m = a.send('e', 'eventserver', 'subscribe', \
... ['comp.mycompany.com', 12350, 2, 'myobject', 'mymsg'])
>>> m.getResponse()
['3']
>>> del m
```

also send the new subscription. This new subscription is only for messages sent to object 'myobject' *and* which name is 'mymsg' – other messages will not be relayed in this subscription (but *can* be relayed by other subscriptions). The subscription id is '3'.

Let's suppose now that the *Event Server* received the message addressed to object 'myobject' and which name is 'mymsg'. This message will be relayed *three* times to the same destination, since three subscription forward criteria match this message.

If the message received is addressed to the same object, but with different name, it will be relayed to the same destination *two* times, since two subscriptions match.

If the message is received for different object, it will be relayed only once.

The command:

```
>>> m = a.send('e', 'eventserver', 'unsubscribe', ['1'])
```

removes the first subscription from the server.

The last command:

```
>>> a.sendOneWay('e', 'eventserver_quit', 'quit')
```

shuts the server down. Note that the name of the message ('quit') is irrelevant.