Introduction

All eBuddy clients communicate with the eBuddy backend via sockets. eBuddy has defined a protocol, called the Ebuddy Message Format (EMF) to unify the IM messages from the various supported networks. Besides IM, the protocol is also used to do tasks like load-balancing, user targeting and bannering.

The essence of EMF is that it is an asynchronous protocol. There is a list of actions that a client can do and there is a list of messages that the clients can expect to receive at any time during the lifecycle of a session. Each action has a response signaling whether that action was successful. Most of the time the real response of an action will be returned some time in the future in the form of a message.

The protocol is line based, meaning that every response and every message is closed with a “\n”. A response is distinguished by a trailing hash (“#”) character. In the current version of the protocol responses have no correlation to requests other than the order they were send in.

For this assignment we won’t bore you with the full list of actions and messages that comprise the protocol but we do want to see how you implement connection handling, sending responses and handling asynchronous messaging. In addition to that we would like to see some simple widgets. We basically ask you to build a very simple client that does these things:

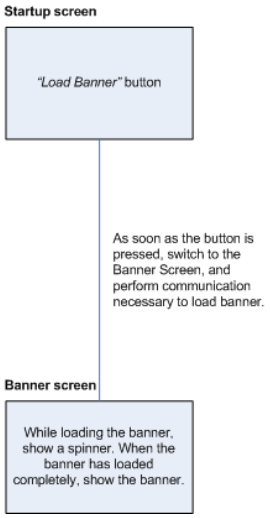
1. Setup a connection to the eBuddy server cluster, this includes load-balancing to the final server
2. Request a banner from the server (this is an aynchronous call)
3. Handle the banner data message
4. Show the banner on the screen of the phone.

Next to this we ask you to focus on properly handling an unstable mobile connection. By this we mean you have to make sure your application supports stuff like failing to setup a connection or experiencing packet loss and sudden disconnects. It’s up to you to decide whether you want to do this by either failing gracefully or by attempting to retry; whatever you feel suits the situation best.

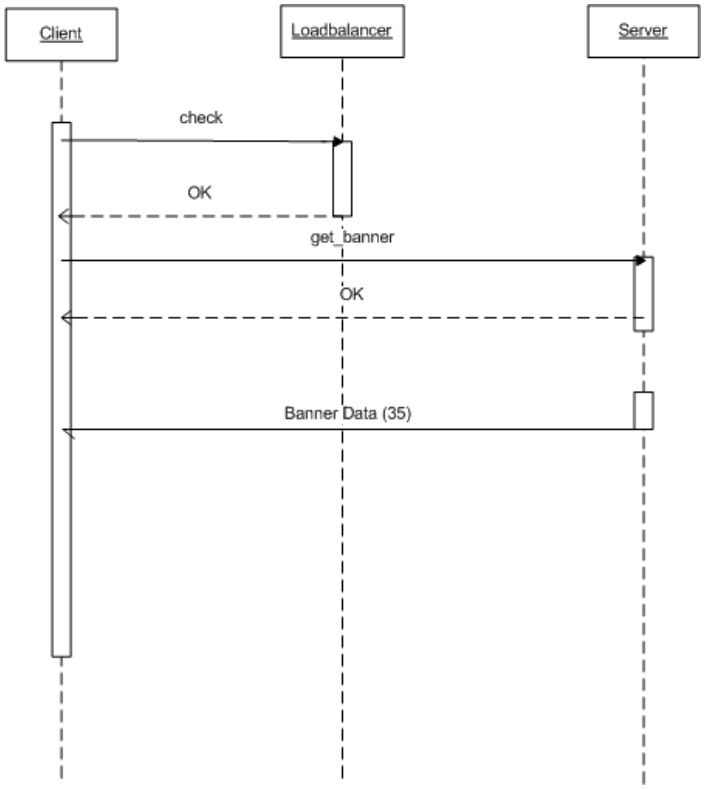
The next sections will give you an overview of the application.

Wireframes

User Interface



Application Flowchart



Description of steps

We will now describe the steps that need to be taken to accomplish the task.

Retrieve an available server to connect to

1. Create a socket connection to **s-connector.ebuddy.com** port **110**
2. On this socket connection do the following call:
   e_action=check;e_version=J2ME2;e_device=TestDevice;e_build=1.5;e_operator=TestOperator\n (don't forget the '\n'!)
3. Handle the response for this call, it will look like this:
   #e_result=OK;e_server={server}.ebuddy.com;e_port=110
   (if e_result=KO something is wrong) the EOL is again a '\n'
4. Disconnect the socket to s-connector.ebuddy.com
5. Connect to the new server using the values from *e_server* and *e_port*
6. On this new socket connection do the following call:
   e_action=get_banner;e_device=nokia_6300;e_version=J2ME2;e_format=png;e_ip=62.69.184.55;e_operator=dev;e_width=216;e_height=160\n
7. Handle the response for this call, it will look like this:
   #e_banner_id=e6db9d63-c4c9-4ef7-a3d7-9b48c8966ae2;e_result=OK;e_timeout=1000;
8. Handle the asynchronous BANNER_DATA message (see next section for detailed description

Handle BANNER_DATA message

Messages in EMF are defined as follows;

1. Every message is on one line and is delimited by a '\n'
2. Each message contains of several components, logically separated by a ':' . The components of each message are defined as follows:
   <type>:<timestamp>:<account>:<network>:<parameters>
   a. ***type:*** number of message, for instance 35 for get_banner (mandatory)
   b. ***timestamp:*** timestamp in milliseconds (mandatory)
   c. ***account:*** account name of the user (optional, can be empty)
   d. ***network:*** IM network (i.e. MSN) (optional, can be empty)
   e. ***parameters:*** semicolon (';') separated list of message parameters, different for every message type

The only message that you must handle for this assignment is the BANNER_DATA message, this message is type 35. The following is an example (this will all be on one line, for clarity we have inserted linefeeds for the parameters):

35:1239721671102:::
status=success;
banner_id=e6db9d63-c4c9-4ef7-a3d7-9b48c8966ae2;
type=1;
text=Metro%20-
%20Just%20a%20Lifestyle;link=http%3A%2F%2Fbank1.clicks.mp.mydas.mobi%2FhandleClick.php5%3Fapid%3D8812%26acid%3D1022%26auid%3D193.238.162.57%26osid%3D8%26urid%3D67fb552402dc9e8b004cadc559a8ace4%26ri%3D8%26mmid%3D0%26uip%3D62.69.184.55%26orut%3D1239721671;

As you can see, all parameters are by default urlencoded and need to be decoded by the client before they can be used.

The following is the description for the Banner Data message from the EMF documentation:

Banner Data

type = 35

Banner Data messages are send when a get_banner call was done by a client
The banner data message contains the data for the banner (either text or image) or a failure

The client should always check the value of the status param first! If this is error then only reason and banner_id are valid.

Name	Description
status	can be “success” or “error” (for now)
reason	Only available when status = “error”, contains the error message from the exception (e.getMessage())
banner_id	The correlation id for the banner, this should be correlated with the e_banner_id field of the get_banner response
type	The type of the banner: “0” for “NONE “1” for TEXT , “2” for IMAGE , “3” for HTML and “4” for “PC_ZONE” (i.e. invocation code)
link	Link to go to when the banner is clicked, available when type = “1” and type = “2”
text	The content of the TEXT banner, only available when type = “1”
html	The content of the HTML banner, only available when type = “3”
content_type	Content type of the IMAGE banner, only available when type = “2”
content_base64	Content of the banner image, base64 encoded. Only available when type = “2”. Decoded bytes are in the format specified by “content_type”param

The important parameter to check here is the *type* parameter. In this specific case only three values are important: 0,1 and 2. We will now explain how to handle the three types:

1. Type = 0 (NONE) there was no banner returned from the bannering subsystem, please show the text “No banner found”
2. Type = 1 (TEXT) the banner returned was a text banner, pleasse show the contents of the text parameter (properly url decoded)
3. Type = 2 (IMAGE) the banner returned was an image. The contents of the *content_base64* parameter need to be base64 decoded an the resulting image should be shown on the screen. The content type of the image can be derived from the *content_type* parameter.

Conclusion

This is the assignment, if anything is unclear please don’t hesitate to contact us. You should have been provided with the contact details of an eBuddy developer that will assist you with anything you need

Good luck!

eBuddy Development Team