FdAjax last version before 1.0

July 27th, 2007 by Grzegorz Daniluk

I am happy to release FdAjax module version 0.9. This is the last release before version 1.0. Version 0.9 doesn't have many new features and version 1.0 is going to have only bugs fixed if you find any.

FdAjax main goal is to free the application server (RoR, PHP) of keeping open connections to browsers. In my opinion this **goal was achieved** and I do not want to put too many functionalities into the module. Now a better documentation and promotion are needed.

Here are the changes in version 0.9

- A new fdajax.admin_password configuration parameter was added. After setting it all admin commands must have admin_passwd parameter added set to correct the password. As for now passwords are entered in clean text for simiplicity. If you have any suggestions how to improve this, post a comment.
- A bug was fixed which caused some messages removals before they were forwarded to clients.
- Source code licence was changed to BSD license so that the module might be integrated with main lighttpd sources.

FdAjax version 0.8

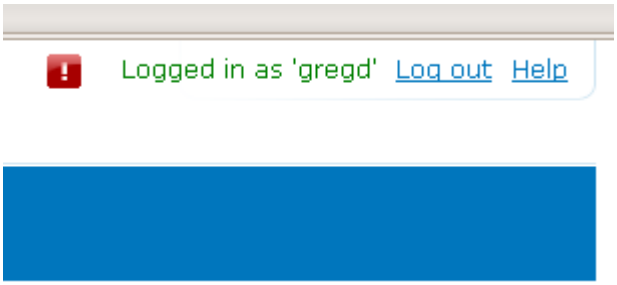
March 14th, 2007 by Grzegorz Daniluk

In version 0.8 we went one step further and we implemented two new functionalities: alerts and public group messages.

Alerts

In the previous article we were talking about a special event type "unw". It really works fine. When a message is not awaited by any window, it gets sent to the first window waiting for "unw" events. The window then displays this event as an alert, e.g. a flickering exclamation mark – it's enough to press on it and you get to see the information about an important event. This functionality is applied in Anslive to display the information about a new task offer when Refwell user is currently on a different web page, and not the one with the list of offers and prices.

In version 0.8, after the user goes to another window, the alert still remains active. It is sent immediately after the connection with the module is established. In order to deactivate the alert message, you need to send a command to cancel it. In this respect it's similar to **the Windows tray icons**.



It was possible to implement this functionality in a traditional way, by entering the information about alerts into the application database or user's session. There are, however, two reasons why the implementation of the alerts in the module is better:

1. Alerts display doesn't have to be implemented twice: first time in the code generating the website, and second – in the JS code displaying alerts coming via the FdAjax connection.
2. Storing persistent alerts information, concerning for example a chat message from one user, requires loading the other user's session, which is not that easy in all frameworks used for setting up web applications.

Public Group Messages

Up until now a message could be sent to one user only, which is enough for most applications. A large group of them, however, requires refreshing the main page that is identical for all visitors. It may be for example a site with stock exchange rates and indices.

FdAjax makes it easy now. Version 0.8 allows every visitor to establish a connection to the module. All you need to do is to set the id session in the browser cookie and you get automatically logged on to the module as a guest user_id = 0. Then you'll get logged on to the module with the first "wait" or "stream" command.

Sending a public group message is very similar to sending a regular message to a pre-specified user. You just don't enter the peer_id. Group messages have their own event types. A group message event type can't have an id object added. Names for group messages are as follows:

```
main gchat ques answ ofrs pric opin users ords geic feed mese netk task
frid info
```

For the "wait" or "stream" commands the user_id needs to be set on 0 for guest users. For the "types" parameter you need to add the name of the group event and it's done.

Other changes

First of all, a few new options were added to the module configuration. The old blog post with the commands description will be updated and it'll contain the description of new parameters.

Now a few words about fdajax.send_pings. Setting this option higher than 0 will result in the module sending to the browser the JS code stored in the fdajax.js_ping_code parameter after

send_pinds seconds. After the code is sent, the connection is closed even for the “stream” command.

This way you can forget the “cancel” command. You can add the parameter “force=1” to the “wait” or “stream” command, which will close the previous connection to the module (if it was open before). Because of this redirecting isn’t necessary when the user tries to connect to the same address (www.refwell.com). When the user closes the website with the anti-virus program still keeping the connection open and thus unnecessarily using the server resources, sending the ping code will close this connection. Of course, if the user has the website open all the time, a page JS code should start a new connection to the module.

The examples attached to the source code don’t use send_pings. They were added on the request of the users who use the module in various configurations.

One of early adopters of the module configured LightTPD and FdAjax in such a way that the whole application operates in a traditional way on the **Apache server. LightTPD works independently** and handles hundreds of connections to the FdAjax module. The page downloaded from Apache uses cross-side scripting to download the fdajax.js code from the LightTPD server. You just need to configure the addresses and session cookie domain path. A lot of people use LightTPD for static content serving when Apache + PHP generate the web pages.

Example

This is a small chat application which uses public group messages and chat_sig for message sending via the module only. Thanks to it a single server is able to serve hundreds and maybe even **thousands** of such messages every second. Which means that even if **all Digg users** at one moment decide to log on and start chatting, they still won’t be able to block the whole server.

Update: Many thanks to Sébastien Gruhier for providing compiled libraries for Apple OsX.

FdAjax PHP tiny tutorial

March 7th, 2007 by Grzegorz Daniluk

You installed and configured Lighttpd with fdajax module and PHP. First of all you have to check if the module loaded well. The port number is in lighttpd.conf.

```
server.port = 80
```

for port 80 open the page

```
http://localhost/live.fdajax?cmd=info
```

You can add the first user if the module page is displayed. You have to have a session id in order to log in a user on the module. Create a new page called tutorial.php with the following content:

```
<? session_start(); ?>
```

```
FdAjax PHP tiny tutorial<br>
Session id: <? echo session_id(); ?><br>
Session cookie: <? echo session_name(); ?><br>
```

Next run the page. Make sure that session cookie name is correctly set in the module configuration (fdajax.session_cookie). If not, you should correct the fdajax module configuration. You can log in the first user when the session cookie is the same as in the module configuration. Type the following address in a new browser window:

```
http://localhost/live.fdajax?cmd=login&user_id=1&ip_addr=127.0.0.1&sess_id=XXX
```

XXX stands for the session id that is on the tutorial.php page. The module should display: "LOGIN_OK". In a second browser window type:

```
http://localhost/live.fdajax?cmd=wait&user_id=1&win_id=1&types=chat
```

The page should start loading. It simply waits for an answer from the server. In the next window browser type:

```
http://localhost/live.fdajax?cmd=push&peer_id=1&types=chat&mes=hello_world
```

The module should display "MESSAGE_SENT". A message "hello_world" should be displayed in the first window. Now try a different command which does not end the connection:

```
http://localhost/live.fdajax?cmd=stream&user_id=1&win_id=1&types=chat
```

send data to this window just as in the first case.

```
http://localhost/live.fdajax?cmd=push&peer_id=1&types=chat&mes=1
```

then

```
http://localhost/live.fdajax?cmd=push&peer_id=1&types=chat&mes=2

http://localhost/live.fdajax?cmd=push&peer_id=1&types=chat&mes=3
```

the first window should display 123

Worked? Great! You're one of the first people to get to know the basis of FdAjax.

The wait command is used to communicate with the web server by using XmlHttpRequest. The stream command constantly loads the data from the server by placing a hidden IFRAME on the page.

Next week I'll give a more advanced example for PHP and now you can see how an exemplary application for Ruby on Rails is built. The application is included in the sources archive.

FdAjax configuration and commands

March 7th, 2007 by Grzegorz Daniluk

Introduction

The way the module works is simple. After the page is loaded the JS code opens a background connection to the Lighttpd server, which is intercepted by the FdAjax module. This connection is open all the time. When the application sends data to the module which is to be transfered to a given page, the module sends the data and closes the connection. It allows to neatly write web applications which have to reload real-time pages e.g. a chat application.

The module is designed to work with various programming languages (PHP, Python, Ruby on Rails).

Installation

The description of the source code compilation can be found in INSTALL file. I have included compiled versions for Ubuntu Linux and Mac for those of you who do not want to compile the module manually. The modules are compiled for Lighttpd 1.4.13. All you need to do copy them to the directory with other Lighttpd modules.

Configuration

The configuration of the module is simple. The lighttpd.conf file from the Minichat application can serve as an exemplary configuration. In fact, all the default settings in the Minichat work for other applications.

The two most important parameters to configure:

- **admin_ips** – a list of IP addresses from which one is allowed to send admin commands.
- **session_cookie** – the name of the cookie which stores the session ID.

Other parameters:

- **ext** – an extension used by the module. As default it is „fdajax" and you should not change it.
- **mes_live_max** – the numer of milliseconds which defines after what time the message that is to be sent is considered as expired. After that time the message is deleted (to be more precise it is deleted when the next command related to a given user is executed).
- **win_timeout** –the number of milliseconds which defines how long the module stores the information about the user's window after the connection to that window is closed. It is a time during which the JS has to open a new connection to the module after receiving messages from the server. After that time the module deletes data related to that window e.g. messages sent to that window. This option is important for chunked mode.
- **log_level** – defines which messages are saved in the lighttpd error log file. The value range is 0-3. 0- nothing is saved in the logs, 3- all messages are saved.

- **log_security** – defines which messages related to the authorization of users are saved in the logs. The values are the same as for log_level.
- **js_peer_func** – the name of the JS function which will be used to marshall a direct message from the other user. A detailed description below.
- **js_logout_code** –JS code which will be sent to all open windows when the user is logged out of the module.
- **js_maxwin_code** – JS code which will be sent to the earliest opened window by the same user if he/she opens more than 5 windows (connections to fdajax module).
- **js_nocookies** - JS code which will be sent as a reply to the user's command which does not contain session_cookie
- **js_notfound** – JS code which will be sent as a reply to the user's command if his/her ID is not found- the user is not logged in.
- **js_relogin** – the name of the JS function which will be sent to the user if his/her ID is found but the IP address or session_id do not match the data transfered by the application in the login command. The module will add a name of the parameter which describes the reason of the failure to the name of the function. The names are: „sessid" or „ipaddr".
- **js_redirect** – the name of the JS function which will be sent to the user if there already is an open connection from the given user to the module in the module. If you open two connections to the same address, the browser will be blocked. In order to fix this problem you can configure several addresses for an application. www0, www1, … www5. The module adds the no. of the free slot as a parameter. The application should switch to a different address after receiving this message.
- **merge_messages** – merges messages. Thanks to this when two messages are queued to one window, they will be delivered as one message.
- **send_redirect**s – sends the js_redirect code. When this option is on the browser will receive the js_maxwin_code when a second connection from a given user to the same address is detected.

Lighttpd configuration

The following settings can be added to the Lighttpd server configuration. The first option is important because the connections to the module can remain unused for a long time. The two other options are important only for heavily loaded installations.

```
server.max-write-idle   = 86400
server.event-handler    = "linux-sysepoll"
server.max-fds          = 10000
```

User's commands

wait

Browser waits for a message from the server (chunked mode)

Parameters

- **user_id** - Numerical user's id
- **win_id** - Random integer number which identifies an open browser window or tab.
- **types** - Event categories handled by a given window. e.g. chat, usr, gen, …

- **sec** – Timestamp from which the window should start to receive messages.
- **usec** – Timestamp microseconds.

stream

Browser waits for messages from the server (stream mode). The parameters are the same as for the wait command. There is an additional parameter:

- **win_st** – When set to 1, fdajax will send messages with script tags added.

chat_sig

Allows to send a direct signal or message to another user's browser. The message is marshalled directly to another logged user skipping the application server (PHP or RoR). It's very useful to send messages like: the user has started to write a message.

Parameters:

- **user_id** - Numerical user's id
- **peer_id** - Recipient's user id
- **types** - Event category e.g. chat, usr, gen, … and optional numerical object ID
- **mes** - JavaScript function parameter list which will be marshalled to a defined function in the recipient's browser. FdAjax will add sender's id as a first parameter of the above list. Only two types of parameters are allowed: string and number. This is for security reasons. The module will check parameters syntax and return syntax error in case of problems. If you want to implement a RPC call for JavaScript you have to give the name of the function as a parameter in the string and then check that parameter before calling a given function. I do not recommend calling eval on a string from a different user.
  Examples:

  ```
  250, "hello world"
  "hello world", 3.1415 ,"fdajax"
  ```

  If the sender has an ID the value of which is 1000, the receiver will get the following message:

  ```
  js_peer_func(1000, 250, "hello world");
  js_peer_func(1000, "hello world", 3.1415, "fdajax");
  ```

  It is important for the JS code to check if the user with the ID 1000 can send such a message e.g. if he/she belongs to the members of the chat with the ID 250. The list of the chat members should be set while generating the page or transfered by the admin command push.

cancel

Closing an open connection. It is required due to anti-virus programs, which monitor the connections on the users' computers and do not close the connection to the web server even if

the browser has already closed it. When you close a page you have to inform the module about it, so that it can close the connection to that window.

Parameters

- **user_id -** Numerical user's id
- **win_id** – window id to close

Admin commands

login

Logs the user to the module, from that time on the user with a given ID can establish a connection to the module.
Parameters

- **user_id -** Numerical user's id
- **ip_addr** – user's IP address
- **sess_id** – user's session ID

logout

Logs the user out of the module

Parameters

- **user_id** - Numerical user's id

check

Checks if the user has open windows and if not logs the user out.

Parameters

- **user_id -** Numerical user's id

push

Sends a message to the user (ends the user's connection in case of chunked mode).

Parameters

- **peer_id** – Recipient's user id
- **mes** – Message to be sent to the other user.
- **types** – event's name and optionally object's ID

add_chat

If the users are logged in it enables to send chatsig among users.

Parameters

- **user_id** – First user numerical id
- **peer_id** - Second user numerical id

rm_chat

If the users are logged in it disables the possibility to send chatsig among users.

Parameters

- **user_id** – First user numerical id
- **peer_id** - Second user numerical id

Events

Events are useful for bigger applications in which the user can open several browser windows with a given application. Thanks to events, the messages are sent to windows, that are waiting for those messages. In other words events specify the way the messages should be routed. Currently the names of the events are defined permanently and cannot be changed. They are:

```
chat gen mes ord ccd itm lne usr
```

(and events used on Refwell). Optionally after the event's name you can give the objects' numerical ID. This allows even better control e.g. you can open several windows with various chats and each window will only receive messages related to that given chat.

Example for windows which have one chat:

```
chat,250
```

Example for a window which has 3 different chats:

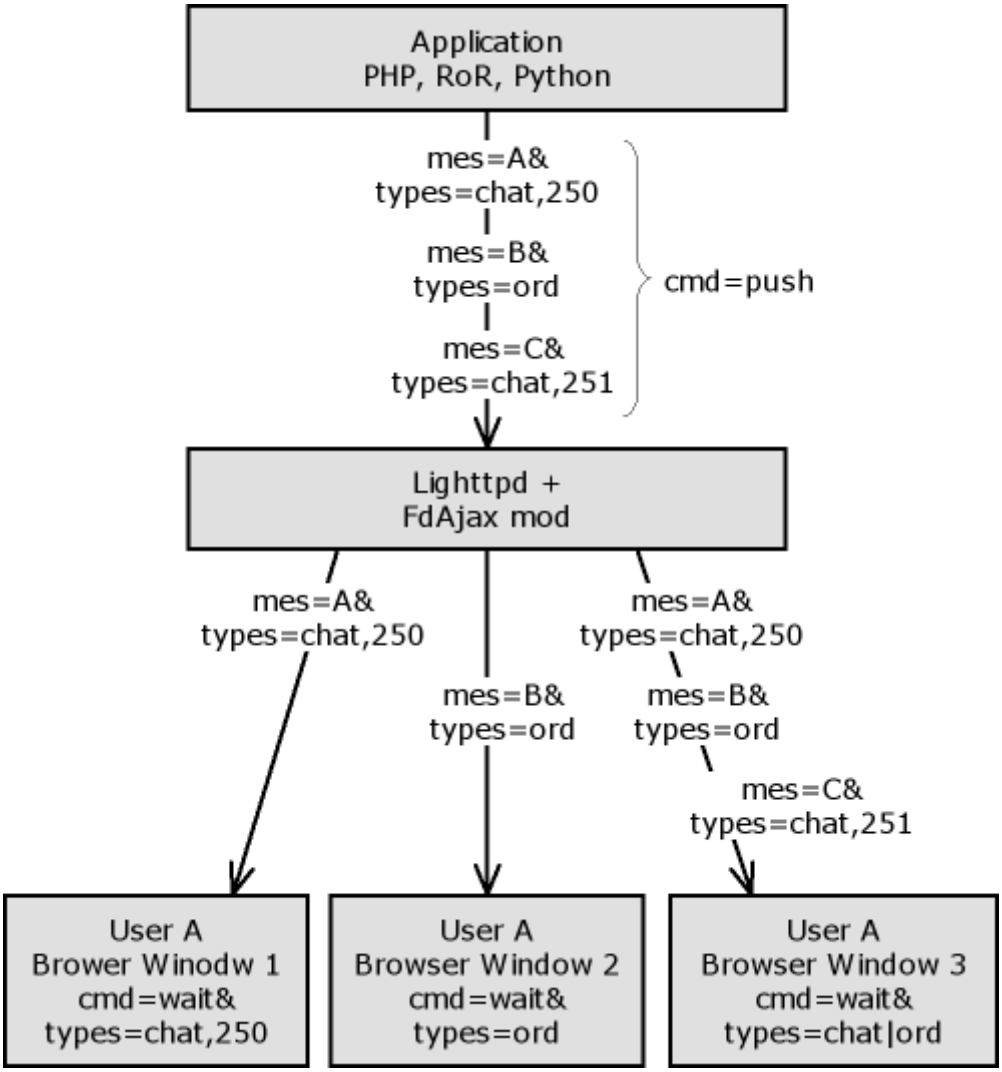
```
chat,250,251,252
```

The names of the events can be joined e.g. an application window can have an open chat and an list of orders. Example:

```
chat,250|ord
```

While sending a message (push i chatsig) types can only have one event name and one optional object ID.

At the beginning to make it simple the application can use only one event with the name "chat". Various events should be used when the traffic in your application is bigger.



Special event type “unw”

There is one more special event “unw”. A bigger and integrated application can have several types of objects, which have to be refreshed in real-time e.g. a user has an open window with the order’s list and at the same time someone sends him/her an important chat message. Normally this message would not be transferred to the window with the orders. FdAjax, however, allows to add a special “unw” event to the list of handled events “ord|unw”. Thanks to this the window will receive a message “chat” but only if the user doesn’t currently have an open window, which receives chat messages. To mark this situation, a “unw” prefix is added to the message.

This is very useful when the messages are JS code. It allows to handle these kind of messages in a different way. For example the function

```
chat.display_message(...)
```

displays a chat message. If you don’t have a chat in the currently open window, the chat message should be displayed in a different way. That is you have to define the function

```
unw.chat.display_message(...)
```

for the remaining windows, which will display the message as e.g. a popup box.

Redirects

There are ways to get around the default limit of open connections from the browser to one www server. The limit is 2 connections. If you configure the addresses

```
www.example.com, www0.example.com, www1.example.com ... www5.example.com
```

so that they point to the same IP address- lighttpd server with fdajax mod, you can cheat the browser and change the address.
If you use redirects you should set a domain path for session cookie e.g. for example.com so that the browser sends session cookie when communicating with all the addresses www, www0 ….

The module checks to which www address a given command has been sent. It recognizes the numbers that are added to the first part of the address, they are called slots. Upon receiving a wait or stream command to an occupied slot, the module sends js_redirect as a reply together with the parameter of a free slot. In such a case the application should switch to a different address, using the number of the slot it received as a reply.

Problems

In case of problems please check the file with the messages about lighttpd server errors. These messages are rather difficult to understand for people who do not know the module's code. However, they might help. It's better to get professional help on Anslive.

FdAjax version 0.7

March 7th, 2007 by Grzegorz Daniluk

This is the new official logo for FdAjax module.

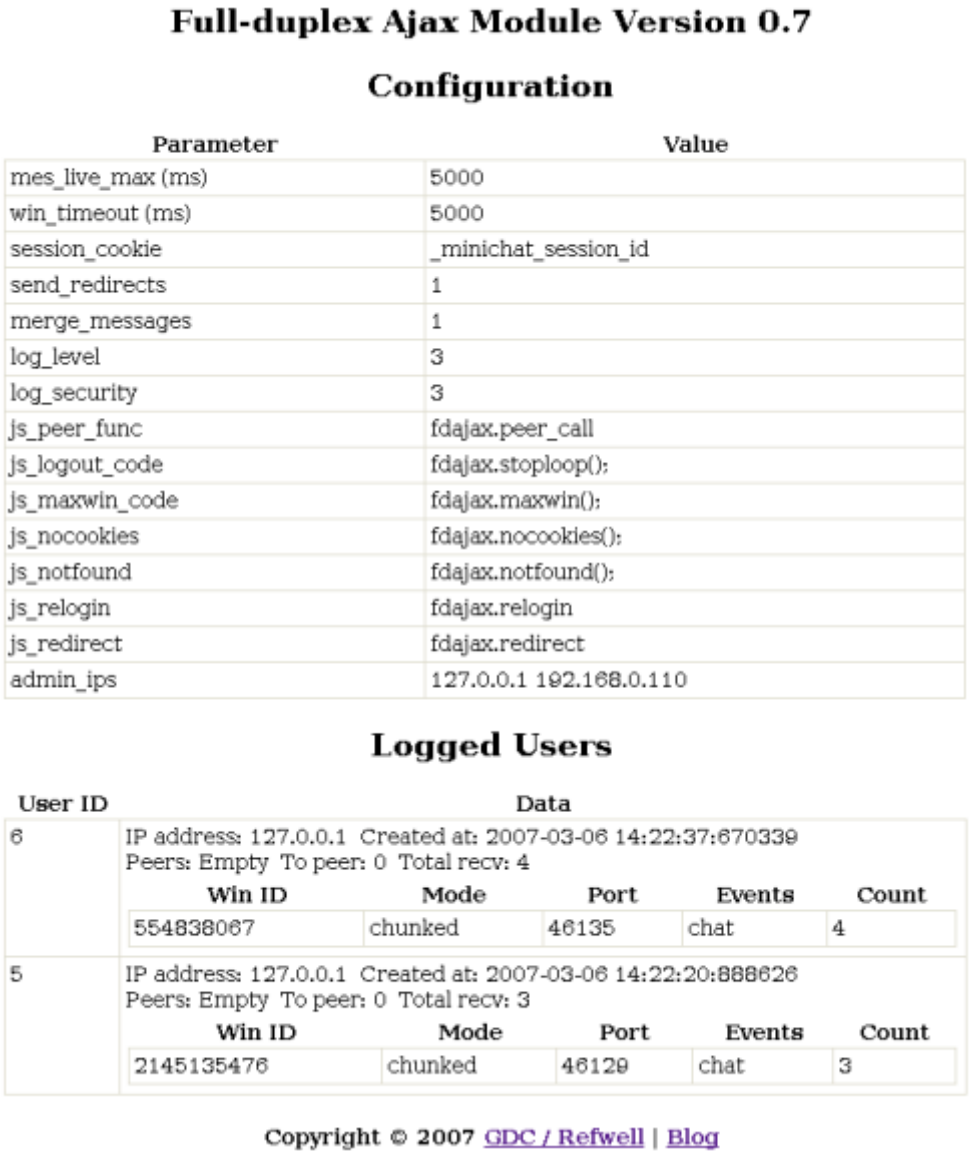
Some quick remarks about the changes in version 0.7:

- The moment of sending the js_maxwin_code has changed. In the previous versions the JavaScript code was sent to the last opened window. This caused problems when one had an anti-virus program installed, because it kept all connections to the module open and did not recognize that the browser has already closed the connection. In version 0.7 the code is sent to the oldest window, which is a better idea.
- The module can be now compiled on Mac as well. In the previous versions the module used an option of the function getsockopt, which is not available for some Unix systems. From now on for other operating systems the system function select () is used to check if the socket is open.

- A new admin command “info” has been added. It shows the current configuration of the module and a list of logged in users. For each user there is information about his/her open connections to the module displayed. This is very useful while debugging applications and helps to better understand the way the module works.

  ```
  http://localhost:3000/live.fdajax?cmd=info
  ```

  And the screen-shot:



Full-duplex Ajax Module Version 0.7

Configuration

Parameter	Value
mes_live_max (ms)	5000
win_timeout (ms)	5000
session_cookie	_minichat_session_id
send_redirects	1
merge_messages	1
log_level	3
log_security	3
js_peer_func	fdajax.peer_call
js_logout_code	fdajax.stoploop();
js_maxwin_code	fdajax.maxwin();
js_nocookies	fdajax.nocookies();
js_notfound	fdajax.notfound();
js_relogin	fdajax.relogin
js_redirect	fdajax.redirect
admin_ips	127.0.0.1 192.168.0.110

Logged Users

User ID	Data
6	IP address: 127.0.0.1 Created at: 2007-03-06 14:22:37:670339 Peers: Empty To peer: 0 Total recv: 4

Win ID	Mode	Port	Events	Count
554838067	chunked	46135	chat	4

User ID	Data
5	IP address: 127.0.0.1 Created at: 2007-03-06 14:22:20:888626 Peers: Empty To peer: 0 Total recv: 3

Win ID	Mode	Port	Events	Count
2145135476	chunked	46129	chat	3

Copyright © 2007 GDC / Refwell | Blog

- Anslive widget is also displayed on the info page. Now you don’t have to log in to Refwell to download FdAjax source code. And the sponsor of the module is happy:)

FdAjax and Mini-chat

February 2nd, 2007 by Grzegorz Daniluk

In my last post I promised to write more about FdAjax mod. First of all I would like to write about the changes that were introduced and the problems that were fixed in version 0.6.

Anti-virus programs

Some Refwell users had an annoying problem. When they switched the pages it turned out that the browser on the new page switches to a new address e.g. www1, www2 etc. After many hours spent on trying to find the problem I discovered, that the problem lies in anti-virus programs, which monitor the whole traffic from the browser to the web. It was quite a big surprise because version 0.5 was tested with Squid proxy server and everything worked well. The anti-virus program works similarly to the Proxy server. The difference is that it turned out that it is badly written.

The html page with the JavaScript code opens the connection to the FdAjax server using XmlHttpRequest. The XHR connection hangs open and is connected to the FdAjax module. When the user leaves the page, Firefox automatically closes this connection. When it comes to IE you have to handle the unload event and execute the abort() method on an open XHR object. FdAjax will keep the connection open until the XHR object is closed by the browser.

Unfortunately the Anti-virus program waits until the web server closes the connection. It does not pay attention to the browser. How should FdAjax distinguish this situation from the one where a user just opens a new browser window with our application? I did not find an easy answer for that. But I managed to find a work-around. During the unload event there is one more XHR which informs FdAjax that the connection with a given win_id was closed. Thanks to this FdAjax mod knows which connection has to be closed. This solves the problem but does not satisfy me. If you can think of a better solution please leave your comment.

There is one more problem. Opera does not execute the unload event when the user presses the “Back” or “Forward” buttons. I can’t test it on Safari yet so please tell me if it works well there.

Changes

1. There is a new ’stream’ command in the 0.6 version. The difference between the ‘wait’ and the ’stream’ command is that a connection opened using the ’stream’ command is not closed after the data has been sent to the user. Therefore it is just normal http streaming. Nothing changes as far as the application is concerned. There is still only one ‘push’ command to send data to a given user. FdAjax decides whether to send the data and close the connection or just to send the data.

2. The ‘push’ command was called ’stop’ in the 0.5 version. However, the word push better describes what this command actually does in the 0.6 version.

3. The ’stream’ and ‘wait’ commands accept new parameters now, that is ’sec’ and ‘usec’. ‘Sec’ stands for seconds and ‘usec’ for microseconds. When the page is generated the time (gettimeofday()) is saved in JavaScript variables. Next, during the first XHR to the FdAjax mod they are given as parameters to the ‘wait’ and ’stream’ commands. The time from generating the page to the first connection to FdAjax can last even a few seconds. If during this time there is an event which needs to be sent to the user it will be properly sent to the browser thanks to the queuing of messages and the ’sec’ and ‘usec’ parameters. FdAjax checks the time when it received data that is to be sent to a given user using the ‘push’

command and compares this time with the timestamp received from ’sec’ and ‘usec’ from the page. Thanks to this the risk of loosing a message has significantly decreased.

How to use FdAjax module - Hello world example

Let’s say that a user has logged in to your application. You have his session id and his numerical id in your database. Session id is generated automatically by RoR and PHP or Python. Now you have to send these pieces of information to FdAjax mod. It is done using the ordinary HTTP GET request. An example for RoR:

```
cmd = "cmd=login&user_id=" + user_id + "&sess_id=" + session.session_id +
        "&ip_addr=" + request.remote_ip
Net::HTTP.get_response('127.0.0.1', '/live.fdajax?' + cmd, 80)
```

After executing the above command JavaScript can connect to FdAjax. An example that uses the Prototype library.

```
var fdajax = {
   user_id: 0,
   win_id: 0,
   init: function(user_id) {
      this.user_id = user_id;
      this.win_id = Math.floor(Math.random() * 2147483646 + 1);
      Event.observe(window, "load", fdajax.send_request);
   },
   send_request: function() {
      var opt = {
          onSuccess: function(resp) {
              try { eval(resp.responseText); } catch (e) {}
              setTimeout("fdajax.send_request();", 20);
          },
          onFailure: function(req) {
              setTimeout("fdajax.send_request();", 10000);
          },
          method: 'get',
          parameters: "cmd=wait&user_id=" + fdajax.user_id + "&win_id=" +
                          fdajax.win_id + "&types=chat"
      };
      new Ajax.Request('/live.fdajax', opt);
   }
}
```

Now your application can send a message to that user.

```
mes = "alert('Hello world!');"
cmd = "cmd=push&types=chat&peer_id=" + user_id + "&mes=" + CGI::escape(mes)
Net::HTTP.get_response('127.0.0.1', '/live.fdajax?' + cmd, 80)
```

The message can be any JavaScript code. It can contain a chat message from another user. Where to get that message from? Just an ordinary XHR or HTTP POST call – so traditional Ajax. After a closed session you can log the user out.

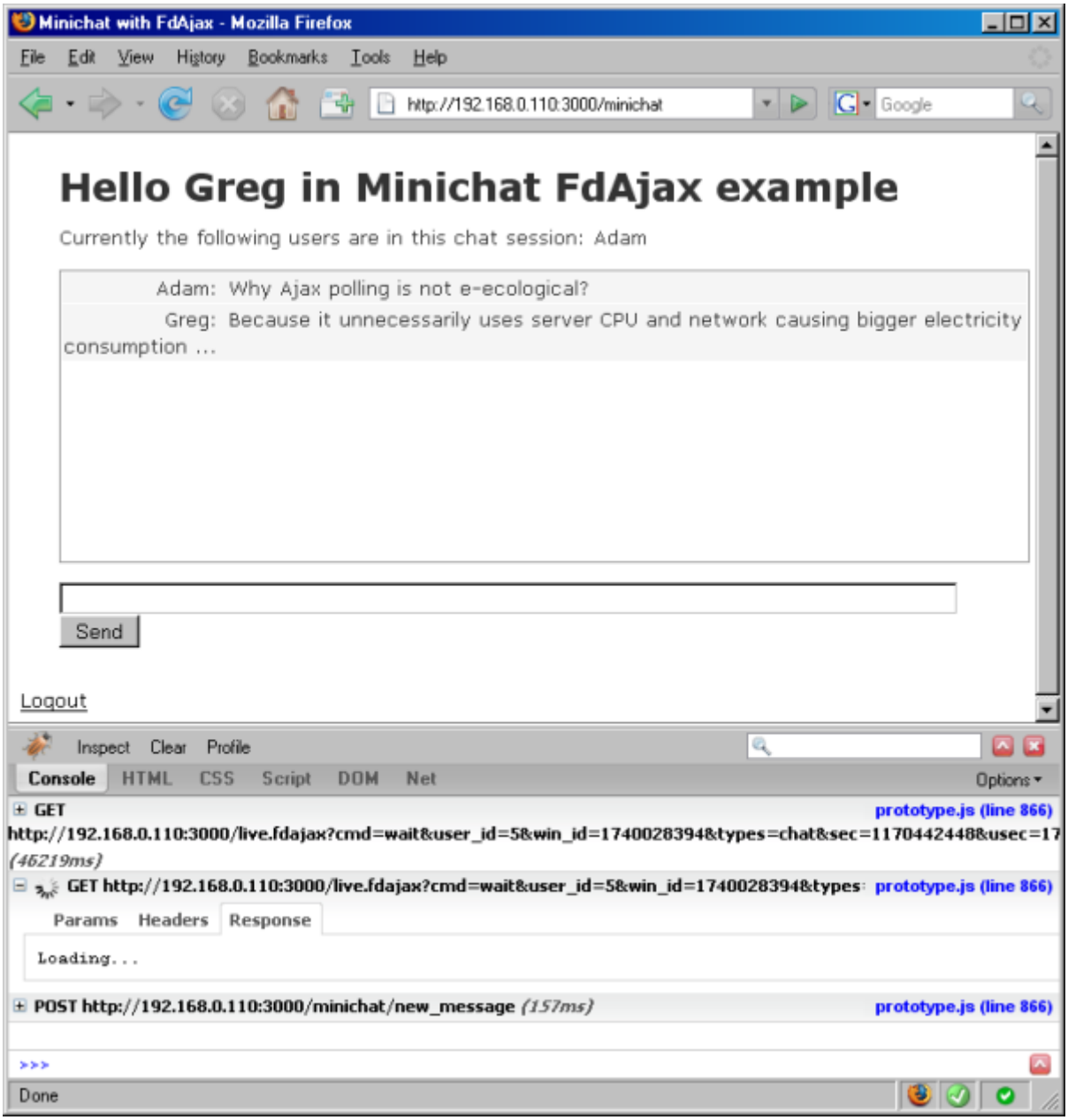
```
cmd = "cmd=logout&user_id=" + user_id
Net::HTTP.get_response('127.0.0.1', '/live.fdajax?' + cmd, 80)
```

That’s it. If you managed to compile the module and run the example write your comment below and also tell me what should be improved in the next version of FdAjax.

Minichat application

As an example we have included a mini chat application written in Ruby on Rails (v.1.2). The most important files are minichat_controller.rb and application.js. To run this application:

- Compile and install mod_fdajax for Lighttpd
- In the lighttpd.conf check the fdajax.admin_ips field and add the IP address of the computer which runs the RoR application.
- Create minichat_development MySql database. Run “rake db:migrate”
- Go to the directory with the application and run the web server “./script/server lighttpd”
- Open the browser with an address e.g. http://localhost:3000
- Add a few users
- Log in to the mini chat. That’s it. Open IE and Firefox if you work on one computer.



digg this | add to del.icio.us | Posted in fdajax | 8 Comments »

Full-Duplex Ajax module for Lighttpd

January 25th, 2007 by Grzegorz Daniluk

Full-Duplex Ajax (FdAjax) is an extension of the Ajax technology. It allows to initiate communication from the HTTP server to the browser and therefore enables to easily create efficient solutions like web chats. It is the server that sends new messages to the browser when they appear. Normally the browser (JavaScript code) has to periodically check if there are new messages and send queries to the server. This is very expansive and causes delays.

FdAjax is used on Refwell. Thanks to FdAjax, in most cases, you only have to click twice to get your help on Anslive. In order to find something on Google you have to click once. And

in order to perform a much more complicated transaction between two users on Anslive you only have to click twice. Check it out. Watch our Anslive video tour.

FdAjax is implemented as a module to the great Lighttpd web server. No additional components are needed. Therefore the **configuration is easy and unified**. E.g. you have to configure the SSL communication and the access control in the configuration file only once, it will also apply to the communication with the FdAjax module.

The module can be used with different frameworks e.g. Ruby On Rails, PHP, Python. Upon completing the Lighttpd mod_proxy, FdAjax will also be easily integrated even with Java applications.

At present the module is only available for the Lighttpd server. This server has been chosen due to its inner architecture (event loop). Thanks to this each client that is connected to a server via FdAjax takes up less than **1kB of memory and one file descriptor**. There's no need to create new processes for each client, as it was done before.

FdAjax is similar to Comet. The main difference is that Comet is an additional software installed next to HTTP server. FdAjax is an extension of Ajax therefore the prefix well describes the technology. The present web 2.0 applications can be described as Half-Duplex Ajax.

Why open source?

You probably wonder why we decided to release this innovative software as an open source. We want to show that **open source projects have a new way to bring income**. Anslive enables to organize technical support for users and that way open source project contributors can make money. Up till now only big projects such as MySQL were able to organize commercial technical support. Now thanks to Anslive it can also be organized by individuals, even without the need of answering users' questions by themselves. You only have to embed our widget on your project's site.

The FdAjax project will be supported and developed thanks to commercial technical support. Are you an open source project contributor? Embed our widget on you site, let's check if thanks to Anslive you will be able to earn money for developing your project. Serwhizz will soon make similar tools for open source project contributors available.

Installation

At present the 0.5 version of FdAjax is available. It has to manually compiled and installed. In the future it will be simplified. It will also depend on the popularity of the module.

This module is compiled and tested on Ubuntu Linux 6.06 and Lighttpd 1.4.13. It should also work well with older versions of Lighttod 1.4.x as well as other Linux distributions.

Configuration

The present version gives the possibility to set parameters on the server level. It is sufficient at this stage of the project. E.g. Lighttpd configuration:

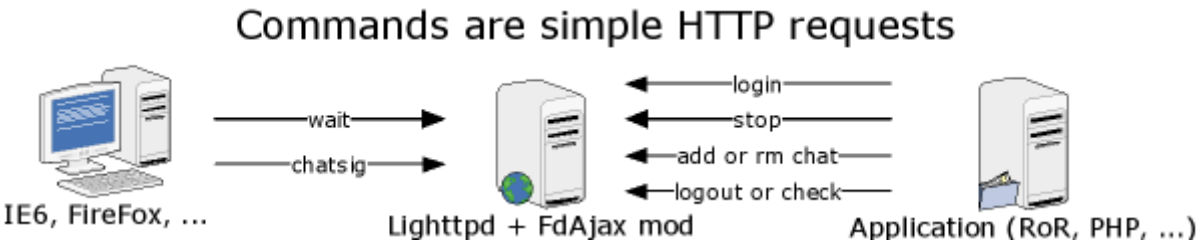
```
server.modules = ( "mod_access", "mod_rewrite", "mod_accesslog",
                   "mod_fdajax", "mod_fastcgi" )

fdajax.ext              = "fdajax"
fdajax.admin_ips        = ( "127.0.0.1" )
fdajax.mes_live_max     = 1000
fdajax.win_timeout      = 5000
fdajax.log_level        = 3
fdajax.log_security     = 3
fdajax.js_peer_func     = "gdclive.peer_call"
fdajax.js_logout_code   = "gdclive.stoploop();"
fdajax.js_maxwin_code   = "gdclive.maxwin();"
fdajax.js_nocookies     = "gdclive.nocookies();"
fdajax.js_notfound      = "gdclive.notfound();"
fdajax.js_relogin       = "gdclive.relogin"
fdajax.js_redirect      = "gdclive.redirect"
fdajax.session_cookie   = "_session_id"
fdajax.merge_messages   = "enable"
```

Commands

There are two groups of commands: user commands and admin commands. The admin commands can only be called by your application, which decides who can connect to FdAjax and sends the information necessary to identify the user (session id) to the module.

The following diagram shows how to use FdAjax commands in your application.



For more details on commands please go to our download page. Please sign up to access this page.

Event dispatcher

Due to the fact that the user can open several windows with our application, FdAjax has the ability to **route messages to particular windows**. In the *stop* command you have to specify for which type of events a given window waits. The basic event is chat. A window which does not have a given type of event will not receive a given message.

Additionally to the type of event we can include a list of object IDs that are related to it. That means you can open two windows, each with a chat event but with a different chat ID. Thanks to this a user may open two chat windows at once and the messages will be routed to appropriate windows.

In the application code you only have to give the chat ID in the *stop* command. In the JavaScript code you have to give the randomly generated win_id and chat ID to the *wait* command. The rest is done by FdAjax.

If two windows have the same event and object ID, **the message will be duplicated** and FdAjax will route it to these two windows.

An example of a window with one chat with the ID 10 :

```
http://www.refwell.com/live.fdajax?cmd=wait&user_id=5&win_id=311645485&types=chat,10
```

Deployment

FdAjax 0.5 can be installed on a PHP or RoR dedicated server. The most important things you have to do:

- define the admin IP addresses in FdAjax configuration
- configure a firewall so that the admin IP addresses will be protected by it
- if you allow users to open more than one connection and redirect them to addresses www1, www2,…you have to set the session id cookies for the domain. The addresses www1, www2 should also indicate the same Lighttpd server.

FdAjax module should be well tested and ready for production use by the end of February.

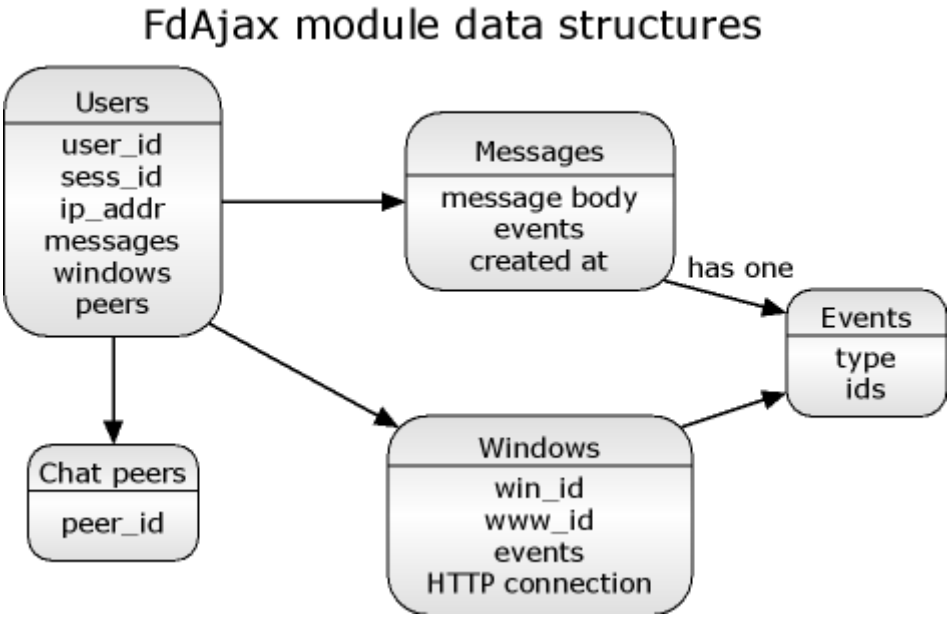
Technical details

The way the module works is quite easy. FdAjax intercepts all HTTP requests to live.fdajax URL. In case of *wait* command, the connection is open until it receives a *stop* command with a message for a given user (chunked mode). Thanks to this solution one can **avoid problems with Proxy servers** which will not forward the answer to the browser until the HTTP server closes the connection. When a FdAjax window connection is closed, messages to that window will be queued and sent immediately as one package when the window calls the next *wait* command.

If the user doesn't have problems with the connection they can use other commands which enable streaming and send messages without the need to end the FdAjax connection, which is more efficient. In such a case you should use the hidden IFRAME which will load the JavaScript code that needs to be executed.

You have to remember that the browser will allow to create only two connections to one HTTP server. If you open a third one, the browser will stop loading content from your site and will wait until other connections will be closed. FdAjax checks if a given user already has a connection. If that's the case they will receive a JavaScript message asking to **switch the browser to a different address**.

The following diagram shows the data stored by FdAjax for every user. This will give you a rough overview of how the module works.



Road Map

Short term

- Version for lighttpd 1.5.0
- Streaming mode
- Public groups – stream content to people who are not logged in.
- Detect browser behind Proxy server and switch between Streaming and Chunked modes.

Long term

- Scaling to several web servers (Lighttpd instances)
- Linux kernel patch- a new sockopt which allows a socket to receive a connection error if the socket isn't in a readable or writable state.

Download

We would like to know who uses our module. The source code download page is available only for users registered on Refwell. In order to sign up you only have to give your email address.

In our next post we will show an exemplary mini-chat application which uses FdAjax. All suggestions are welcome.