JQuery Terminal Emulator Plugin - Api Reference

Originalartikel

Backup

<html> <head><meta charset=„utf-8“/><title>JQuery Terminal Emulator Plugin - Api Reference</title><meta name=„author“ content=„Jakub Jankiewicz - jcubic@onet.pl“/><meta name=„Description“ content=„API refernce for JQuery Terminal Emulator - list of all methods and options.“/><meta name=„keywords“ content=„jquery,terminal,interpreter,console,bash,history,authentication,ajax,server,client“/></head ><body class=„readabilityBody“ readability=„229“>

```
  <header id="main" readability="12">
  <a href="http://terminal.jcubic.pl/" readability="21"><pre id="sig"
readability="33">
```

```
   __ _____                  _________                          __
  / // _  /__ __ _____ ___ __ _/__  ___/__ ___ _____ __ __  __ ___  / /
```

/_ _ / / _ _ \/ _ \/ / / _ / / / / _ / / / / /\ / / \ _ __/ _ _ / / __/ / _/ / __/ / _/ \\ _/ \/ // 1.4.3   / _ / _ / _ _ / / / / _ _ \/ _ \/ / / / / _ / / / / /\ / / \ _ \ \ /__/ / _/ / \/ 1.4.3   / _ / _ / _ / / / _ _ / / / / / / _ / / / / _ \ / __/ / _/ / _/ \/ 1.4.3 </pre><!– for FB bigger then gihub ribbon –> <pre class=„separator“>——</pre> </header> <section readability=„94“><article><header id=„api“/>
Interpreter Options Terminal Object Instance Methods Terminal Utilites Command Line Additional terminal controls Changing Colors Translation Error Handling Style Keyboard events Authentication Thrid party code and additional plugins </article><article><!– black wide –><ins class=„adsbygoogle“ data-ad-client=„ca-pub-6153701670678834“ data-ad-slot=„5835458303“/> </article><article id=„interpreter“ readability=„74“><header><h2>Interpreter</h2></header> To create

terminal you must pass interpreter function (as first argument) which will be called when you type enter. Function has two argumentss command that user type in terminal and terminal instance. Optionally you can pass string as first argument, in this case interpreter function will be created for you using passed string as <abbr title=„Uniform Resource Identifier“>URI</abbr> (path to file) of JSON-RPC service (it's ajax so must be on the same server). <pre class=„javascript“> $('#some_id').terminal(function(command) { if (command == 'test') { this.echo(„you just typed 'test'“); } else { this.echo('unknown command'); } }, { prompt: '>', name: 'test' }); </pre> You can pass object as first argument - the methods will be invoked by commands typed by a user. In those methods this will point to terminal object. <pre class=„javascript“> $('#some_id').terminal({ echo: function(arg1) { this.echo(arg1); }, rpc: 'some_file.php', calc: { add: function(a, b) { this.echo(a+b); }, sub: function(a, b) { this.echo(a-b); } } }, { prompt: '>', greeting: false }); </pre> This code will create two command echo that will print first argument and add that will add two integers. From version 0.8.0 you can also use array with strings, objects and functions. You can use multiple number of objects and strings and one function (that will be called last if no other commands found). If you have ignoreSystemDescribe function enabled you will be able to use only one string (JSON-RPC url). If you have completion enabled then your commands will be that from objects and JSON-RPC that have system.describe <pre class=„javascript“> $('#some_id').terminal([„rpc.php“, { „mysql“: function() { this.push(function(command, term) { $.jrpc(„rpc.php“, 'mysql', [command], function(json) { term.echo(json.result); }); }, { prompt: 'mysql> '); } }], { prompt: '>', greeting: false }); </pre> In previous example mysql will be exception, even that rpc have that method it will not call it but create new interpreter. Terminal will always process numbers if processArguments is set to true (by default). </article><article id=„options“ readability=„21“><header><h2>Options</h2></header> This is list of options (for second argument): <ul readability=„2“><li id=„history“>history [bool] — if false will not store your commands. <li id=„prompt“>prompt [string|function] — default is “> ” you can set it to string or function with one parameter which is callback that must be called with string for your prompt (you can use ajax call to get prompt from the server). You can use the same formatting as in echo. <li id=„name“>name [string] — name is used if you want to distinguish two or more terminals on one page or on one server. (if name them differently they will have different history and authentication). <li id=„greetings“>greetings [string|function(callback)] — default is set to JQuery Terminal Singnature. You can set it to string or function (like prompt) with callback argument which must be called with your string. <li id=„processarguments“>processArguments [bool | function] — if set to true it will process arguments when using an object (replace regex with real regex object number with numbers and process escape characters in double quoted strings - like \x1b \033 will be Escape for ANSI codes) - default true. If you pass function you can parse command line by yourself - it have one argument with string without name of the function and you need to return an array. <li id=„outputlimit“>outputLimit [number] — if non negative it will limit the printing lines on terminal. If set to 0 it will print only lines that fit on one page (it will not create scrollbar if it's enabled). Default -1 which disable the function. <li id=„linksnoreferer“>linksNoReferer [bool] — if set to true it will add rel=„noreferer“ to all links crated by terminal (default false). <li id=„exit“>exit [bool] — if this option is set to false it don't use CTRL+D to exit from terminal and don't include “exit” command, default is true. <li id=„clear“>clear

[bool] — if this option is set to false it don't include “clear” command, default is true. <li id=„login“ readability=„4“>login [function(user, password, callback)|string] — login can be function, string or boolean. Function must have 3 arguments login password and callback which must be called with token (if login and password match) or falsy value (if authentication fail). If interpreter is string with valid URI JSON-RPC service you can set login option to true (it will use login remote method) or name of RPC method. this in login function is terminal object. <pre class=„javascript“> function(user, password, callback) { if (user == 'demo' && password == 'secret') { callback('AUTHENTICATION TOKEN'); } else { callback(null); } }</pre> But you need to know that everybody can look at your javascript source code so it's better to call server using AJAX here and call callback on response. If callback receive truthy value, you can get that value using token method so you can pass when calling the server (and server then can identify that token). <li id=„tabcompletion“><strike>tabcompletion [bool] — enable tab completion when you pass object as first argument. Default is false (tabulation key default insert tabulation character).</strike> removed in version 0.8.0. <li id=„completion“>completion [function (string, callback)|array|boolean] — function with a callback that need to be executed with list of commands for tab completion (you need to pass array of commands to callback function), from version 0.8.0 you can also use true (it will take completion from object or RPC, if it have system.describe, as interpreter) or array if you know what your commands are and don't need to call ajax to get them. From version 1.0.0 it no longer have terminal as parameter, terminal is now in <code>this</code> context. <li id=„enabled“>enabled [bool] — default is true, if you want disable terminal you can set it to false. This is usefull if you want to hide terminal and enable on some action (If Terminal is enabled it intercept keyboard). <li id=„checkarity“>checkArity [bool] — if set to true (by default) it will check number of arguments in functions and in JSON-RPC if service return system.describe (only 1.1 draft say that it must return it, new Spec 2.0 don't say anything about it, json-rpc used by examples return system.describe). <li id=„memory“>memory [bool] — if set to true it will not use localStorage nor Cookies and save everything in memory only, default false. <li id=„onInit“>onInit [function(terminal)] — callback function called after initialization (if there is login function it will be called after authentication). <li id=„onRPCError“>onRPCError [function(error)] — callback function that will be called instead of built in RPC error. (this in that function is terminal object). <li id=„onExit“>onExit [function(terminal)] — callback function called when you logout. <li id=„onClear“>onClear [function(terminal)] — callback function called when clear command is executed. <li id=„onBlur“>onBlur [function(terminal)] — callback function called when terminal get out of focus. If you return false in this callback function the terminal will not get out of focus. <li id=„onResize“>onResize [function(terminal)] — callback function called when terminal get resized. <li id=„onFocus“>onFocus [function(terminal)] — callback function called when terminal get focus. <li id=„onTerminalChange“>onTerminalChange [function(terminal)] — callback function called when you switch to next terminal. <li id=„onBeforeLogin“>onBeforeLogin [function(terminal)] — callback function called called before login. <li id=„processRPCResponse“>processRPCResponse [function(object)] — callback function that will be use with any result returned by JSON-RPC. So you can create better handler. <li id=„onCommandChange“>onCommandChange [function(command, terminal)] — event fired when command line is changed. <li id=„exceptionHandler“>exceptionHandler [function(exception)] — callback that will be executed instead of default print exception on terminal. <li

id=„historyFilter“>historyFilter [function(command)] — if you return false in this function command will not be added into history. <li id=„historySize“>historySize [number] — size of the history (default 60) if you pass falsy value it will be not restricted. <li id=„historyState“>historyState [boolean] — if set to true terminal will record all commands in url hash. <li id=„keypress“>keypress [function(event, terminal)] — function called on keypress event if you return false it will not execute default actions (keypress event is called when you type printable characters). <li id=„keydown“>keydown [function(event, terminal)] — function called on keydown event if you return false it will not execute default actions (keydown event is use for the shortcuts). <li id=„convertLinks“>convertLinks [boolean] — if set to true it will convert urls to a tags, it do that by default. <li id=„scrollOnEcho“>scrollOnEcho [boolean] — indicate if terminals should scroll to bottom on echo or flush. If set to false it will scroll to bottom if terminal was at the bottom, it use <code>is_bottom</code> method. <li id=„linksNoReferrer“>linksNoReferrer [boolean] — if set to true it will set noreferrer on links, default set to false. <li id=„maskChar“>maskChar [boolean|string] — default mask character by default it's `' (if set to true), used when you use <code>set_mask(true)</code>. <li id=„wrap“>wrap [boolean] — if set to false terminal will not wrap long lines (it can be overwritten by echo option), default true <li id=„execHash“>execHash [boolean] — if set to true it will execute commands from url hash, the hash need to have a form of JSON array that look like this <code>#0,1,"command"],[0,2,"command2"</code> first number is index of terminal on a page second is index of command for terminal. (0 is initial state of the terminal so first command have index of 1). Set to false by default. <li id=„onAfterCommand“>onAfterCommand [function(command)] — callback function executed after the command. <li id=„onBeforeLogout“>onBeforeLogout [function] — function executed before logout from main interpreter, if function return false terminal will not logout. <li id=„onAfterLogout“>onAfterLogout [function] — function executed after logout from the terminal if there was a login. <li id=„onAjaxError“>onAjaxError [function(xhr, status, error)] — function executed on JSON-RPC ajax error. (this in this function is terminal object). <li id=„onBeforeCommand“>onBeforeCommand [function(command)] — function executed before command. If function return false the command will not be executed. <li id=„onCommandNotFound“>onCommandNotFound [function(command, terminal)] — function executed if there are no command with that name, by default terminal display error message, it will not work if you use function as interpreter. <li id=„onPause“>onPause [function] — function executed when you call pause() or return a promise from a command. <li id=„onResume“>onResume [function] — function executed when you call resume() or when promise returned in command is resolved. <li id=„onPop“>onPop [function] — function exectuted when each time pop is called, CTRL+D also call pop so the function is triggered, this method is also executed when you have login and you exit from main interpreter. <li id=„onPush“>onPush [function] — function executed when you push new interpreter on the stack of interpreters. <li id=„scrollBottomOffset“>scrollBottomOffset number — indicate offset from bottom in which terminal will consider at bottom of the terminal. Used in <code>is_bottom</code> method. <li*

id=„importHistory“>importHistory [boolean] — if the options is to true it will import history in import_view exported by export_view, default set to false. <li id=„request“>request [function(jxhr, terminal, request)] — callback function called before senidng JSON-RPC request to the server (it's also called on system.describe), you can modify request or jQuery XHR object, see CSRF Example. Added in version 1.0.0. <li id=„response“>response [function(jxhr, terminal, response)] — callback function called after JSON-RPC response (it's also called on system.describe), you can modify response before it's processed by jQuery Terminal, also you can call methods on jQuery XHR object. see CSRF Example. Added in version 1.0.0. <li id=„wordAutocomplete>“>wordAutocomplete [boolean] — if set to false it will autocomplete whole command before cursor, default set to true to autocomplete only word. <li id=„terminal_keymap“>keymap — object where keys are uppercase shortcuts like CTRL+ALT++C the order of modifiers is CTRL, META, SHIFT, ALT. Added in version 1.0.0. <li id=„echo_command“>echoCommand [boolean] — if set to false terminal will not echo command you enter with prompt, default true. <li id=„num_chars_rows“>numChars/numRows [number] — fixed number of rows and cols, created mainly for testing from node. <li id=„on_export_import“>onExport/onImport [function] — callback functions executed when calling export/import. You can add properties additial state to be saved and restored. in export you need to return object whitch properies will be aaded to export object and in on import you get imported object as argument. It's used in leash to save/restore current working directory and directory listing for completion. <li id=„pause_events“>pauseEvents [boolean] — if set to false keypress, keydown and keymap will be executed when terminal is paused. Default set to true. <li id=„soft_pause“>softPause [boolean] — if set to true it will not hide command line when paused, usefull if you want to have progress animation using propmt. Default false. </article><article id=„terminal“ readability=„26“><header><h2>Terminal object</h2></header> You will have access to terminal object in this object when you put object as first argument. In second argument if you put a function. That object is also returned by the plugin itself. The terminal is created only once so you can call that plugin multiple times. The terminal object is jQuery object extended by methods listed below. </article><article id=„instance_methods“ readability=„21“><header><h2>Instance Methods</h2></header> This is list of available methods (you can also use jQuery methods): <ul readability=„14“><li id=„clear“>clear() — clear terminal. <li id=„pause_resume“>pause([boolean])/resume() — if your command will take some time to compute (like in AJAX call) you can pause terminal (terminal will be disable and command line will be hidden) and resume it in AJAX response is called. (if you want proper timing when call exec on array of commands you need to use those functions). From version 0.11.1 pause accept optional boolean argument that indicate if command line should be visible (this can be used with animation). <li id=„paused“>paused() — return true if terminal is paused. <li id=„echo“ readability=„28“>echo([string|function], [options]) — display string on terminal — (additionally if you can call this function with a function

*as argument it will call that function and print the result, this function will be called every time you resize the terminal or browser).
There are five options: raw — it will allow to display raw html, finalize — which is callback function with one argument the div container, flush — default is true, if it's false it will not print echo text to terminal until you call flush method, wrap — default is undefined, if set to true or false it will overwritten global option, keepWords — it will not break text in the middle of the word (available from version 0.10.0). You can also use basic text formating using syntax as folow: 8212; open formating. u — underline. s — strike. o — overline. i — italic. b — bold. g — glow (using css text-shadow). ! — it will create link instead of span, <strike>you need to turn off convertLinks option for this to work</strike>. ; — separator color — color of text (hex, short hex or html name of the color). ; — separator. color — background color (hex, short hex or html name of the color). ; — separator [optional]. class — class adeed to format span element [optional]. ; — separator [optional]. text — text that will be used in data-text attribute or href it used with ! this is added automatically by split_equal function.] — end of format specification. text — text that will be formated (most of the time for internal use, when you format text that's wrap in more then one line you'll get full text in data-text attribute, it's used also for href attribute for links).] — end of formating. From version 0.4.19 terminal support ANSI formatting like \x1b[1;31mhello[0m will produce red color hello. Here is shorter description of ansi escape codes. From version 0.7.3 it also support Xterm 8bit (256) colors (you can test using this GNU Head) and formatting output from man command (overtyping). From version 0.8.0 it support html colors like blue, navy or red From version 0.9.0 Ansi escape code require unix_formatting.js file. From version 0.9.0 you can execute commands using echo (you can return command to be executed from server) using same syntax as for formatting, if you echo: <code>[[command arg1 arg2...</code> it will execute that command. If you want to execute terminal methods from JSON-RPC you can use code like this: <pre class=„javascript“>$(function() { $('body').terminal([{ exec: function(name) { var args = [].slice.call(arguments, 1); if (this[name]) { if (name == 'signature') {* if you echo signature function it will change on resize

```
                this.echo(this[name]);
            } else {
                var ret = this[name].apply(this, args);
                if (ret != this) {
                    this.echo(ret);
                }
            }
        } else {
```

```
            this.error('Command not found');
        }
    }
}, "rpc-service"], {
    checkArity: false
});
```

});</pre>

```
        <p>Then you can execute command from server by returning string
<code>"[[exec command arg1 arg2 ...]]"</code> for instance <code>"[[exec
clear]]"</code> or <code>"[[exec purge]]"</code>.</p>
      </li>
      <li id="error"><strong>error([string|function])</strong> &#8212; it
display string in in red.</li>
      <li id="exception"><strong>exception(Error, [Label])</strong>
&#8212; display exception with stack trace on terminal (second paramter is
optional is used by terminal to show who throw the exception).</li>
      <li id="level"><strong>level()</strong> &#8212; return how deeply
nested in interpreters you correctly in (It start from 1).</li>
      <li id="last_index"><strong>last_index()</strong> &#8212; return
index of last line that can be use with <strong><a
href="http://terminal.jcubic.pl/api_reference.php#update">update</a></strong
> method after you echo something and you lost the reference using -1.</li>
      <li id="login"><strong>login([function(user, password, callback),
boolean])</strong> &#8212; execute login function the same as login option
but first argument need to be a function. The function will be called with 3
arguments, user, password and a function that need to be called with truthy
value that will be stored as token. Each interpreter can have it's own login
function (you will need call <strong><a
href="http://terminal.jcubic.pl/api_reference.php#push">push</a></strong>
function and then login. The token will be stored localy, you can get it
passing true to token function. Second argument indicate if terminal should
ask for login and password infinitely.</li>
      <li id="exec"><strong>exec([string, bool])</strong> &#8212; Execute
command that like you where type it into terminal (it will execute user
defined function). Second argument is optional if set to true, it will not
display prompt and command that you execute. If you want to have proper
timing of executed function when commands are asynchronous (use ajax) then
you need to call pause and resume (make sure that you call
<strong>pause</strong> before ajax call and <strong>resume</strong> as last
in ajax response).</li>
      <li id="scroll"><strong>scroll([number])</strong> &#8212; you can
use this method to scroll manually terminal (you can pass positive or
negative value).</li>
      <li id="logout"><strong>logout()</strong> &#8212; if you use
authentication it will logout from terminal. If you don't set login option
this function will throw exception.</li>
      <li id="flush"><strong>flush()</strong> &#8212; if you echo using
option <code>flush: false</code> (it will not display text immediately) then
you can send that text to the terminal output using this function.</li>
```

```
        <li id="token"><strong>token([boolean])</strong> &#8212; return
token which was set in authentication process or by calling login function.
This is set to null if there is no login option. If you pass true as an
argument you will have local token for the interpreter (created using <a
href="http://terminal.jcubic.pl/api_reference.php#push">push</a> function)
it will return null if that interpreter don't have token.</li>
        <li id="set_token"><strong>set_token([string, boolean])</strong>
&#8212; update token.</li>
        <li id="get_token"><strong>get_token([boolean])</strong> &#8212;
same as <a
href="http://terminal.jcubic.pl/api_reference.php#token">token()</a>.</li>
        <li id="login_name"><strong>login_name()</strong> &#8212; return
login name which was use in authentication. This is set to null if there is
no login option.</li>
        <li
id="set_prompt"><strong>set_prompt([string|function(callback)])</strong>
&#8212; change the prompt.</li>
        <li id="next"><strong>next()</strong> &#8212; if you have more then
one terminal instance it will switch to next terminal (in order of creation)
and return reference to that terminal.</li>
        <li id="cols_rows"><strong>cols()/rows()</strong> &#8212; returns
number of characters and number of lines of the terminal.</li>
        <li id="history"><strong>history()</strong> &#8212; return command
line History object (need documentation - for now you can check the source
code)</li>
        <li id="name"><strong>name()</strong> &#8212; return name of the
interpreter.</li>
        <li id="push" readability="3"><strong>push([string|function],
{object})</strong> &#8212; push next interpreter on the stack and call that
interpreter. First argument is new interpreter (<strong>the same</strong> as
first argument to <strong>terminal</strong>). The second argument is a list
of options as folow:
          <ul><li><strong>name</strong> &#8212; to distinguish interpreters
using command line history.</li>
            <li><strong>prompt</strong> &#8212; new prompt for this
terminal.</li>
            <li><strong>onExit</strong> &#8212; callback function called on
Exit.</li>
            <li><strong>onStart</strong> &#8212; callback function called on
Start.</li>
            <li><strong>keydown</strong> &#8212; interpreter keydown
event.</li>
            <li><strike><strong>historyFilter</strong> &#8212; the same as
in terminal</strike> in next version.</li>
            <li><strong>completion</strong> &#8212; the same as in
terminal.</li>
            <li><strong>login</strong> &#8212; same as <a
href="http://terminal.jcubic.pl/api_reference.php#login">login</a> main
option or calling login method after push.</li>
            <li><strong>keymap</strong> &#8212; same as <a
```

```
href="http://terminal.jcubic.pl/api_reference.php#terminal_keymap">keymap in
terminal</a>.</li>
            <li><strong>mousewheel</strong> &#8212; interpreter based
mousewheel handler.</li>
            <li><strong>infiniteLogin</strong> &#8212; if set to true it
will ask infinetly for username and password if login is set.</li>
          </ul><p>Additionally everything that is passed within the object
will be stored with interpreter on the stack &#8212; so it can be
<strong>pop</strong> later. See also <a
href="http://terminal.jcubic.pl/examples.php#multiple_interpreters">Multiple
intepreters example</a>.</p>
        </li>
        <li id="pop"><strong>pop()</strong> &#8212; remove current
interpreter from the stack and run previous one.</li>
        <li id="focus"><strong>focus([bool])</strong> &#8212; it will
activate next terminal if argument is false or disable previous terminal and
activate current one. If you have only one terminal instance it act the same
as disable/enable.</li>
        <li id="enable_disable"><strong>enable()/disable()</strong> &#8212;
as names says it enable or disable terminal.</li>
        <li id="destroy"><strong>destroy()</strong> &#8212; remove
everything created by terminal. It will not touch local storage, if you want
to remove it as weel use <a
href="http://terminal.jcubic.pl/api_reference.php#purge">purge</a>.</li>
        <li id="purge"><strong>purge()</strong> &#8212; remove all local
storage left by terminal. It will act like logout because it will remove
login and token from local storage but you will not be logout until you
refresh the page.</li>
        <li id="resize"><strong>resize([number, number]</strong> &#8212;
change size of terminal if is called with two arguments (width,height) it
will resize using this values. If is called without arguments it will act
like refresh and use current size of element (you can use this if you set
size in some other way).</li>
        <li id="signature"><strong>signature()</strong> &#8212; return
JQuery Singature depending on size of terminal.</li>
        <li id="get_command"><strong>get_command()</strong> &#8212; return
current command.</li>
        <li id="insert"><strong>insert(string)</strong> &#8212; insert text
in cursor position.</li>
        <li id="export_view"><strong>export_view()</strong> &#8212; return
object that can be use to restore the view using <a
href="http://terminal.jcubic.pl/api_reference.php#import_view">import_view</
a>.</li>
        <li id="import_view"><strong>import_view([view])</strong> &#8212;
restore the view of the terminal using object returned prevoiusly by <a
href="http://terminal.jcubic.pl/api_reference.php#export_view">export_view</
a>.</li>
        <li id="set_prompt"><strong>set_prompt([string|function])</strong>
&#8212; set prompt.</li>
        <li id="get_prompt"><strong>get_prompt()</strong> &#8212; return
current prompt.</li>
```

```
        <li id="set_command"><strong>set_command(string)</strong> &#8212;
set command using string.</li>
        <li id="set_mask"><strong>set_mask([bool|string])</strong> &#8212;
toogle mask of command line if argument is true it will use maskChar as
mask.</li>
        <li id="get_output"><strong>get_output([boolean])</strong> &#8212;
return string contains whatever was print on terminal, if argument is set to
true it will return raw lines data.</li>
        <li id="freeze_frozen"><strong>freeze([boolean])/frozen()</strong>
&#8212; freeze: disable/enable terminal that can't be enabled by clicking on
terminal, frozen check if terminal has been frozen by freeze command.</li>
        <li id="read"><strong>read([string, function])</strong> &#8212;
wrapper over push, it set prompt to string and wait for text from user then
call user function with entered string.</li>
        <li id="autologin"><strong>autologin([username, token])</strong>
&#8212; autologin if you get username and token in other way, like in <a
href="https://github.com/jcubic/sysend.js">sysend</a> event.</li>
        <li id="save_state"><strong>save_state([command, boolean])</strong>
&#8212; it save current state of the terminal and update the hash. If second
argument is true it will not update hash.</li>
        <li id="history_state"><strong>history_state([boolean])</strong>
&#8212; disable or enable history sate save in hash. You can create commads
that will start or stop the recording of commands, the commands itself will
not be recorded.</li>
        <li id="clear_history_state"><strong>clear_history_state()</strong>
&#8212; clear saved history state.</li>
        <li id="reset"><strong>reset()</strong> &#8212; reinitialize the
terminal.</li>
        <li id="prefix_name"><strong>prefix_name([boolean])</strong> &#8212;
return name that is used for localStorage keys, if argument is true it will
return name of local interpreter (added by <a
href="http://terminal.jcubic.pl/api_reference.php#push">push()</a>
method).</li>
        <li id="settings"><strong>settings()</strong> &#8212; return
reference to settings object that can change options dynamicaly. Note that
not all options can be change that way, like history based options.</li>
        <li id="set_interpreter"><strong>set_interpreter([interpreter,
login])</strong> &#8212; overwrite current interpreter.</li>
        <li id="is_bottom"><strong>is_bottom()</strong> &#8212; return true
if terminal scroll is at the bottom. It use <a
href="http://terminal.jcubic.pl/api_reference.php#scrollBottomOffset">scroll
BottomOffset</a> option to calculate how much from bottom it will consider
at bottom.</li>
        <li id="scroll_to_bottom"><strong>scroll_to_bottom()</strong>
&#8212; as the name suggest is scroll to the bottom of the terminal.</li>
        <li id="complete"><strong>complete([array, options])</strong>
&#8212; automplete text based on array, usefull if custom autocomplete need
to be implemended, see <a
href="http://terminal.jcubic.pl/examples.php#autocomplete">autocomplete
example</a>. There are two options word &#8212; to indicate of completion
```

```
should be for whole command or only a word before cursor (default true) and
echo that indicate if it should echo matched commands if more then one found
(default false).</li>
        <li id="before_cursor"><strong>before_cursor([boolean])</strong>
&amp;#8212; get string before cursor if the only argument is true it will return
word otherwise it will return whole text.</li>
      </ul></article><article id="terminal_utilites"
readability="23"><header><h2>Terminal Utilites</h2></header><p>Object
<strong><abbr title="jQuery">$</abbr>.terminal</strong> contain bunch of
utilities use by terminal, but they can also be used by user code.</p>
      <ul><li id="split_equal"><strong>split_equal([string],
[number])</strong> &amp;#8212; return array. It split text into equal length
lines and keep terminal formatting in place for displaying each line
separately.</li>
        <li id="encode"><strong>encode([string])</strong> &amp;#8212; encode
&amp;amp;, new line, space, tabs, &amp;lt; and &amp;gt; with entities.</li>
        <li id="format"><strong>format([string, object]</strong> &amp;#8212;
create html &amp;lt;span&amp;gt; elements from terminal formattings. Second argument
are options with one option linksNoReferrer.</li>
        <li id="format_split"><strong>format_split([string])</strong>
&amp;#8212; return array of formatting and text between them.</li>
        <li id="escape_brackets"><strong>escape_brackets([string])</strong>
&amp;#8212; replace [ and ] with number entities.</li>
        <li id="escape_regex"><strong>escape_regex([string])</strong>
&amp;#8212; covert string that can be use in regex (RegExp constructor)
literally.</li>
        <li id="have_formatting"><strong>have_formatting([string])</strong>
&amp;#8212; test if string have terminal formatting inside.</li>
        <li id="is_formatting"><strong>is_formatting([string])</strong>
&amp;#8212; test it string is full formatting (contain only one formatted text
and nothing else).</li>
        <li id="strip"><strong>strip([string])</strong> &amp;#8212; remove
formatting from text.</li>
        <li id="active"><strong>active()</strong> &amp;#8212; return selected
terminal.</li>
        <li id="last_id"><strong>last_id()</strong> &amp;#8212; return id of the
last terminal. If you add 1 to that number it will be id of the next
terminal.</li>
        <li id="ansi_colors"><strong>ansi_colors</strong> &amp;#8212; object
contain 4 objects normal, fainted, bold and pallete (8bit colors) that
contains hex colors for ansi formatting (taken from linux terminal
emulator), <strong>NOTE: from version 0.9.0 provided by <a
href="http://terminal.jcubic.pl/js/unix_formatting.js">unix_formatting.js</a
> file.</strong></li>
        <li id="palette"><strong>palette</strong> &amp;#8212; array of 8bit
XTerm colors. <strong>NOTE: from version 0.9.0 provided by <a
href="http://terminal.jcubic.pl/js/unix_formatting.js">unix_formatting.js</a
> file.</strong></li>
        <li id="overtyping"><strong>overtyping([string])</strong> &amp;#8212;
convert string containing formatting from <strong>man</strong> command
(<i>overtyping</i>) to terminal formatting. If used with format it will
```

```
produce html from <strong>man</strong>. <strong>NOTE: from version 0.9.0
provided by <a
href="http://terminal.jcubic.pl/js/unix_formatting.js">unix_formatting.js</a
> file.</strong></li>
        <li id="from_ansi"><strong>from_ansi([string])</strong> &#8212;
convert ANSI encoding to terminal encoding. If used with format it will
produce html from ANSI encoding. <strong>NOTE: from version 0.9.0 provided
by <a
href="http://terminal.jcubic.pl/js/unix_formatting.js">unix_formatting.js</a
> file.</strong></li>
        <li id="parseArguments"><strong>parse_arguments([string])</strong>
&#8212; return array from command line string. It process number (integer
and floats) and regexes, it also convert escaped \x \0 to real characters
when inside double quote. It remove enclosing quotes from strings.</li>
        <li id="splitArguments"><strong>split_arguments([string])</strong>
&#8212; similar to <strong>parse_arguments</strong> but convert only escape
space to space and remove enclosing quotes from strings.</li>
        <li id="parseCommand"><strong>parse_command([string])</strong>
&#8212; return object with keys: <strong>name</strong>,
<strong>args</strong> and <strong>rest</strong> that contain name of the
command, it's arguments and string without command name. It use
<strong>parse_arguments</strong> function.</li>
        <li id="splitCommand"><strong>split_command([string])</strong>
&#8212; similar to <strong>parse_command</strong> but use
<strong>split_arguments</strong>.</li>
        <li id="defaults"><strong>defaults</strong> &#8212; contain all
default options used by terminal plugin. All strings are in
<strong>defaults.strings</strong> and can be translated.</li>
        <li id="normalize"><strong>normalize([string])</strong> &#8212;
function that add extra last item in formatting if not present (added in
1.3.0) .</li>
        <li id="substring"><strong>substring([string, start_index,
end_index])</strong> &#8212; return subset of the string keeping formatting,
end_index is optional (added in 1.3.0) .</li>
        <li id="unclosedStrings"><strong>unclosed_strings([string])</strong>
&#8212; return true if string have unclosed strings, it's used when parsing
command for internal use (rpc or object interpreter) if return true it will
throw exception (added in 1.3.0).</li>
        <li id="iterateFormatting"><strong>iterate_formatting([string,
callback(data)])</strong> &#8212; helper function used in substring and
split_equal that iterate over string and execute callback when in text with
object:
          <ul><li>count: number of characters in text (it skip brackets and
formatting)</li>
            <li>index: character index (including brackets and
formatting)</li>
            <li>formatting: string containing current formatting if itration
is in formatting or empyt string if not</li>
            <li>space: index of last space</li>
          </ul>
```

```
          Function added in 1.3.0
        </li>
      </ul></article><article id="cmd" readability="36"><header><h2>Command
Line</h2></header><p>Command Line is created as separate plugin, so you can
create instance of it (if you don't want whole terminal):</p>
      <pre class="javascript">
```

$('#some_id').cmd({

```
  prompt: '$&gt; ',
  width: '100%',
  commands: function(command) {
      //process user commands
  }
```

});</pre>

```
      <p>Command Line options: name, keypress, keydown, mask, enabled,
width, prompt, commands, keymap.</p>
    </article><article id="cmd-methods" readability="21"><header><h2>Command
Line Methods</h2></header><p>This is a list of methods if you are what to
use only command line.</p>
      <ul><li><strong>name([string])</strong> &#8212; if you pass string it
will set name of the command line (name is use for tracking command line
histor) or if you call without argument it will return name.</li>
        <li><strong>history()</strong> &#8212; returns instance of history
object.</li>
        <li><strong>set(string, [bool])</strong> &#8212; set command line
(optional parameter is is set to true will not change cursor position).</li>
        <li><strong>insert(string, [bool])</strong> &#8212; insert string to
command line in place of the cursor if second argument is set to true it
will not change position of the cursor.</li>
        <li><strong>get()</strong> &#8212; return current command.</li>
        <li><strong>commands([function])</strong> &#8212; set or get
function that will be called when user hit enter.</li>
        <li><strong>destroy()</strong> &#8212; remove plugin.</li>
        <li><strong>prompt([string|function])</strong> &#8212; set prompt to
function or string &#8212; if called without argument it will return current
prompt.</li>
        <li><strong>position([number])</strong> &#8212; set or get position
of the cursor.</li>
        <li><strong>resize([number])</strong> &#8212; set numbers of
characters &#8212; if called with number it will set number of character if
call without argument it will recalculate the number of characters depending
on actual size.</li>
        <li><strong>enable/disable/isenabled</strong> &#8212; guess what
they do.</li>
        <li><strong>mask([string])</strong> &#8212; if argument is true it
will mask all typed characters with provided string. If called without
argument it will return current mask.</li>
      </ul></article><article id="shortcuts"
```

```
readability="21"><header><h2>Keyboard shortcuts</h2></header><p>This is list
of keyboard shortcuts (mostly taken from bash):</p>
      <ul><li><strong>TAB</strong> &#8212; tab completion is available or
tab character.</li>
        <li><strong>Shift+Enter</strong> &#8212; insert new line.</li>
        <li><strong>Up Arrow/CTRL+P</strong> &#8212; show previous command
from history.</li>
        <li><strong>Down Arrow/CTRL+N</strong> &#8212; show next command
from history.</li>
        <li><strong>CTRL+R</strong> &#8212; Reverse Search through command
line history.</li>
        <li><strong>CTRL+G</strong> &#8212; Cancel Reverse Search.</li>
        <li><strong>CTRL+L</strong> &#8212; Clear terminal.</li>
        <li><strong>CTRL+Y</strong> &#8212; Paste text from kill area.</li>
        <li><strong>Delete/backspace</strong> &#8212; remove one character
from right/left to the cursor.</li>
        <li><strong>Left Arrow/CTRL+B</strong> &#8212; move left.</li>
        <li><strong>CTRL+TAB</strong> &#8212; swich to next terminal (use
scrolling with animation) &#8212; don't work in Chrome.</li>
        <li><strong>Right Arrow/CTRL+F</strong> &#8212; move right.</li>
        <li><strong>CTRL+Left Arrow</strong> &#8212; move one word to the
left.</li>
        <li><strong>CTRL+Right Arrow</strong> &#8212; move one word to the
right.</li>
        <li><strong>CTRL+A/Home</strong> &#8212; move to beginning of the
line.</li>
        <li><strong>CTRL+E/End</strong> &#8212; move to end of the
line.</li>
        <li><strong>CTRL+K</strong> &#8212; remove the text after the cursor
and save it in kill area.</li>
        <li><strong>CTRL+U</strong> &#8212; remove the text before the
cursor and save it in kill area.</li>
        <li><strong>CTRL+V/SHIFT+Insert</strong> &#8212; insert text from
system clipboard.</li>
        <li><strong>CTRL+W</strong> &#8212; remove text to the begining of
the word (don't work in Chrome).</li>
        <li><strong>CTRL+H</strong> &#8212; remove text to the end of the
line.</li>
        <li><strong>ALT+D</strong> &#8212; remove one word after the cursor
&#8212; don't work in IE.</li>
        <li><strong>PAGE UP</strong> &#8212; scroll up &#8212; don't work in
Chrome.</li>
        <li><strong>PAGE DOWN</strong> &#8212; stroll down &#8212; don't
work in Chrome.</li>
        <li><strong>CTRL+D</strong> &#8212; run previous interpreter from
the stack or call logout (if terminal is using authentication and current
interpreter is the first one). It also cancel all ajax call, if terminal is
paused, and resume it.</li>
      </ul></article><article id="additional"
readability="25"><header><h2>Additional terminal
```

```
controls</h2></header><p>All interpreters have attached
<strong>mousewheel</strong> event so you can stroll them using mouse. To
swich between terminals you can just <strong>click on terminal</strong> that
you want to <strong>activate</strong> (you can also use <a
href="http://terminal.jcubic.pl/api_reference.php#focus">focus</a>
method).</p>
      <p>If you select text using mouse you can paste it using middle mouse
button (from version 0.8.0).</p>
    </article><article id="change_colors"
readability="28"><header><h2>Changing Colors</h2></header><p>To change color
of terminal simply modify "jquery.terminal.css" file it's really short and
not complicated, but you should set inverted class background-color to be
the same as color of text.</p>
      <p>To change color of one line you can call css jquery method in
finalize function passed to echo function.</p>
      <pre class="javascript">terminal.echo("hello blue", {
  finalize: function(div) {
      div.css("color", "blue");
  }
```

});</pre>

```
      <p>You can also use <a
href="http://terminal.jcubic.pl/api_reference.php#echo">formating using
echo</a> function. To change whole terminal colors see <a
href="http://terminal.jcubic.pl/api_reference.php#style">style
section</a>.</p>
    </article><article id="translation"
readability="23"><header><h2>Translation</h2></header><p>All strings used by
the plugin are located in <code>$.terminal.defaults.strings</code> object,
so you can translate them and have i18n.</p>
    </article><article id="errors" readability="28"><header><h2>Error
Handling</h2></header><p>All exceptions in user functions (interpreter,
prompt, and greetings) are catch and proper error is displayed on terminal
(with stack trace). If you want to handle exceptions differently you can add
<a
href="http://terminal.jcubic.pl/api_reference.php#exceptionHandler">exceptio
nHandler</a> option and create different logic, for instance send exceptions
to server or show just exception name without stack trace.</p>
    </article><article id="style"
readability="93"><header><h2>Style</h2></header><p>From version 0.8.0
blinking cursor is created using CSS3 animations (if available) so you can
change that animation anyway you like, just look at <a
href="http://terminal.jcubic.pl/css/jquery.terminal.css">jquery.terminal.css
</a> file. If browser don't support CSS3 animation blinking is created using
JavaScript.</p>
      <p>To change color of the cursor to green and backgroud to white you
can use this css:</p>
      <pre class="css">.terminal, .cmd {
  background: white;
```

```
  color: #0f0;
```

} .terminal .inverted, .cmd .inverted, .cmd .cursor.blink {

```
  background-color: #0f0;
  color: white;
```

} @-webkit-keyframes terminal-blink {

```
0%, 100% {
    background-color: #fff;
    color: #0f0;
}
50% {
    background-color: #0e0;
    color: #fff;
}
```

} @-ms-keyframes terminal-blink {

```
0%, 100% {
    background-color: #fff;
    color: #0f0;
}
50% {
    background-color: #0e0;
    color: #fff;
}
```

} @-moz-keyframes terminal-blink {

```
0%, 100% {
    background-color: #fff;
    color: #0f0;
}
50% {
    background-color: #0e0;
    color: #fff;
}
```

} @keyframes terminal-blink {

```
0%, 100% {
    background-color: #fff;
    color: #0f0;
}
50% {
    background-color: #0e0;
    color: #fff;
```

```
}
```

}</pre>

```
      <p>From version 1.0.0 you can use css variables with code like
this:</p>
      <pre class="css">.terminal {
--color: rgba(0, 128, 0, 0.99);
--background: white;
```

}</pre>

```
      <p>If you want to have consistent selection you should use rgba color
with 0.99 transparency, see this <a
href="http://stackoverflow.com/a/7224621/387194">stackoverflow
answer</a>.</p>
      <p>The only caveat is that css variables are not supported by IE nor
Edge.</p>
      <p>To change cursor to vertical bar you can use this css:</p>
      <pre class="css">.cmd .cursor.blink {
color: #aaa;
border-left: 1px solid #aaa;
background-color: black;
margin-left: -1px;
```

} .terminal .inverted, .cmd .inverted, .cmd .cursor.blink {

```
  border-left-color: #000;
```

} @-webkit-keyframes terminal-blink {

```
0%, 100% {
    border-left-color: #aaa;
}
50% {
    border-left-color: #000;
}
```

} @-ms-keyframes terminal-blink {

```
0%, 100% {
    border-left-color: #aaa;
}
50% {
    border-left-color: #000;
}
```

} @-moz-keyframes terminal-blink {

```
0%, 100% {
```

```
    border-left-color: #aaa;
}
50% {
    border-left-color: #000;
}
```

} @keyframes terminal-blink {

```
0%, 100% {
    border-left-color: #aaa;
}
50% {
    border-left-color: #000;
}
```

}</pre>

```
      <p>From 1.0.0 version you can simplify this using this css:</p>
      <pre class="css">.terminal {
--color: rgba(0, 128, 0, 0.99);
--background: white;
--animation: terminal-bar;
```

}</pre>

```
      <p>If you need to support IE or Edge you can set animation using:</p>
      <pre class="css">.cmd .cursor.blink {
  -webkit-animation-name: terminal-underline;
     -moz-animation-name: terminal-underline;
      -ms-animation-name: terminal-underline;
          animation-name: terminal-underline;
```

} .terminal .inverted, .cmd .inverted {

```
  border-bottom-color: #aaa;
```

}</pre>

```
    <p>Or this css for bar cursor:</p>
    <pre class="css">.cmd .cursor.blink {
  -webkit-animation-name: terminal-bar;
     -moz-animation-name: terminal-bar;
      -ms-animation-name: terminal-bar;
          animation-name: terminal-bar;
```

} .terminal .inverted, .cmd .inverted {

```
  border-left-color: #aaa;
```

}</pre>

```
      <p>To change the color of the cursor with differerent animation that
will work in IE or Edge you will need to create new <code>@keyframes</code>
with different colors, like in previous examples.</p>
      <p>To change font size of the terminal you can use this code:</p>
      <pre class="css">.terminal, .cmd, .terminal .terminal-output div div,
.cmd .prompt {
  font-size: 20px;
  line-height: 24px;
```

}</pre>

```
      <p>Or from version 1.0.0 (and if you don't care about IE or Edge) you
can simplify the code using --size css variables like this:</p>
      <pre class="css">.terminal {
--size: 2;
```

}</pre>

```
      <p>The size is relative to original size so 1 is normal size 2 is
double size.</p>
      <p>You can take a look at the <a
href="http://codepen.io/jcubic/pen/xReWxJ?editors=0100">demo</a>.</p>
    </article><article id="keyboard" readability="27"><header><h2>Keyboard
events</h2></header><p>There are 3 keyboard events (all of them you can add
in terminal, cmd and push command):</p>
      <ul><li><strong>keymap</strong> &amp;#8212; simpler events you can add
uppercase shortcut like CTRL+V, the callback function is <code>function(e,
original) {</code>, the original is original function callback that can be
called, because your function overwrite original behvaior.</li>
        <li><strong>keydown</strong> &amp;#8212; this event is fired before
<strong>keymap</strong> so you can return false to prevent default
<strong>keymap</strong></li>
        <li><strong>keypress</strong> &amp;#8212; is used to handle inserting of
characters if you want to prevent certain characters to be inserted you can
return false for those characters.</li>
      </ul><p>Caveats: the shortcut CTRL+D is handled by both
<strong>keydown</strong> and <strong>keymap</strong>. If terminal is paused
is handled by <strong>keydown</strong> and if not in
<strong>keymap</strong>. If you want to overwrite CTRL+D when terminal is
paused you need to pass false to <a
href="http://terminal.jcubic.pl/api_reference.php#pause_events">pauseEvents<
/a> option and use <strong>keydown</strong> otherwise you need to add
function to <strong>keymap</strong>.</p>
    </article><article id="authentication"
readability="37"><header><h2>Authentication</h2></header><p>You can provide
your authentication function which will be called when user enter login and
password. Function must have 3 arguments first is <strong>user
name</strong>, second his <strong>password</strong> and third is
<strong>callback function</strong> which must be called with token or falsy
```

```
value if user enter wrong user and password. (You should call server via
AJAX to authenticate the user).</p>
      <p>You can retrieve token from terminal using <a
href="http://terminal.jcubic.pl/api_reference.php#token">token method</a> on
terminal instance. You can pass this token to functions on the server as
first parameter and check if it's valid token.</p>
      <p>If you set interpreter to string (it will use this string as URI
for JSON-RPC service) you can set login function to string (to call custom
method on service passed as interpreter) or true (it will call login
method).</p>
      <p>If you set URI of JSON-RPC service and login to true or string, it
will always pass token as first argument to every JSON-RPC method.</p>
    </article><article id="3rd" readability="20"><header><h2>Third party
code and additional plugins</h2></header><p>Terminal include this 3rd party
libraries:</p>
      <ul><li>Storage plugin by Dave Schindler.</li>
        <li><a href="http://jquery.offput.ca/timers/">jQuery
Timers</a>.</li>
        <li>Cross-Browser Split 1.1.1 by Steven Levithan.</li>
        <li>jQuery Caret by Gideon Sireling.</li>
        <li>sprintf.js by Alexandru M&#259;r&#259;&#537;teanu.</li>
      </ul><p>terminal also define 2 helper functions:</p>
      <ul><li>$.jrpc &#8212; JSON-RPC helper function.</li>
        <li>$.omap &#8212; version of map that handle objects.</li>
        <li><strike>$.json_stringify &#8212; terminal own JSON stringify,
because prototype library used by biwascheme messed up
JSON.stringify.</strike></li>
        <li>$.fn.scroll_element &#8212; plugin that return scroll object for
normal objects that the same object but for body element it return html or
body depend on which one need to be scrolled.</li>
        <li>$.fn.resizer &#8212; helper plugin that execute callback when
element is resized. If called with string 'unbind' it will remove the event.
Based on <a
href="https://github.com/marcj/css-element-queries/blob/master/src/ResizeSen
sor.js">ResizeSensor.js file from marcj/css-element-queries</a>.</li>
      </ul></article></section><!-- Piwik --><noscript><p><img
src="http://piwik.jcubic.pl/piwik.php?idsite=1" alt=""/></p></noscript>
  <!-- End Piwik Code -->
    </body>
```

</html>