

Dialogs - The Web Browser as a Graphical User Interface for Ruby App

Table of Contents

RubyWebDialogs.....	1
The Web Browser as a Graphical User Interface for Ruby Applications.....	1
1. Introduction.....	2
2. Internals.....	4
3. Usage.....	5
3.1. Screen Definitions.....	5
a) Help Screens.....	6
3.2. Ruby Script.....	7
a) Standard Methods.....	8
b) Standard Instance Variables.....	8
c) message, error and text.....	9
d) timeout.....	9
e) progressbar.....	9
f) download.....	12
g) samewindow?.....	12
h) Extended Classes.....	12
3.3. Files.....	13
3.4. Network.....	13
3.5. The Application.....	14
a) Buttons.....	15
b) Configuration File.....	15
c) Port Range.....	15
d) Browser.....	16
e) Theme.....	16
3.6. RWDReconnect.....	17
4. Examples.....	18
4.1. Hello World.....	18
4.2. A Little Calculator.....	18
4.3. Panels.....	19
4.4. Tabs.....	20
4.5. message and error.....	21
4.6. Progress Bar.....	23
4.7. download.....	23
4.8. Japanese.....	24
4.9. An Address Book.....	25
4.10. SetBackground.....	26
5. License.....	28
6. Download.....	29
7. Known Issues.....	30

RubyWebDialogs

The Web Browser as a Graphical User Interface for Ruby Applications

Fri May 25 15:06:25 UTC 2007

Erik Veenstra <rubywebdialogs@erikveen.dds.nl>

1. Introduction

RubyWebDialogs is a platform independent graphical user interface for Ruby applications. It generates HTML and serves it with an internal HTTP server, so you can use your favorite web browser as the front end for your Ruby application. All this means, that it can be used on almost every platform, like Ruby itself.



The basic idea of RubyWebDialogs is to keep development simple, so one can build simple applications in a couple of minutes and not-that-simple applications in a couple of hours. It's not meant for building big applications or things like adding functionality to the corporate's web site. (They'll use Java, anyway...) But if things get faster, and more stable, and more feature-rich, and more community driven, and more integrated with WEBrick, and Apache, well, who knows what all this will lead to... For now, it's just a thing I've already used for quiet some time and want to share.

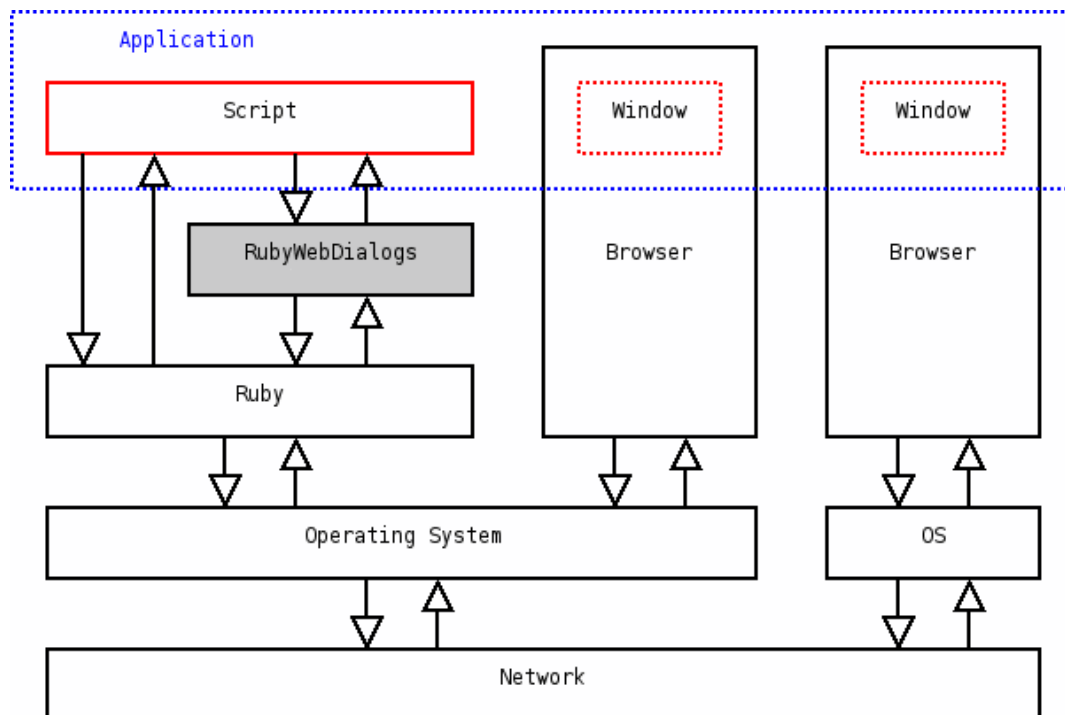
You don't have to know anything about HTML, or HTTP, or cookies, or TCP/IP or anything else that's necessary to communicate with browsers over a network. All that has to be and is covered by RubyWebDialogs. The only thing you need to know, besides Ruby, is the very basics of XML, because it's used to define the layout of the screens.

RubyWebDialogs doesn't require any external packages, no RubyXML, no WEBrick, no Amrita. Just plain Ruby. Version 1.6 or 1.8, maybe even an older one.

RubyWebDialogs applications can be used over a network as well. A simple authentication mechanism is built in.

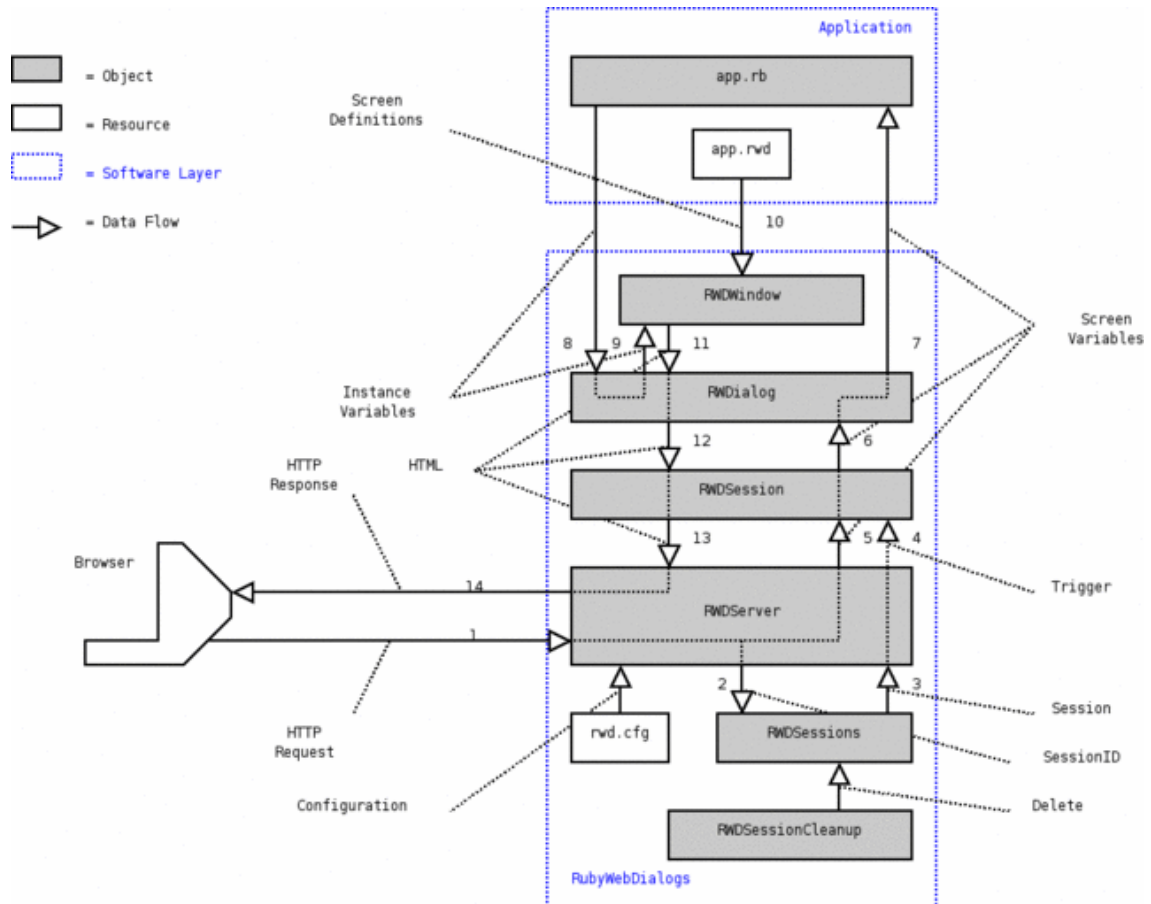
Distributing your application is easy with [Tar2RubyScript](#) and [RubyScript2Exe](#).

1. Introduction



2. Internals

Have a look at the code...



3. Usage

See also [A Little Calculator](#). It's almost self-explaining.

3.1. Screen Definitions

The screen definitions are defined with (inaccurate?) XML. RubyWebDialogs parses this XML and generates HTML.

Here's an example:

```
<application>
  <window name="main" title="HW">
    <p>Hello World!</p>
  </window>
</application>
```



Multiple windows can be defined. One of them must have the name "main".

`%variable%` and `%%variable%%` in the screen definitions are replaced by `@variable.to_s`.

Suppose `@user` is set to "Erik":

```
<application>
  <window name="main" title="HE">
    <p>Hello %user%!</p>
  </window>
</application>
```



`%variable%` is supposed to be replaced by normal text. `%%variable%%` is supposed to be replaced by a new piece of screen definitions. For now, both are handled (almost) the same. A single "%" on the screen is achieved by %% (a double "%") in the screen definitions.

Suppose `@tag` is set to "`Bold Text`":

```
<application>
  <window name="main" title="Tag1">
    <p>%tag%</p>
  </window>
</application>
```

3. Usage

With %tag%, "Bold Text" is handled as text:



```
<application>
  <window name="main" title="Tag2">
    <p>%%tag%%</p>
  </window>
</application>
```

With %%tag%%, "Bold Text" is handled as a screen definition:



Here is a list of the screen elements and their attributes.

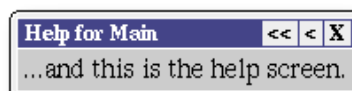
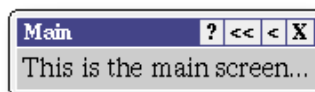
(Is somebody able and willing to make a DTD?)

a) Help Screens

When the screen definitions have a help window with the same name as the current window, a [?] will be added to the top of the window. This is a simple way of having help windows in your application.

```
<application>
  <window name="main" title="Main">
    <p>This is the main screen...</p>
  </window>

  <helpwindow name="main" title="Help for Main">
    <p>...and this is the help screen.</p>
  </helpwindow>
</application>
```



3.2. Ruby Script

First of all, you have to require the file `ev/rwd.rb`:

```
require "ev/rwd"
```

And your class has to be a subclass of the class `RWDialog`:

```
class YourUIClass < RWDialog
  .
  .
  .
end
```

Finally, create an object and serve it on the first free TCP port in the range 7701-7709 (see [Port Range](#)):

```
youruiobject = YourUIClass.file("screendefinitions.rwd")
youruiobject.serve
```

When an object of *YourUIClass* is created, `@rwd_action` and `@rwd_window` are set to "main" (the default). (`@rwd_action` is usually set in a window (e.g. action of a button) and used to determine which method is to be called. `@rwd_window` is usually set in the method and used to determine which window is to be rendered.) After that, a couple of things happen:

1. Because `@rwd_action` is set to "main", *youruiobject*.main is called (if it exists).
2. Because `@rwd_window` is set to "main", the window named "main" (`<window name="main">`) will be searched for in the screen definitions.
3. The window is rendered and served, accessible by a web browser from the local host (default behavior).

When a button on the window in the browser is pushed, all input fields of that window are converted to instance variables and `@rwd_action` is set to the action (= method) of the button. This method is invoked, does the things it has to do (business logic) and usually sets `@rwd_window` to the next window. Finally, this window is rendered and served. And the story starts over again:

youruiobject.newmethod is called and `<window name="newwindow">` is rendered and served. This will go on and on... Like `@rwd_window`, you can also set `@rwd_tab` for rendering a specific tab of a window.

youruiobject stays alive as long as the application is running. That means that the state of *youruiobject* is preserved on the server, not in the browser.

You might use *youruiobject.serve(port)* to serve your application on port *port*.

We used `RWDialog.file(rwdfile)` in this example to separate the screendefinitions from the code. But with `RWDialog.new(rwd)`, it's possible to use a string that already contains the screendefinitions itself, instead of the name of the file in which the screendefinitions are stored. The *s* in `RWDialog.new(s)` are screendefinitions; the *s* in `RWDialog.file(s)` is the filename in which the screendefinitions reside. The [Examples](#) section has examples of both variants.

3. Usage

(I have to enhance the algorithm for checking for free ports. Running 2 or more RubyWebDialogs applications at the same time is no problem, but starting them at exactly the same moment (during the scanning of used ports) is not a good idea, yet...)

a) Standard Methods

Only a couple of methods are important for your application:

- `RWDialog.new(rwd)`
- `RWDialog.file(rwdfile)`
- `RWDialog#serve(port=nil, auth=nil, realm=self.class.to_s)`
- `RWDialog#message(text)`
- `RWDialog#error(text)`
- `RWDialog#text(text)`
- `RWDialog#timeout(seconds)`
- `RWDialog#progressbar() {}`
- `RWDialog#download(data)`
- `RWDialog#samewindow?`

The two class methods create a new object of the class `RWDialog`. You only need one of them. Don't redefine `YourUIClass#initialize`, unless you do a `super`. The other five methods are explained elsewhere.

b) Standard Instance Variables

All instance variables used by RubyWebDialogs internally, start with `@rwd_`. They can't be shown on the screen. (They are kind of contained by `@rwd_ignore_vars`.)

Some instance variables are used to influence the behavior of RubyWebDialogs:

- `@rwd_action`
- `@rwd_window`
- `@rwd_tab`
- `@rwd_ignore_vars`
- `@rwd_call_after_back`

The array `@rwd_ignore_vars` excludes the variables that may not or can not be shown on the screen. Excluding variables speeds up the rendering a little bit as well. Just fill it like this:

```
@rwd_ignore_vars << "@variable"
```

When `<back/>` or `[<]` is hit, the previous window is rendered, but the previous method (which has lead to the previous window) isn't called. Sometimes you do want to call it after going back, which is achieved by adding that method to the array `@rwd_call_after_back`:

```
@rwd_call_after_back << "method"
```

Suppose this: Method `m1` has led to window `w1`. After hitting a button on the window, method `m2` is invoked and window `w2` is rendered. Normally, when hitting `[<]` on window `w2`, window `w1` will be rendered, but without invoking method `m1` (which probably set the values for the window, the first time). If you do want it to be invoked, you have to do a:

3. Usage

```
@rwd_call_after_back << "m1"
```

The other variables are already explained before.

c) message, error and text

Two default windows are provided, besides the ones used for internal purposes:

- `RWDMessage`, activated with `RWDDialog#message(text)`.
- `RWDError`, activated with `RWDDialog#error(text)`.

Both `RWDDialog#message` and `RWDDialog#error` force `RubyWebDialogs` to show a window, containing the given text. This text may contain screen definitions, like `
` and `<hr/>` (notice the slashes, it's XML, not HTML).

`RWDDialog#text` does the same, but shows the given text as body of an HTML page. No windows. The message is simply wrapped by `"<html><body><pre>"` and `"</pre></body></html>"`. The given text may contain HTML tags.

There is something special about these methods: The desired "pop-up" window (or HTML page) doesn't appear at the moment you call these methods. All these methods do is setting some flags, so the "pop-up" window will be rendered after the current method has ended.

See also [message and error](#).

d) timeout

If the user closes the browser instead of closing the application, the application keeps running for ever. It's possible to search for "headless" processes with [RWDReconnect](#). You could as well set a timeout with `RWDDialog#timeout(seconds)`, so the application commits suicide once the time passed since the last action is greater than or equal to the given number of seconds:

```
def main
  timeout(60*60)
end
```

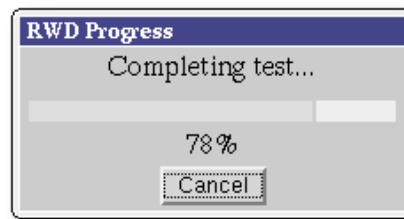
e) progressbar

(This is an experimental feature.)

The response time in a GUI-application is important. Customers get irritated if there's no response for more than half a second. You can occasionally stretch that to a couple of seconds, especially when the customer knows that the action they initiated takes a bit longer to complete. But if there's no response for more than 3 seconds, they start pushing the stop button or, even worse, the back button. That's not what you want.

If you know that the response time will be more than 3 seconds, you have to show them a progress bar. The user can see that the program is busy, so he can get a coffee if he wants to.

3. Usage



If you only want to use `RWDDialog#progressbar`, without knowing how it works: Surround all the code of your method with `progressbar` and add an instance variable which indicates the progress.

Old code:

```
def yourmethod do
  @rwd_window = next window

  some iteration do
    ...the real work...
  end
end
```

New code:

```
def yourmethod do
  progressbar(1, "Waiting for action to complete...", @progress) do
    @rwd_window = next window

    some iteration do
      ...the real work...
      @progress = some value between 0.0 and 1.0
    end
  end
end
```

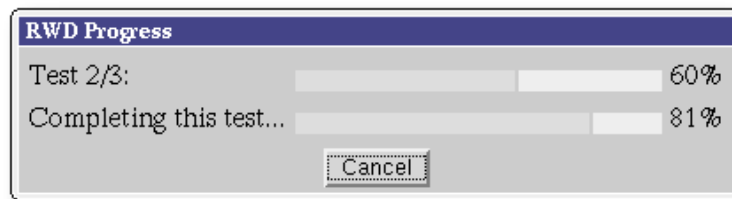
That's all!

You can actually use `RWDDialog#progressbar` in several ways:

```
progressbar(refresh, text, progress) {}
progressbar(refresh, [text, progress]) {}
progressbar(refresh, text1, progress1, text2, progress2) {}
progressbar(refresh, [text1, progress1], [text2, progress2]) {}
and so on...
```

I added a [demo](#).

3. Usage



Skip reading if you're satisfied. Read on if you dare...

The behavior of `RWDDialog#progressbar` might be a bit strange. I'm just experimenting...

The general syntax is:

```
def yourmethod do

  code A

  progressbar(1, "Waiting for action to complete...", @progress) do
    @rwd_window = next window

    some iteration do
      .
      .
      @progress = some value between 0.0 and 1.0
      .
      .
    end
  end

  code B

end
```

`RWDDialog#progressbar`, when invoked, sets some flags and values, starts the given block in the background and gives the control back to your method (*code B*). The progress bar itself, on the screen, will be shown *after* the current method has ended. In the meanwhile, the long running code (in the block) is being executed in the background, doing the real work and setting an instance variable (e.g. `@progress`) to a value between 0.0 and 1.0. The browser reloads the current page after 1 second (first parameter) and your method is invoked once again: *Code A* is executed, `progressbar` is invoked and *code B* is executed. This time, `progressbar` detects that the code in the block is already running, so it doesn't start the given block. This goes on and on. When the real work has ended, `progressbar` resets its flags and values and, at the end, the desired window (`@rwd_window`) is rendered like before.

Difficult? Well, maybe it is under water (like everything else of `RubyWebDialogs...`), but I tried to make the use of it really easy: Surround all the code of your method with `progressbar` and add an instance variable which indicates the progress. That's all! No *code A*, no *code B*. Just a block of code which is executed only once, although the method itself (and `progressbar`) is invoked several times.

3. Usage

f) download

With `RWDDialog#download(data, filename=" ")`, you have a facility to let the user download some data directly to a file.

Like **message, error and text**, `RWDDialog#download` just sets a flag. The download will start *after* the current method has ended.

You can't start a download and go to another window in one click, simply because the browser expects only one response after one request.

See also **download**.

(RWDDialog#download doesn't work with the IE which comes with a clean installation of Windows 98. The examples on the 'Net don't either, so it's not _my_fault... Any solutions?)

(A similar RWDDialog#upload is under construction, but is much more complicated.)

g) samewindow?

The method `RWDDialog#samewindow?` tells you whether RWD is about to render the same window as the last one it rendered: Is the window on top of the stack the same as `@rwd_window`? Sometimes you want to know if you enter a window or rebuild a window. At least, I do... Well, I did, once...

h) Extended Classes

Some standard classes are extended with RubyWebDialogs-specific methods. These methods transform the objects into RubyWebDialogs screen definitions. Strictly spoken, these extension are not part of RubyWebDialogs.

Have a look at the **examples**.

<code>Array#rwd_method(method)</code>	Transforms the array into a list of <code>Element</code> .
<code>Array#rwd_options(emptyline=nil)</code>	Transforms the array into a list of options for embedding in <code><SELECT></code> .
<code>Array#rwd_row(key=nil, value=nil, bold=false)</code>	Transforms the array into a list of table rows.
<code>Array#rwd_headers(emptyfield=false)</code>	Transforms the array into a table header.
<code>Array#rwd_form(prefix, values=[], twoparts=0, options={})</code>	Transforms the array into a form.
<code>Array#rwd_table(headers=nil, highlightrows=[])</code>	Transforms the array into a table.
<code>Hash#rwd_table(field=nil, joinwith=@sep, headers=nil)</code>	Transforms the hash into a table.

3.3. Files

The internal web server serves files as well. The files have to be in the directory `rwd_files`, relative to the directory in which the script started. (Well, to be accurate: The `Dir.pwd` at the moment `ev/rwd.rb` was required.) This gives us the opportunity to load images and serve HTML and other files.

For example: Someone entered `http://localhost:7701/images/test.gif` in the browser. The internal web server will serve the file `./rwd_files/images/test.gif`.

```
<application>
  <window name='main' title='HTML'>
    <p><a href='test.html'>Click here to go to an HTML page.</a></p>
  </window>
</application>
```

There's one exception: `/rwd_pixel.gif` is stored hard-coded in `ev/rwd`, so you can't serve `./rwd_files/rwd_pixel.gif`, even when it exists.

See also [Japanese](#).

3.4. Network

It's possible to serve an application over a network. That's achieved by adding a second parameter to the method `RWDDialog#serve` of the object. (The first parameter is the TCP port.) This can be either a hash (`{ "user1" => "password", "user2" => "pass phrase" }`), an empty string (no authentication) or a string with the name of the authorization file, relative to the home directory of the user that started the service. This file has to be in the form of:

```
user1 = password
user2 = pass phrase
```

User and password are separated by `/\s*=\s*/`.

The home directory is defined as `(ENV["HOME"] or ENV["USERPROFILE"] or (File.directory?("h:/") ? "h:" : "c:")).gsub(/\//, "/") + "/"`.

This file is loaded every time a login is attempted, so the server keeps running, even when a password has been changed.

This might have been one of my cases:

```
myobject.serve(1234, ".rwduids/myapp")
```

And the file `/home/erik/.rwduids/myapp` containing:

```
erik = secret
```

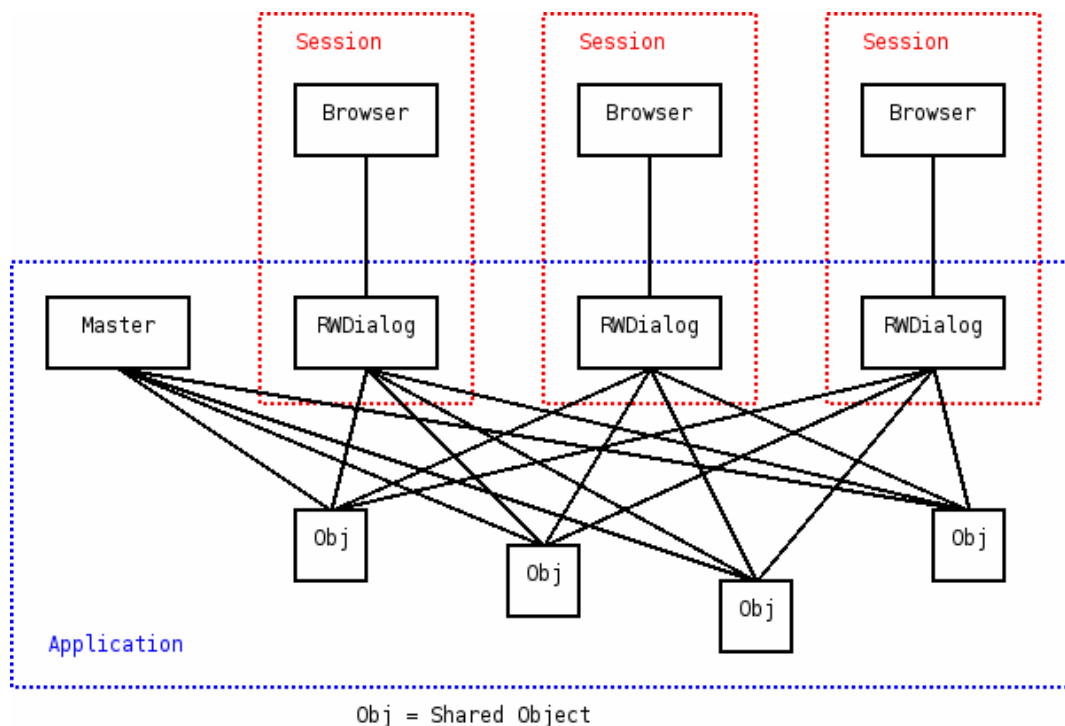
Now I can login with:

3. Usage



If the second parameter to `RWDialog#serve` is `nil`, the application is running in "no-network mode". If the second parameter to `RWDialog#serve` is not `nil`, the application is running in "network mode".

In network mode, the `RWDialog` object you've created in your script will be the master of which a copy will be made for every connection of a browser (`youruiobject.clone`). So every user has an object of his own on the server. But it's just the `RWDialog` object that gets cloned. All objects after the first one are shared.



`RWDialog` objects have a time-out of 24 hours, unless your application is running in the no-network mode.

For now, `RubyWebDialogs` might still be vulnerable for "Ruby-injection" (like in "SQL-injection"). I need to have a closer look. For now, I've set `$SAFE` to 2.

(Is Safe mode 2 good enough?)

3.5. The Application

a) Buttons

[?]	Help for this screen.
[<<]	Go back to the beginning of the application.
[<]	Go back one screen.
[X]	Terminate the application. / Log out.

Closing the browser doesn't terminate the application. You have to use the application's close-button (see **RWDReconnect**). And terminating the application doesn't terminate the application when running in network mode. You just log out.

Don't use the back-button of the browser! Strange and unpredictable things may and will happen!

Hitting enter is like pressing the first button of the window. (Unless you use Konqueror...)

b) Configuration File

The configuration file is searched for on three locations, in this order:

1. ENV["HOME"] + ".rwdrc"
2. ENV["USERPROFILE"] + ".rwd.cfg"
3. ENV["windir"] + ".rwd.cfg"

RubyWebDialogs will only use the first file it can find. Every line in this file is treated as a key-value pair, separated by `/\s*=\s*/`. These pairs are added to ENV, unless ENV already has the key. Keys are case sensitive!

My `~/ .rwdrc` on Linux contains:

```
RWDPORTS      = 7701-7799
RWDBROWSER    = mozilla -CreateProfile RWD-%port% ; mozilla -P RWD-%port%
RWDTHEME      = WINDOWSLOOKALIKE
```

This results in:

```
ENV[ "RWDPORTS" ]      = "7701-7799"
ENV[ "RWDBROWSER" ]    = "mozilla -CreateProfile RWD-%port% ; mozilla -P RWD-%port%"
ENV[ "RWDTHEME" ]      = "WINDOWSLOOKALIKE"
```

These ENV will/can be used later-on.

c) Port Range

ENV["RWDPORTS"] is the range in which a free TCP port will be searched for. If it isn't set, the range 7701-7709 will be used.

d) Browser

If ENV includes "RWDBROWSER", ENV["RWDBROWSER"], will be started in a thread, with "http://localhost:#{port}/" as argument. It's possible to use %port% in ENV["RWDBROWSER"], which is replaced with the port number. On Windows, you need to give the full path to the executable, e.g. "C:\Program Files\Internet Explorer\IEXPLORE.EXE", including the quotes.

For me, this results in:

```
Thread.new do
  system('mozilla -CreateProfile RWD-7701 ; mozilla -P RWD-7701 \
        "http://localhost:7701/"')
end
```

On Windows, the browser found at

Win32::Registry::HKEY_CLASSES_ROOT.open('htmlfile\shell\open\command') { |reg| ENV["RWDBROWSER"] = reg[" "] } is used if no ENV["RWDBROWSER"] is found. That defaults to IE. Well, it's good enough... Uh, IE 4.0 never ends with exit code 0, so RubyWebDialogs tells you that the browser died...

On Linux, the first browser found in the following list will be the default:

- galeon
- mozilla
- firefox
- opera
- konqueror
- htmlview

By the way, all this is only true for the no-network mode. When serving the application over a network, no browser is started at all. Whatever your settings are.

e) Theme

You can change the way a window looks, by changing ENV["RWDTHEME"]. For now, two themes are part of RubyWebDialogs: "DEFAULT" (which is the default...) and "WINDOWSLOOKALIKE". The "DEFAULT" theme is a lot more complex than the "WINDOWSLOOKALIKE" theme, so the latter will be rendered faster by the browser.

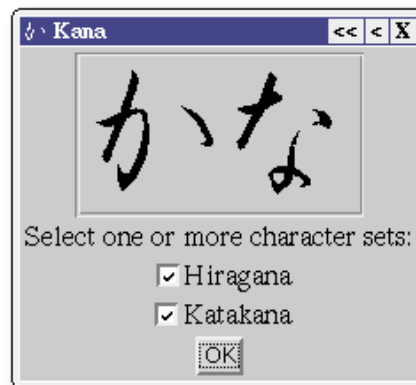
If you connect a PDA (at least PocketPC 2002, might be true for PALM and friends) to your application, RubyWebDialogs recognize it's (dis)abilities and uses a simplified theme, called "PDA".

If you want to change the default from "DEFAULT" to "WINDOWSLOOKALIKE" in your program (it can still be overwritten in the **Configuration File** by the user), you can add the following line *before* requiring ev/rwd:

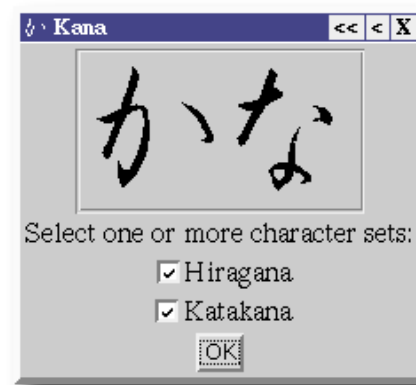
```
ENV[ "RWDTHEME" ] = (ENV[ "RWDTHEME" ] or "WINDOWSLOOKALIKE")
```

"DEFAULT" theme:

3. Usage



"WINDOWSLOOKALIKE" theme:



3.6. RWDRReconnect

If you accidentally close the browser without terminating the application, your application keeps running for ever... You could reconnect your browser manually, but do you remember which port it's listening on? That's where `rwdreconnect.rbw` comes in. It scans all ports in the reserved `range` and eventually reconnects a browser to every application.

4. Examples

The examples in this document are distributed as executable packages (or RBA's, Ruby archives, like JAR's, Java archives), built with **Tar2RubyScript**. You can run them without installing RubyWebDialogs. Pack them with **RubyScript2Exe**, if you want your friends to run them.

4.1. Hello World

Get **rwdhelloworld.rbw** or **rwdhelloworld.tar.gz** . You can run it without installing RubyWebDialogs.

```
$ ruby rwdhelloworld.rbw
```



File helloworld.rwd:

```
<application>
  <window name="main" title="HW">
    <p>Hello World!</p>
  </window>
</application>
```

File helloworld.rbw:

```
require "ev/rwd"

class Helloworld < RWDDialog
end

Helloworld.file("helloworld.rwd").serve
```

Do I have to explain it?...

Here's a one-liner:

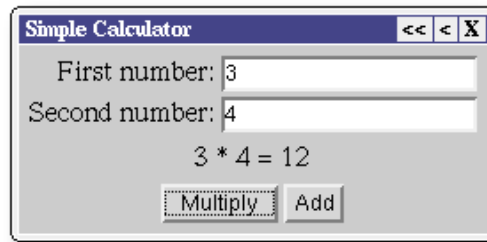
```
$ ruby -r 'ev/rwd' -e '
RWDDialog.new("<window name=main title=HW><p>Hello World!</p></window>").serve'
```

4.2. A Little Calculator

Get **rwdcalc.rbw** or **rwdcalc.tar.gz** . You can run it without installing RubyWebDialogs.

```
$ ruby rwdcalc.rbw
```

4. Examples



File `rawdcalc.rwd`:

```
<application>
  <window name="main" title="Simple Calculator">
    <table>
      <row> <p align="right">First number:</p> <text name="a"/> </row>
      <row> <p align="right">Second number:</p> <text name="b"/> </row>
    </table>
    <p>%result%</p>
    <horizontal>
      <button caption="Multiply" action="multiply"/>
      <button caption="Add" action="add"/>
    </horizontal>
  </window>
</application>
```

File `rawdcalc.rbw`:

```
require "ev/rwd"

class Demo < RWDDialog
  def multiply
    @result = "%s * %s = %s" % [@a, @b, @a.to_i*@b.to_i]
  end

  def add
    @result = "%s + %s = %s" % [@a, @b, @a.to_i+@b.to_i]
  end
end

Demo.file("rawdcalc.rwd").serve
```

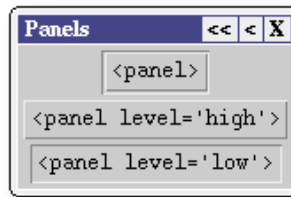
On startup, both `@rwd_action` and `@rwd_window` are set to "main". Method `main` isn't called, because it simply doesn't exist, and `<window name="main">` is extracted from the file `rawdcalc.rwd`. During the rendering of this window, `%result%` is replaced by nothing, because there is no `@result` yet. After entering the numbers and pushing the button "Multiply", method `multiply` is called, because that is the action of that button. `@result` is set to "3*4=12". Because `@rwd_window` is not changed, the last window ("main") will be rendered once more. This time `%result%` will be replaced by `@result` ("3*4=12").

4.3. Panels

Get [rawdpanels.rbw](#) or [rawdpanels.tar.gz](#). You can run it without installing RubyWebDialogs.

```
$ ruby rwdpanels.rbw
```

4. Examples



```
require "ev/rwd"

RWD = <<EOF
<application>

  <window name="main" title="Pannels">
    <panel>
      <pre>&lt;panel&gt;</pre>
    </panel>
    <panel level='high'>
      <pre>&lt;panel level='high'&gt;</pre>
    </panel>
    <panel level='low'>
      <pre>&lt;panel level='low'&gt;</pre>
    </panel>
  </window>

</application>
EOF

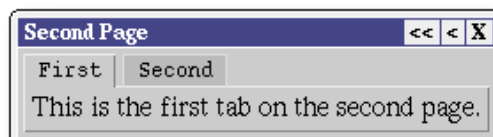
class TestPannels < RWDDialog
end

TestPannels.new(RWD).serve
```

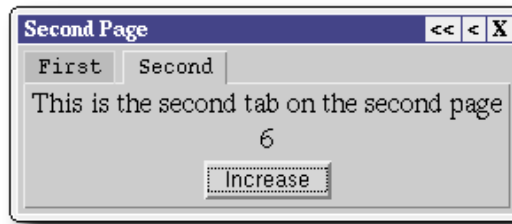
4.4. Tabs

Get [rwdtabs.rbw](#) or [rwdtabs.tar.gz](#) . You can run it without installing RubyWebDialogs.

```
$ ruby rwdtabs.rbw
```



4. Examples



(Both Konqueror and Opera don't like these tabs. I use `<td width='100%>`, which makes them use the full width of the screen. I need to look for a better way.)

```
require "ev/rwd"

RWD = <<EOF
<application>

  <window name="main" title="First Page">
    <p>This is the first page.</p>
    <button caption="Next" action="second"/>
  </window>

  <window name="second" title="Second Page">
    <tabs>
      <tab name="one" caption="First">
        <p>This is the first tab on the second page.</p>
      </tab>
      <tab name="two" caption="Second">
        <p>This is the second tab on the second page</p>
        <p>%counter%</p>
        <button caption="Increase" action="increase"/>
      </tab>
    </tabs>
  </window>

</application>
EOF

class TestTabs < RWDDialog
  def main
    @counter = 1
  end

  def second
    @rwd_window = "second"
  end

  def increase
    @counter += 1
  end
end

TestTabs.new(RWD).serve
```

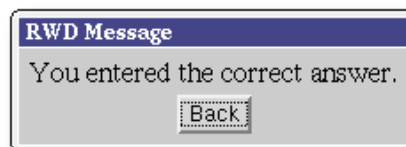
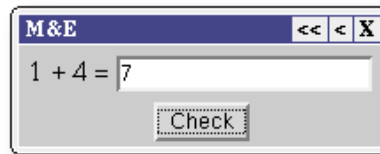
4.5. message and error

Get [rwdmessageanderror.rbw](#) or [rwdmessageanderror.tar.gz](#) . You can run it without installing

4. Examples

RubyWebDialogs.

```
$ ruby rwdmessageanderror.rbw
```



File messageanderror.rwd:

```
<application>

  <window name="main" title="M&E">
    <horizontal>
      <p>%a% + %b% =</p>
      <text name="c"/>
    </horizontal>
    <button caption="Check" action="answer"/>
  </window>

</application>
```

File messageanderror.rbw:

```
require "ev/rwd"

class MessageAndError < RWDDialog
  def main
    ask
  end
end
```


4. Examples

```
def ask
  @a = rand(10)+1
  @b = rand(10)+1
  @c = ""
end

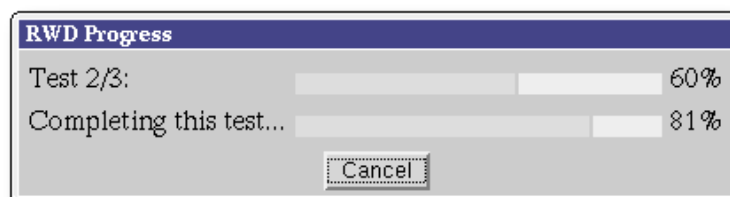
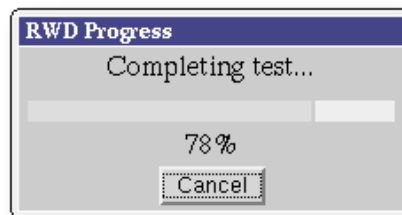
def answer
  if @a + @b == @c.to_i
    message("You entered the correct answer.")
    ask
  else
    error("<br/>You're answer wasn't correct.<br/>Try again.")
  end
end

MessageAndError.file("messageanderror.rwd").serve
```

4.6. Progress Bar

Get [rwdprogressbar.rbw](#) or [rwdprogressbar.tar.gz](#) . You can run it without installing RubyWebDialogs.

```
$ ruby rwdprogressbar.rbw
```



4.7. download

Get [rwddownload.rbw](#) or [rwddownload.tar.gz](#) . You can run it without installing RubyWebDialogs.

```
$ ruby rwddownload.rbw
```

4. Examples



File download.rwd:

```
<application>
  <window name="main" title="Download">
    <a action="test">Download</a>
    <button caption="Download" action="test"/>
  </window>
</application>
```

File download.rbw:

```
require "ev/rwd"

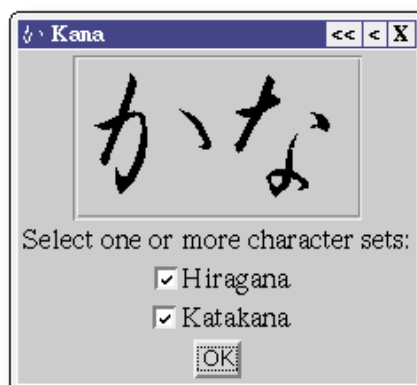
class Demo < RWDDialog
  def test
    download("This is binary test data.", "test.bin")
  end
end

Demo.file("rwdcalc.rwd").serve
```

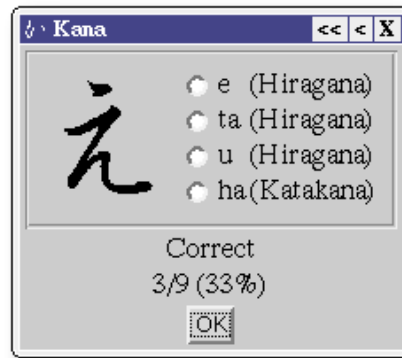
4.8. Japanese

Get [rwdkana.rbw](#) or [rwdkana.tar.gz](#) . You can run it without installing RubyWebDialogs.

```
$ ruby rwdkana.rbw
```



4. Examples



4.9. An Address Book

Get [rwdtsv.rbw](#) or [rwdtsv.tar.gz](#) . You can run it without installing RubyWebDialogs.

```
$ ruby rwdtsv.rbw
```

This is not only an example, it's a real-world application. I use it on a couple of sites to update small tables (tab separated files), e.g. my personal address book. In fact, I've created a `rwdtsv.exe` with [RubyScript2Exe](#). You can do it yourself with:

```
ruby rubyscript2exe.rb rwdtsv.rbw --rubyscript2exe-rubyw --rwd-exit
```

Attach this `rwdtsv.exe` to the extension `tsv` and double-clicking on `adres.tsv` will do the trick. Looks really nice. No DOS box.

```
Usage: ruby rwdtsv.rb file.tsv [port] [-r]
```

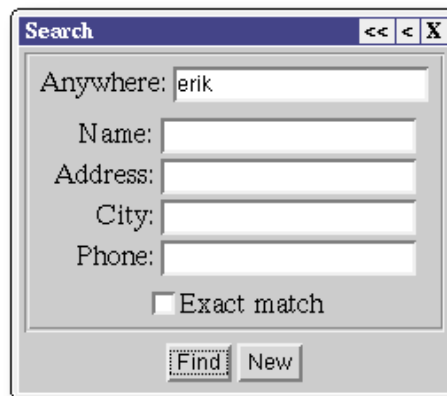
```
-r      Remote mode. User/password combinations are read
        from ~/.rwduids .
```

```
If file.tsv is omitted, a dummy database is connected.
This dummy database won't be saved between two runs.
```

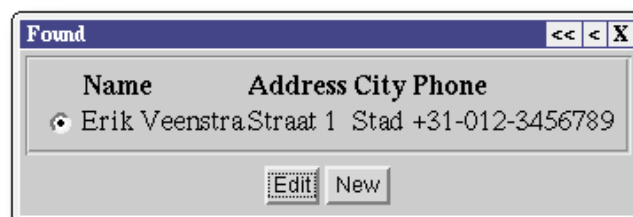
The *file.tsv* of this demo looks like this (replace "\t" with real tabs):

```
# 1 Name
# 2 Address
# 3 City
# 4 Phone
# KEY 1
# SEP
Erik Veenstra\tStraat 1\tStad\t+31-012-3456789
```

4. Examples



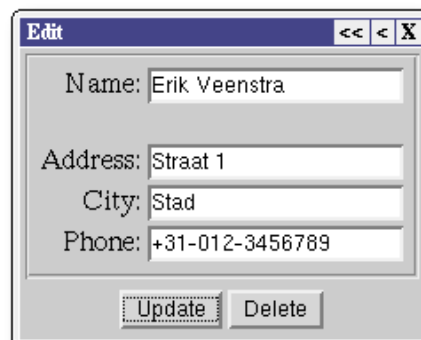
A search dialog box titled "Search" with a close button (X) and navigation buttons (back, forward). It contains four text input fields labeled "Anywhere:", "Name:", "Address:", "City:", and "Phone:". The "Anywhere:" field contains the text "erik". Below the fields is a checkbox labeled "Exact match" which is unchecked. At the bottom are two buttons: "Find" and "New".



A dialog box titled "Found" with a close button (X) and navigation buttons (back, forward). It displays a table with the following data:

Name	Address	City	Phone
Erik Veenstra	Straat 1	Stad	+31-012-3456789

Below the table are two buttons: "Edit" and "New".



An edit dialog box titled "Edit" with a close button (X) and navigation buttons (back, forward). It contains four text input fields labeled "Name:", "Address:", "City:", and "Phone:". The fields contain the following text: "Erik Veenstra", "Straat 1", "Stad", and "+31-012-3456789". Below the fields are two buttons: "Update" and "Delete".



A small dialog box titled "Updated" with a close button (X) and navigation buttons (back, forward). It contains the text "Erik Veenstra' updated." and an "OK" button.

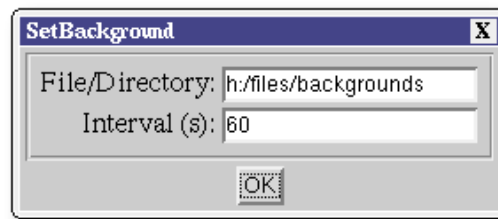
4.10. SetBackground

It's for Windows. Especially for those stupid sites where you can't change the background... Well, now you can...

Get [rwdsetbackground.rbw](#) or [rwdsetbackground.tar.gz](#) . You can run it without installing RubyWebDialogs.

```
$ ruby rwdsetbackground.rbw
```

4. Examples



5. License

RubyWebDialogs, Copyright (C) 2003 Erik Veenstra <rubywebdialogs@erikveen.dds.nl>

This program is free software; you can redistribute it and/or modify it under the terms of the *GNU General Public License (GPL)*, version 2 or the *GNU Lesser General Public License (LGPL)*, version 2.1, as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, **but without any warranty**; without even the implied warranty of **merchantability** or **fitness for a particular purpose**. See the *GNU General Public License (GPL)* or the *GNU Lesser General Public License (LGPL)* for more details.

You should have received a copy of the *GNU General Public License (GPL)* or the *GNU Lesser General Public License (LGPL)* along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

The full text of both licenses can be found [here](#) and [here](#).

6. Download

Current version is 0.2.0 (04.06.2005). It's an alpha release, but very usable. I say alpha, because some API's may change in the future.

Tested on:

- Red Hat Linux 8.0 with Ruby 1.6.7
- Red Hat Linux 8.0 with Ruby 1.8.1
- Red Hat Linux 8.0 with Ruby 1.8.2
- Debian (Testing) with Ruby 1.8.2
- Windows 95 with Ruby 1.8
- Windows 98 with Ruby 1.8
- Windows 2000 with Ruby 1.8
- Windows 2000 with Ruby 1.8 (Cygwin)
- Windows XP with Ruby 1.8
- Windows XP with Ruby 1.8 (Cygwin)
- Mac OS X 10.4.2 with Ruby 1.8

With [rubywebdialogs.rb](#) or [rubywebdialogs.tar.gz](#), you can install the current version into the site_lib directory ev/. That's where all my Ruby libraries go. Just delete it if you want to get rid of all my stuff. RubyWebDialogs is available as [rubywebdialogs.gem](#) as well.

Did you know that you can use RubyWebDialogs without installing it? This is the trick: You can do a `"require 'rubywebdialogs'"` without installing instead of doing a `"require 'ev/rwd'"` after installing. Thus, this file is both a library file and an installation file at the same time! This is exactly the way you would collect dependencies in the Java world with JAR files. You do not extract a JAR file before using its contents.

If you installed RubyWebDialogs, you have to do a `"require 'ev/rwd'"`. If want to do a `"require 'rubywebdialogs'"` directly, you have to copy it to the current directory (or place it in the site_lib directory manually) and do a `require "rubywebdialogs"`.

Send me reports of all bugs and glitches you find. Propositions for enhancements are welcome, too. This helps *us* to make *our* software better.

A change log and older versions can be found [here](#). A generated log file can be found [here](#).

RubyWebDialogs is available on [SourceForge.net](#) and on [RubyForge](#) .

7. Known Issues

- On Darwin, you might run into a `Too many open files - getcwd (Errno::EMFILE)`. I've no solution for this. As far as I know, this is not directly related to `RubyWebDialogs` or `Tar2Rubyscript`. It's an OS kind of thing. Run `ulimit -n` to show the maximum number of open files. It is 256 on Mac OS X. Run `ulimit -n3000` (for example) to fix it.
- A `SecurityError` exception is thrown on my box when starting the browser. This bug (?) is introduced in Ruby 1.8.3. Ruby 1.8.2 worked fine. It's probably caused by `$SAFE=2`, which is done in `ev/rwd.rb`. I'll reconsider this.