Client-server Web applications with Ocsigen

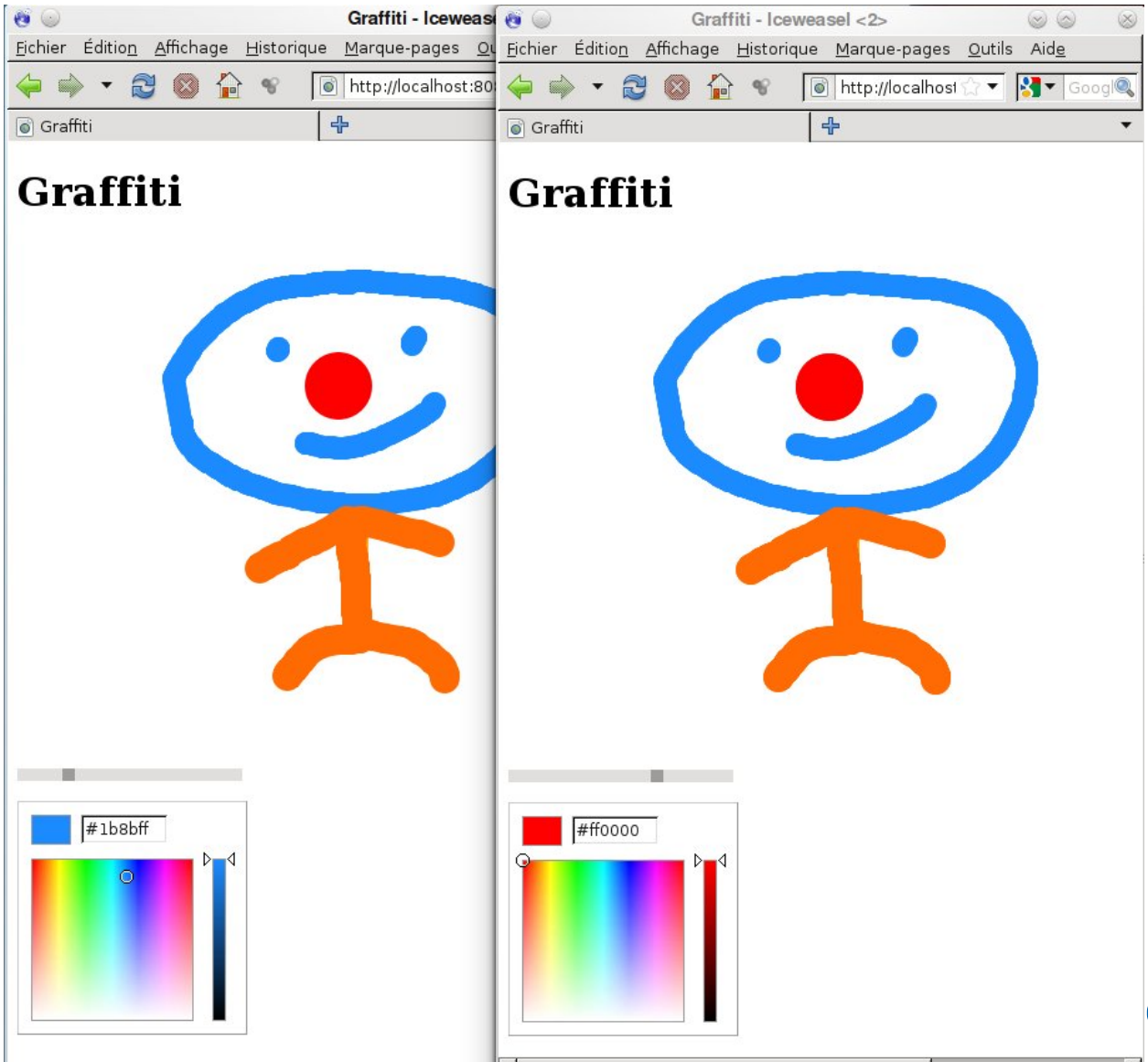
Vincent Balat
Pierre Chambart, Grégoire Henry

WWW2012 Dev Track — April 20th, 2012

Example:
a drawing application

ocsigen
fresh air in web programming

Example:
a collaborative drawing application



csigen
fresh air in web programming

The code

The full *Graffiti* program

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}


let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

     Lwt.return
          << <html> <head> <title>Graffiti</title>
                           <link rel="stylesheet" href="./css/style.css"/>
                           <script src="./oclosure.js"></script>
                    </head>
                    <body> <h1>Graffiti</h1> </body>
             </html> >>
```

The code

The full *Graffiti* program

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}


let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

     Lwt.return
          << <html> <head> <title>Graffiti</title>
                           <link rel="stylesheet" href="./css/style.css"/>
                           <script src="./oclosure.js"></script>
                    </head>
                    <body> <h1>Graffiti</h1> </body>
             </html> >>
```

- short

Our goals

Expressiveness

Our goals

Expressiveness

- Reflexion on concepts
- Analysis of common behaviours
- Semantic view rather than technological

Our goals

Expressiveness

- Reflexion on concepts
- Analysis of common behaviours
- Semantic view rather than technological

Reliability

Our goals

Expressiveness

- Reflexion on concepts
- Analysis of common behaviours
- Semantic view rather than technological

Reliability

- Take in charge many security issues
- Reduce bugs by sophisticated static typing

$\Rightarrow$ The compiler helps you to write good code

$\Rightarrow$ Easy maintenance and evolution

The language

OCaml

- Very expressive: functional, object oriented, parametrized modules …
- Very rich type system (static typing!)
- Compiled language (fast)
- Extensible syntax
- Rich set of libraries and bindings
- Community of users, industrial users

Client and server code

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}


let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
          << <html> <head> <title>Graffiti</title>
                           <link rel="stylesheet" href="./css/style.css"/>
                           <script src="./oclosure.js"></script>
                    </head>
                    <body> <h1>Graffiti</h1> </body>
             </html> >>
```

- One code

Client and server code

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
         << <html> <head> <title>Graffiti</title>
                          <link rel="stylesheet" href="./css/style.css"/>
                          <script src="./oclosure.js"></script>
                   </head>
                   <body> <h1>Graffiti</h1> </body>
            </html> >>
```

ocsigen
fresh air in web programming

Client and server code

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
         << <html> <head> <title>Graffiti</title>
                          <link rel="stylesheet" href="./css/style.css"/>
                          <script src="./oclosure.js"></script>
                  </head>
                  <body> <h1>Graffiti</h1> </body>
            </html> >>
```

{{ ... }}

ocsigen
fresh air in web programming

Client and server code

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       << <html> <head> <title>Graffiti</title>
                        <link rel="stylesheet" href="./css/style.css"/>
                        <script src="./oclosure.js"></script>
                 </head>
                 <body> <h1>Graffiti</h1> </body>
          </html> >>
```

```
{{ ... }}
```

compiled to JS

ocsigen
fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
         << <html> <head> <title>Graffiti</title>
                          <link rel="stylesheet" href="./css/style.css"/>
                          <script src="./oclosure.js"></script>
                  </head>
                  <body> <h1>Graffiti</h1> </body>
            </html> >>
```

open, constants, type

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
         << <html> <head> <title>Graffiti</title>
                          <link rel="stylesheet" href="./css/style.css"/>
                          <script src="./oclosure.js"></script>
                   </head>
                   <body> <h1>Graffiti</h1> </body>
            </html> >>
```

open, constants, type

Initializing the application

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
         << <html> <head> <title>Graffiti</title>
                          <link rel="stylesheet" href="./css/style.css"/>
                          <script src="./oclosure.js"></script>
                  </head>
                  <body> <h1>Graffiti</h1> </body>
            </html> >>
```

open, constants, type

Initializing the application Bus

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
         << <html> <head> <title>Graffiti</title>
                          <link rel="stylesheet" href="./css/style.css"/>
                          <script src="./oclosure.js"></script>
                  </head>
                  <body> <h1>Graffiti</h1> </body>
            </html> >>
```

open, constants, type

Initializing the application Bus

draw function

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       << <html> <head> <title>Graffiti</title>
                        <link rel="stylesheet" href="./css/style.css"/>
                        <script src="./oclosure.js"></script>
                 </head>
                 <body> <h1>Graffiti</h1> </body>
          </html> >>
```

open, constants, type

Initializing the application Bus

draw function

Service

ocsigen

fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       << <html> <head> <title>Graffiti</title>
                        <link rel="stylesheet" href="./css/style.css"/>
                        <script src="./oclosure.js"></script>
                 </head>
                 <body> <h1>Graffiti</h1> </body>
          </html> >>
```

HTML5 page

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       << <html> <head> <title>Graffiti</title>
                        <link rel="stylesheet" href="./css/style.css"/>
                        <script src="./oclosure.js"></script>
                 </head>
                 <body> <h1>Graffiti</h1> </body>
          </html> >>
```

Page specific code

HTML5 page

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       << <html> <head> <title>Graffiti</title>
                        <link rel="stylesheet" href="./css/style.css"/>
                        <script src="./oclosure.js"></script>
                 </head>
                 <body> <h1>Graffiti</h1> </body>
          </html> >>
```

Canvas

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
         << <html> <head> <title>Graffiti</title>
                          <link rel="stylesheet" href="./css/style.css"/>
                          <script src="./oclosure.js"></script>
                  </head>
                  <body> <h1>Graffiti</h1> </body>
            </html> >>
```

Canvas

Slider

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       << <html> <head> <title>Graffiti</title>
                        <link rel="stylesheet" href="./css/style.css"/>
                        <script src="./oclosure.js"></script>
                 </head>
                 <body> <h1>Graffiti</h1> </body>
          </html> >>
```

Canvas

Slider

Color picker

ocsigen

fresh air in web programming

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       << <html> <head> <title>Graffiti</title>
                        <link rel="stylesheet" href="./css/style.css"/>
                        <script src="./oclosure.js"></script>
                 </head>
                 <body> <h1>Graffiti</h1> </body>
          </html> >>
```

Canvas

Slider
Color picker

Function
to compute coordinates
and send them to the server

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       << <html> <head> <title>Graffiti</title>
                        <link rel="stylesheet" href="./css/style.css"/>
                        <script src="./oclosure.js"></script>
                 </head>
                 <body> <h1>Graffiti</h1> </body>
          </html> >>
```

Canvas

Slider
Color picker

Function
to compute coordinates
and send them to the server

Reacting to server events

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

    Lwt.return
      << <html> <head> <title>Graffiti</title>
                       <link rel="stylesheet" href="./css/style.css"/>
                       <script src="./oclosure.js"></script>
                </head>
                <body> <h1>Graffiti</h1> </body>
         </html> >>
```

Canvas

Slider

Color picker

Function
to compute coordinates
and send them to the server

Reacting to server events

Mouse events

Structure

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       << <html> <head> <title>Graffiti</title>
                        <link rel="stylesheet" href="./css/style.css"/>
                        <script src="./oclosure.js"></script>
                 </head>
                 <body> <h1>Graffiti</h1> </body>
          </html> >>
```

open, constants, type

Initializing the application
Bus

draw function

Service

Canvas

Slider
Color picker

Function
to compute coordinates
and send them to the server

Reacting to server events
Mouse events

HTML5 page

ocsigen
fresh air in web programming


Service based Web programming
Generating valid HTML
Client/server Web applications
Sessions, scope of server side state
Mixing Ocsigen apps with traditional Web interaction

Services

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
         << <html> <head> <title>Graffiti</title>
                          <link rel="stylesheet" href="./css/style.css"/>
                          <script src="./oclosure.js"></script>
                  </head>
                  <body> <h1>Graffiti</h1> </body>
            </html> >>
```

```
let main_service = register_service
                     ~path:[""]
                     ~get_params:unit
                     (fun () () -> ... )
```

Services

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

    Lwt.return
      << <html> <head> <title>Graffiti</title>
                       <link rel="stylesheet" href="./css/style.css"/>
                       <script src="./oclosure.js"></script>
                </head>
                <body> <h1>Graffiti</h1> </body>
         </html> >>
```

```
let main_service = register_service
                     ~path:[""]
                     ~get_params:unit
                     (fun () () -> ... )
```

- HTML
- File
- Redirection
- Action
- Application
- ...

ocsigen
fresh air in web programming

Service based Web programming

Powerful service identification mechanism

→ *Precise and straightforward implementation of traditional Web interaction*

Services are first class values

→ *No broken links!*

Typing of services parameters

→ *Conformance of links/forms w.r.t. services!*

→ *Automatic translation to OCaml types*

integers, booleans, but even lists or sets.

Dynamic creation of services

→ *Services customized for one operation by one user*

- Service based Web programming
- Generating valid HTML
- Client/server Web applications
- Sessions, scope of server side state
- Mixing Ocsigen apps with traditional Web interaction

HTML generation

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
     }};

      Lwt.return
        << <html> <head> <title>Graffiti</title>
                         <link rel="stylesheet" href="./css/style.css"/>
                         <script src="./oclosure.js"></script>
                  </head>
                  <body> <h1>Graffiti</h1> </body>
           </html> >>
```

ocsigen
fresh air in web programming

HTML generation

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload
    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                     >>> first [mousemoves Dom_html.document (arr line);
                                mouseup Dom_html.document >>> (arr line)])) ());
    }};

    Lwt.return
      << <html> <head> <title>Graffiti</title>
                       <link rel="stylesheet" href="./css/style.css"/>
                       <script src="./oclosure.js"></script>
                </head>
                <body> <h1>Graffiti</h1> </body>
        </html> >>
```

```
<< <html> <head> <title>Graffiti</title>
                 <link rel="stylesheet" href="./css/sty
                 <script src="./oclosure.js"></script>
          </head>
          <body> <h1>Graffiti</h1> </body>
  </html> >>
```

ocsigen
fresh air in web programming

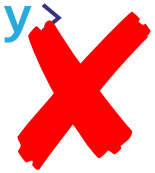
HTML generation

HTML validity checked at compile time!

Producing valid HTML

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</html>
```

```
<html>
  <head><title>Hello</title></head>
  <body><h1>Hello</h1></body>
</hmtl>
```



→ rejected *at compile time!*

```
<html>
  <head><title>Hello</title></head>
  <body><title>Hello</title></body>
</html>
```

Producing valid HTML

Any program that may generate wrong pages will be rejected by the compiler!

$f : () \rightarrow$ block list

<body> $f()$ </body>

 $f()$

- Service based Web programming
- Generating valid HTML
- Client/server Web applications
- Sessions, scope of server side state
- Mixing Ocsigen apps with traditional Web interaction

Client/server applications

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
       << <html> <head> <title>Graffiti</title>
                        <link rel="stylesheet" href="./css/style.css"/>
                        <script src="./oclosure.js"></script>
                 </head>
                 <body> <h1>Graffiti</h1> </body>
          </html> >>
```

```
{{ ... }}
```

compiled to JS

ocsigen
fresh air in web programming

The bus

```
let b =
  Eliom_bus.create ~name:"graff" Json.t<messages>
```

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_mod...(
  struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

    Lwt.return
      << <html> <head> <title>Graffiti</title>
                       <link rel="stylesheet" href="./css/style.css"/>
                       <script src="./oclosure.js"></script>
                </head>
                <body> <h1>Graffiti</h1> </body>
         </html> >>
```

ocsigen

fresh air in web programming

The bus

```
let b =
  Eliom_bus.create ~name:"graff" Json.t<messages>
```

```
Eliom_bus.write %b v
```


ocsigen
fresh air in web programming

The bus

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_mod...(struct
  let application_name = "graffiti"
end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

    Lwt.return
      << <html> <head> <title>Graffiti</title>
                       <link rel="stylesheet" href="./css/style.css"/>
                       <script src="./oclosure.js"></script>
                </head>
                <body> <h1>Graffiti</h1> </body>
         </html> >>
```

```
let b =
  Eliom_bus.create ~name:"graff" Json.t<messages>
```

```
Eliom_bus.write %b v
```

```
Lwt_stream.iter (draw ctx) (Eliom_bus.stream %b)
```

ocsigen
fresh air in web programming

Mouse events

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload

    {{ let canvas = Dom_html.createCanvas Dom_html.document in
       let ctx = canvas##getContext (Dom_html._2d_) in
       canvas##width <- width; canvas##height <- height;
       ctx##lineCap <- Js.string "round";
       Dom.appendChild Dom_html.document##body canvas;

       let slider = jsnew Goog.Ui.slider(Js.null) in
       slider##setMinimum(1.); slider##setMaximum(80.);
       slider##render(Js.some Dom_html.document##body);

       let pSmall = jsnew Goog.Ui.hsvPalette
         (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
       pSmall##render(Js.some Dom_html.document##body);

       let x = ref 0 and y = ref 0 in
       let set_coord ev =
         let x0, y0 = Dom_html.elementClientPosition canvas in
         x := ev##clientX - x0; y := ev##clientY - y0 in
       let compute_line ev =
         let oldx = !x and oldy = !y in
         set_coord ev;
         let color = Js.to_string (pSmall##getColor()) in
         let size = int_of_float (Js.to_float (slider##getValue())) in
         (color, size, (oldx, oldy), (!x, !y))
       in
       let (b : messages Eliom_bus.t) = %b in
       let line ev =
         let v = compute_line ev in
         let _ = Eliom_bus.write b v in
         draw ctx v
       in
       ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
       ignore (run (mousedowns canvas
                    (arr (fun ev -> set_coord ev; line ev)
                    >>> first [mousemoves Dom_html.document (arr line);
                               mouseup Dom_html.document >>> (arr line)])) ());
    }};

     Lwt.return
         << <html> <head> <title>Graffiti</title>
                          <link rel="stylesheet" href="./css/style.css"/>
                          <script src="./oclosure.js"></script>
                  </head>
                  <body> <h1>Graffiti</h1> </body>
            </html> >>
```

Mouse events

```
{shared{
  open Eliom_pervasives
  open HTML5.M
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}}

module My_appl = Eliom_output.Eliom_appl (struct
    let application_name = "graffiti"
  end)

let b = Eliom_bus.create ~name:"graff" Json.t<messages>

{client{
  open Event_arrows
  let draw ctx (color, size, (x1, y1), (x2, y2)) =
    ctx##strokeStyle <- (Js.string color);
    ctx##lineWidth <- float size;
    ctx##beginPath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke();
}}

let main_service = My_appl.register_service ~path:[""] ~get_params:Eliom_parameters.unit
  (fun () () -> Eliom_services.onload {{

    let canvas = Dom_html.createCanvas Dom_html.document in
    let ctx = canvas##getContext (Dom_html._2d_) in
    canvas##width <- width; canvas##height <- height;
    ctx##lineCap <- Js.string "round";
    Dom.appendChild Dom_html.document##body canvas;

    let slider = jsnew Goog.Ui.slider(Js.null) in
    slider##setMinimum(1.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);

    let pSmall = jsnew Goog.Ui.hsvPalette
      (Js.null, Js.null, Js.some (Js.string "goog-hsv-palette-sm")) in
    pSmall##render(Js.some Dom_html.document##body);

    let x = ref 0 and y = ref 0 in
    let set_coord ev =
      let x0, y0 = Dom_html.elementClientPosition canvas in
      x := ev##clientX - x0; y := ev##clientY - y0 in
    let compute_line ev =
      let oldx = !x and oldy = !y in
      set_coord ev;
      let color = Js.to_string (pSmall##getColor()) in
      let size = int_of_float (Js.to_float (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Eliom_bus.t) = %b in
    let line ev =
      let v = compute_line ev in
      let _ = Eliom_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
    ignore (run (mousedowns canvas
                  (arr (fun ev -> set_coord ev; line ev)
                  >>> first [mousemoves Dom_html.document (arr line);
                             mouseup Dom_html.document >>> (arr line)])) ());
  }};

  Lwt.return
    << <html> <head> <title>Graffiti</title>
                     <link rel="stylesheet" href="./css/style.css"/>
                     <script src="./oclosure.js"></script>
              </head>
              <body> <h1>Graffiti</h1> </body>
       </html> >>
```

```
run (mousedowns canvas
       (arr set_coord >>>
          first [mousemoves document (arr line);
                 mouseup document >>> (arr line)])) ()
```

We use *arrows*


Service based Web programming
Generating valid HTML
Client/server Web applications
Sessions, scope of server side state
Mixing Ocsigen apps with traditional Web interaction

Sessions revisited

Session data saved in *references* with a **scope**.

Scopes:

- Site
- Session group (user)
- Browser session (cookie)
- Client side process (tab)
- Request

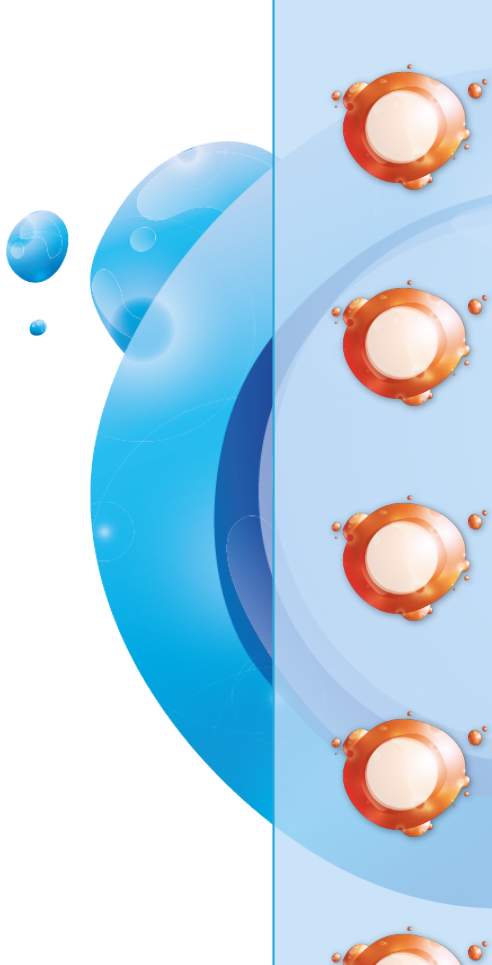
Sessions revisited

Server side state implemented as *references* with a **scope**.

Scopes:

- Site
- Session group (user)
- Browser session (cookie)
- Client side process (tab)
- Request

Services also have a scope.



- Service based Web programming
- Generating valid HTML
- Client/server Web applications
- Sessions, scope of server side state
- Mixing Ocsigen apps with traditional Web interaction

Web 1.0 + Web 2.0 = ?

Web 1.0 + Web 2.0 = ?

The client side program does not stop when you click a link!

Web 1.0 + Web 2.0 = ?

The client side program does not stop when you click a link!

⟶ Ocsigen client/server Web applications are fully compatible with traditional Web interaction (bookmarks, back button)!

⟶ You can keep a state on client side.

⟶ Parts of the page can persist after loading a new page.

⟶ Music/video does not stop.

Example: Music streaming Website

(go to another album without stopping the music)

Conclusion

Many unique features

Many unique features

Dynamic services

Persistance of client side program

Sophisticated service identification

HTML typing

Unified client/server programming

Session services

Scopes

Typing of links, forms, parameters

ocsigen
fresh air in web programming

Users

A growing community

Some small companies:

BeSport, Cowebo, Hypios, Ocamlcore, Ocamlpro, Baoug, Nleyten …

The project

ocsigen.org

Free/open source software --- Version 2 released in 2011.

Authors and contributors:

Vincent Balat, Jérôme Vouillon, Pierre Chambart, Grégoire Henry, Raphaël Proust, Benjamin Canou, Boris Yakobowski, Jérémie Dimino, Benedikt Becker

Séverine Maingaud, Stéphane Glondu, Gabriel Kerneis, Denis Berthod, Gabriel Cardoso, Piero Furiesi, Jaap Boender, Thorsten Ohl, Gabriel Scherer, Simon Castellan, Jean-Henri Granarolo, Archibald Pontier, Nataliya Guts, Cécile Herbelin, Charles Oran, Jérôme Velleine, Pierre Clairambault …

ocsigen
fresh air in web programming