

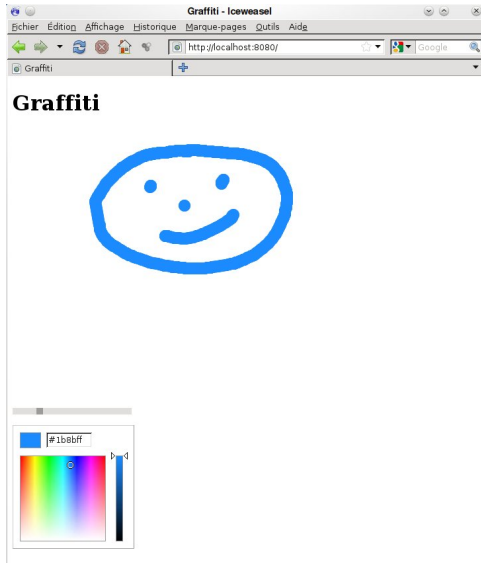


# **Client-server Web applications with Ocsigen**

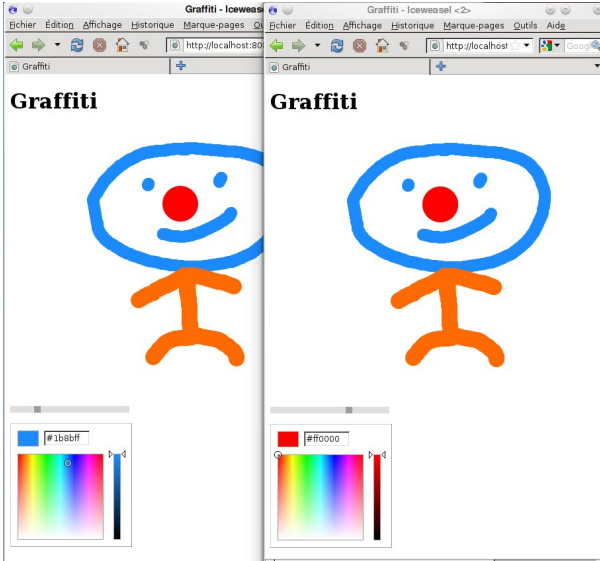
**Vincent Balat  
Pierre Chambart, Grégoire Henry**

**WWW2012 Dev Track — April 20th, 2012**

# Example: a drawing application



# Example: a collaborative drawing application



## 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-se")) 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.toFloat (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 e = Eliom_bus.write b v in
      draw ctx v
    in
    ignore (let_stream.iter (draw ctx) (Eliom_bus.stream b));
    ignore (run (mousedown canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemove 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_Streams
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-se")) 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.toFloat (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 e = Eliom_bus.write b v in
      draw ctx v
    in
    ignore (let_stream.iter (draw ctx) (Eliom_bus.stream b));
    ignore (run (mousedown canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemove 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

*Expressiveness*

## *Expressiveness*

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

## *Expressiveness*

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

## *Reliability*



## *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
  - ⇒ The compiler helps you to write good code
  - ⇒ Easy maintenance and evolution



## 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 Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "Graffiti"
end)

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

(client
  open Event_handlers
  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:Elion_parameters.unit
(fun () -> Elion_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(x.); 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-se")) 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.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let e = Elion_bus.write b v in
      draw ctx v
    in
    ignore (let_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedown canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousemove 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 Elion_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "Graffiti"
end)

let b = Elion_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:Elion_parameters.unit
(fun () () -> Elion_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-se")) 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.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_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> >>
```

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

# Client and server code

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_app1 = Elion_output.Elion_app1 (struct
  let application_name = "Graffiti"
end)

let b = Elion_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_app1.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () -> Elion_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-se")) 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.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_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

# Structure

open, constants, type

```
{shared}
open Elion_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_app1 = Elion_output.Elion_app1 (struct
  let application_name = "Graffiti"
end)

let b = Elion_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_app1.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () -> Elion_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-se")) 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.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (let_stream.iter (draw ctx) (Elion_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> >>
```

# Structure

open, constants, type

Initializing the application

```
{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:"groff" 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-se")) 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.toFloat (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 (let_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> >>
```



# Structure

open, constants, type

Initializing the application  
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_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-se")) 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.toFloat (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 (let_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> >>
```

# Structure

open, constants, type

Initializing the application  
Bus

draw function

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)

let b = Elion_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##strokeWidth <- 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:Elion_parameters.unit
(fun () -> Elion_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-se")) 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.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (let_stream.iter (draw ctx) (Elion_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)]))) ());
  ));

let 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> >>
```

# Structure

```
{shared}
open Elion_pervasives
open HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "Graffiti"
end)

let b = Elion_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##strokeWidth <- 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:Elion_parameters.unit
(fun () -> Elion_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-se")) 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.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = Mb in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_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

# Structure

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_app1 = Elion_output.Elion_app1 (struct
  let application_name = "Graffiti"
end)

let b = Elion_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_app1.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () -> Elion_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-se")) 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.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (let_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousedownes 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="/.closure.js"></script>
  </head>
  <body> <h1>Graffiti</h1> </body>
</html >>
```

HTML5 page

# Structure

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "Graffiti"
end)

let b = Elion_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:Elion_parameters.unit
(fun () -> Elion_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-se")) 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 Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (let_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedowns canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousedowns Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)]))) ());
  )
));

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

Page specific code

HTML5 page

# Structure

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "Graffiti"
end)

let b = Elion_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:Elion_parameters.unit
(fun () {} -> Elion_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-se")) 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.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (let_stream.iter (draw ctx) (Elion_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 Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_appl = Elion_output.Elion_appl (struct
  let application_name = "Graffiti"
end)

let b = Elion_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:Elion_parameters.unit
{fun () -> Elion_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-se")) 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.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (let_stream.iter (draw ctx) (Elion_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 Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_app1 = Elion_output.Elion_app1 (struct
  let application_name = "Graffiti"
end)

let b = Elion_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_app1.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () -> Elion_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-se"))] 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 Elion_bus.t) = b in
    let line ev =
      let v = compute_line ev in
      let _ = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_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}
type Elion_pervasives
name HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (IJson)
}

module My_app1 = Elion_output.Elion_app1 (struct
  let application_name = "Graffiti"
end)

let b = Elion_bus.create ~name:"graff" IJson.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_app1.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () -> Elion_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-se"))] 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 Elion_bus.t) = b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_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}
type Elion_pervasives
name HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (IJson)
}

module My_app1 = Elion_output.Elion_app1 (struct
  let application_name = "Graffiti"
end)

let b = Elion_bus.create ~name:"graff" IJson.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_app1.register_service ~path:[""] ~get_params:Elion_parameters.unit
{fun () -> Elion_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-se"))] 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 Elion_bus.t) = b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mouseevents 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 Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Ison)
}
```

open, constants, type

```
module My_appl = Elion_output.Elion_appl (struct
  let application_name = "graffiti"
end)
```

Initializing the application  
Bus

```
let b = Elion_bus.create ~name:"graff" Ison.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()
  )
}
```

draw function

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

Service

```
(( 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-se")) 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 Elion_bus.t) = %b in
  let line ev =
    let v = compute_line ev in
    let = Elion_bus.write b v in
    draw ctx v
  in
  ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
  ignore (run (mouseevents canvas
    (arr (fun ev -> set_coord ev; line ev)
      >>> first [mousemoves Dom_html.document (arr line);
        mouseup Dom_html.document >>> (arr line)])) ());
  ));
```

Canvas

Slider

Color picker

```
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 Elion_bus.t) = %b in
let line ev =
  let v = compute_line ev in
  let = Elion_bus.write b v in
  draw ctx v
in
ignore (Lwt_stream.iter (draw ctx) (Elion_bus.stream b));
ignore (run (mouseevents canvas
  (arr (fun ev -> set_coord ev; line ev)
    >>> first [mousemoves Dom_html.document (arr line);
      mouseup Dom_html.document >>> (arr line)])) ());
  ));
```

Function

to compute coordinates  
and send them to the server

Reacting to server events  
Mouse events

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

HTML5 page



**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}
type Elion_pervasive
name HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}
```

```
module My_app1 = Elion
  let application_name
end
```

```
let b = Elion_bus.create
```

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

```
let main_service = My_app1.register_service ~path:[] ~get_params:Elion_parameters.unit
  (fun () -> Elion_services.onload
```

```
(( let canvas = Dom.html.createCanvas Dom.html.document in
  #getContext (Dom.html_id_3) in
  canvas##width <- width; canvas##height <- height;
  ctx##lineCap <- s.string "round";
  Dom.appendChild Dom.html.document#body canvas;
```

```
let slider = jsnew Goog.UI.slider(Js.null) in
  slider##setMinimum(x.); 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-se")) 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 Elion_bus.t) = b in
let line ev =
  let v = compute_line ev in
  let = Elion_bus.write b v in
  draw ctx v
in
ignore (let_stream.iter (draw ctx) (Elion_bus.stream b));
ignore (run (mousedowns canvas
  (arr (fun ev -> set_coord ev; line ev)
    >>> first [mousedowns Dom.html.document (arr line);
    mouseup Dom.html.document >>> (arr line)])) ());
```

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

let main\_service = register\_service

~path:[]

~get\_params:unit

(fun () () -> ... )

```
{shared}
type Elion_pervasive
name HTML5_M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_app1 = Elion
  let application_name = "Grafitti"
  let b = Elion_bus.create

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

  let main_service = My_app1
  (fun () -> Elion

  (( let canvas = Dom
    canvas##width <-
    canvas##height <- height;
    ctx##lineCap <-
    Dom.appendChild
    let slider = jsnew Goog.UI.slider(Js.null) in
    slider##setMinimum(x.); slider##setMaximum(80.);
    slider##render(Js.some Dom_html.document##body);
    let pSmall = jsnew Goog.UI.hsvPalette
    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 Elion_bus.t) = b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (let_stream.iter (draw ctx) (Elion_bus.stream b));
    ignore (run (mousedown canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousedownes Dom_html.document (arr line);
          mouseup Dom_html.document >>> (arr line)])) ());
  ));

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

let main\_service = register\_service

~path: [ "" ]

~get\_params: unit

(fun () () -> ... )

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

# 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}
name Eliom_pervasives
name HTML5_E
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_app1 = Eliom_output.Eliom_app1 (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_app1.registor_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-se")) 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
    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 (mousedown canvas
    [arr (fun ev -> set_coord ev; line ev)
    >>> first [mousedown 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="/closure.js"></script>
</head>
<body > <h1>Graffiti</h1> </body>
</html >>
```

# HTML generation

```
{shared
  some Eliom_pervasives
  some HTML5_H
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Json)
})

module My_app1 = Eliom_output.Eliom_app1 (struct
  let application_name = "Graffiti"
end)

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

(client)
open Event_arrows
let draw ctx (color,
  ctx##strokeStyle <-
  ctx##lineWidth <-
  ctx##strokeWidth <-
  ctx##moveTo (float x
  ctx##stroke()

let main_service = My_
(fun () -> Eliom_

(( let canvas = Dom
let ctx = canvas
canvas##width <-
ctx##lineCap <-
Dom.appendChild

let slider = jsn
slider##setLine
slider##render()

let pSmall = jsn
[35.null, 35.m
pSmall##render()

let x = ref 0 an
let set_coord ev
let x0, y0 = D
x := ev##client
let compute_line
let oldx = lx
set_coord ev;
let color = 35
let size = int
(color, size,

in
let (b : messages Eliom_bus.t) = 3b 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 (mousedown canvas
  [arr (fun ev -> set_coord ev; line ev)
  >>> first [mousedown 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> >>
```

# 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>
```



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

→ rejected *at compile time!*



# Producing valid HTML

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

$f : () \rightarrow \text{block list}$

`<body> f() </body>`



`<p> f() </p>`





**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_H
  let width = 700
  let height = 400
  type messages = (string * int * (int * int) * (int * int)) deriving (Isom)
})

module My_app1 = Eliom_output.Eliom_app1 (struct
  let application_name = "Graffiti"
end)

let b = Eliom_bus.create ~name:"graff" Isom.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##strokePath();
    ctx##moveTo(float x1, float y1); ctx##lineTo(float x2, float y2);
    ctx##stroke()
  )
)

let main_service = My_app1.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-se")) 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.toFloat (slider##getValue())) in
      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 v = Eliom_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
    ignore (run (mousedown canvas
      (arr (fun ev -> set_coord ev; line ev)
        >>> first [mousedown 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



# The bus

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

module My_appl = struct
  module output_Eliom
    let application_name = "graffiti"
  end

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

{client}
open Event_arrows
let draw ctx (color, size, (x1, y1), (x2, y2)) =
  ctx##strokeStyle <- (Js.string color);
  ctx##strokeWidth <- (Js.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 b =

Eliom\_bus.create ~name:"graff" Json.t<messages>

```
(( let canvas = Dom_html.createCanvas Dom_html.document in
  let ctx = canvas##getContext (Dom_html.id ) 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-se")) 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 [mousedown canvas
    (arr (fun ev -> set_coord ev; line ev)
    >>> first [mousemove 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="/closure.js"></script>
</head>
<body><div>Graffiti</div> </body>
</html >>
```

# The bus

```
{shared|
name Eliom_pervasive
name HTML5.M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Show)
}
```

let b =

Eliom\_bus.create ~name:"graff" Json.t<messages>

```
module My_appl = Etlom_output.Eliom
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._id ) in
canvas##width <- width; canvas##height <- height;
ctx##lineCap <- Js.string "round";
Dom.appendChild Dom.html.document##body canvas;
```

Eliom\_bus.write %b v

```
let slider = jsnew (Js.null) <#> HTML5.Slider
slider##setMinimum(0); slider##setMaximum(100);
slider##render(0); slider##style

let pSmall = jsnew (Js.null) <#> HTML5.Palette
{Js.null, Js.null, Js.null, Js.null} <#> (Js.string "good-hue-palette-in") in
pSmall##render(0); pSmall##style
```

```
let x = ref 0 and y = ref 0 in
let set_coord =
  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 =
  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 [mousedown canvas
  (arr (fun ev -> set_coord ev; line ev)
  >>> first [mousemove 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="/closure.js"></script>
</head>
<body><div>Graffiti</div> </body>
</html >>
```

# The bus

```
{shared}
name Eliom_pervasive
name HTML5.M
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Show)
}
```

let b =

Eliom\_bus.create ~name:"graff" Json.t<messages>

```
module My_appl = Eliom_output.Eliom
  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;
```

Eliom\_bus.write %b v

```
let slider = jsnew (Js.val `HTMLInputElement)
slider##setMinimumValue(0)
slider##render()
let pSmall = jsnew (Js.val `HTMLParagraphElement)
pSmall##render()
let x = ref 0
let y = ref 0
let set_coord =
  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 : message) =
  let v = compute_line ev in
  let w = Js.to_string (pSmall##write %b v) in
  draw ctx (color, size, (oldx, oldy), (x, y))
in
ignore (Lwt_stream.iter (draw ctx) (Eliom_bus.stream b));
ignore (run (mouseoup Dom.html.document >> (arr line))) ());
});
```

Lwt\_stream.iter (draw ctx) (Eliom\_bus.stream %b)

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

# Mouse events

```
{shared}
open Elion_pervasives
open HTML5_H
let width = 700
let height = 400
type messages = (string * int * (int * int) * (int * int)) deriving (Json)
}

module My_app1 = Elion_output.Elion_app1 (struct
  let application_name = "Graffiti"
end)

let b = Elion_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_app1.register_service ~path:[""] ~get_params:Elion_parameters.unit
(fun () -> Elion_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-se")) 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.toFloat (slider##getValue())) in
      (color, size, (oldx, oldy), (!x, !y))
    in
    let (b : messages Elion_bus.t) = !b in
    let line ev =
      let v = compute_line ev in
      let = Elion_bus.write b v in
      draw ctx v
    in
    ignore (Lwt_stream.iter (draw ctx) (Elion_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> >>
```





**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

**Persistence of client side program**

Sophisticated service identification

HTML typing

Unified client/server programming

Session services

**Scopes**

Typing of links, forms, parameters

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 ...