

# Embedded Web Server

Simon Wright  
*simon@pushface.org*

21.xi.13

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Copyright and Licencing . . . . .	1
<b>2</b>	<b>Cyclic updating and select/options</b>	<b>3</b>
2.1	Displaying the current time . . . . .	3
2.2	Changing the time display format . . . . .	5
<b>3</b>	<b>Radio buttons</b>	<b>7</b>
<b>4</b>	<b>Checkboxes</b>	<b>11</b>
<b>5</b>	<b>File upload</b>	<b>13</b>
<b>6</b>	<b>Retrieving the initial state</b>	<b>17</b>
<b>7</b>	<b>Utilities</b>	<b>20</b>
7.1	Notifying the server of changes . . . . .	20
7.2	Set up radio buttons . . . . .	21
<b>8</b>	<b>HTML pages</b>	<b>22</b>
<b>9</b>	<b>JavaScript</b>	<b>25</b>
<b>10</b>	<b>Ada</b>	<b>26</b>
10.1	Ada code . . . . .	26
10.2	GNAT Project . . . . .	27
<b>A</b>	<b>About this document</b>	<b>28</b>
<b>B</b>	<b>Index</b>	<b>29</b>
B.1	Files . . . . .	29
B.2	Macros . . . . .	29
B.3	Definitions . . . . .	30

# List of Figures

1.1	The dynamic part of the example page . . . . .	1
2.1	Displaying the current time, and controlling the format . . . . .	3
3.1	Using radio buttons . . . . .	7
4.1	Using checkboxes . . . . .	11
5.1	The file upload dialog . . . . .	13
5.2	Upload completed . . . . .	13

# Chapter 1

## Introduction

This document describes the [Embedded Web Server](#) (EWS) in the form of a demonstration program.

EWS is intended for small, limited embedded systems (for example, ones with no file system).

It provides a program (`ews-make_htdocs`) which converts a directory structure containing a set of web pages into an Ada data structure to be compiled with the EWS library and your application and served at run time.

As well as static web pages, EWS supports dynamic interactions, where the client makes a request and the server responds. This can be used to provide a complete new page, constructed on the fly by the server, or (more interestingly) in an [AJAX](#) style, where the server takes some action and the response is interpreted by [JavaScript](#) in the client.

Displaying the current date/time (26/08/13) as seen at the server.	
Using select/options	European ▾
Using radio buttons	Forward Light <input type="radio"/> Red <input checked="" type="radio"/> Blue Aft Light <input checked="" type="radio"/> Red <input type="radio"/> Blue
Using checkboxes	Port Lamp <input checked="" type="checkbox"/> Starboard Lamp <input type="checkbox"/>
File upload	Choose File <input type="file"/> ews.ads <input type="button" value="Send"/>

Figure 1.1: The dynamic part of the example page

Figure 1.1 shows the part of the example page which demonstrates AJAX-style interactions.

These interactions are supported by EWS's `HttpInteraction.js`.

### 1.1 Copyright and Licencing

EWS itself is licenced under the [GPL version 3](#); the code that forms part of the run time (Ada and JavaScript) has the additional permissions granted by the [GCC Runtime Library Exception version 3.1](#).

The demonstration code is released without restriction.

`< Copyright 2a > ≡  
 Copyright 2013 Simon Wright <simon@pushface.org>`

◇  
Fragment referenced in [2bcd](#).

In Ada,

`< Ada licence header 2b > ≡  
 -- < Copyright 2a >  
 --  
 -- This unit is free software; you can redistribute it and/or modify  
 -- it as you wish. This unit is distributed in the hope that it will  
 -- be useful, but WITHOUT ANY WARRANTY; without even the implied  
 -- warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.`

◇  
Fragment referenced in [26](#).

In JavaScript,

`< JavaScript licence header 2c > ≡  
 /*  
 * < Copyright 2a >  
 *  
 * This unit is free software; you can redistribute it and/or modify  
 * it as you wish. This unit is distributed in the hope that it will  
 * be useful, but WITHOUT ANY WARRANTY; without even the implied  
 * warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  
 */`

◇  
Fragment referenced in [25](#).

In HTML,

`< HTML licence header 2d > ≡  
 <!--  
 < Copyright 2a >  
  
 This unit is free software; you can redistribute it and/or modify  
 it as you wish. This unit is distributed in the hope that it will  
 be useful, but WITHOUT ANY WARRANTY; without even the implied  
 warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  
 -->`

◇  
Fragment referenced in [22a](#).

## Chapter 2

# Cyclic updating and select/options

This part of the demonstration, shown in Figure 2.1, shows cyclic updating of a portion of the web page (the time, as seen at the server) and the ability to change the format of the display using the HTML ‘select’ and ‘options’.

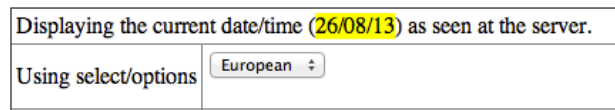


Figure 2.1: Displaying the current time, and controlling the format

```
< Cyclic updating and select/options: HTML 3a > ≡  
  <tr>  
    < Display the current time: HTML 3b >  
  </tr>  
  <tr>  
    < Change the time display format: HTML 5 >  
  </tr>  
  ◊
```

Fragment referenced in 23c.

### 2.1 Displaying the current time

The HTML displays the current time in a span with id `timeDisplay` inside a two-column table data cell.

```
< Display the current time: HTML 3b > ≡  
  <td colspan="2">  
    Displaying the current date/time  
    (<span id="timeDisplay" style="background:yellow">here</span>)  
    as seen at the server.  
  </td>  
  ◊
```

Fragment referenced in 3a.

Defines: `timeDisplay` 4a.

The corresponding JavaScript runs a `CyclicHttpRequest` with a 1-second repetition rate to the URL `ajaxTime`. The response is expected to be a `text/plain` string, which is pasted into the document at the element with id `timeDisplay`.

```

< Cyclic interactions: JavaScript 4a > ≡
  var timeRequest = new CyclicHttpRequest
    ("ajaxTime",
     function (r) {
       document.getElementById("timeDisplay").innerHTML = r.responseText;
     },
     1000);

```

Fragment defined by [4a](#), [9b](#).

Fragment referenced in [25](#).

Defines: `timeRequest` [4b](#).

Uses: `ajaxTime` [4d](#), `timeDisplay` [3b](#).

The cyclic interaction needs to be started when the page is loaded.

```

< Start getting cyclic data: JavaScript 4b > ≡
  timeRequest.start();

```

Fragment defined by [4b](#), [9c](#).

Fragment referenced in [25](#).

Uses: `timeRequest` [4a](#).

The Ada code which receives the cyclic `ajaxTime` request is in the function `AJAX_Time`.

```

< Specs of dynamic pages: Ada 4c > ≡
  function AJAX_Time
    (From_Request : HTTP.Request_P)
  return Dynamic.Dynamic_Response'Class;

```

Fragment defined by [4c](#), [10a](#), [14c](#), [18b](#), [20b](#).

Fragment referenced in [26](#).

Defines: `AJAX_Time` [4d](#).

The function is registered with the server, to be called to respond to the URL `ajaxTime`. We need to use GNAT's implementation-defined attribute `'Unrestricted_Access` because `AJAX_Time` isn't declared at library level; this is unlikely to be a problem in a real program.

```

< Register dynamic pages: Ada 4d > ≡
  Dynamic.Register (AJAX_Time'Unrestricted_Access, "/ajaxTime");

```

Fragment defined by [4d](#), [10b](#), [14d](#), [18c](#), [20c](#).

Fragment referenced in [26](#).

Defines: `ajaxTime` [4a](#).

Uses: `AJAX_Time` [4cf](#).

`AJAX_Time` uses global data to store the time format that is required.

```

< Global data: Ada 4e > ≡
  type Date_Format is (ISO, US, European, Locale);
  Current_Date_Format : Date_Format := ISO;

```

Fragment defined by [4e](#), [8b](#), [12b](#).

Fragment referenced in [26](#).

Defines: `Current_Date_Format` [4f](#), [6b](#), [18d](#), `Date_Format` [6b](#).

The implementation of `AJAX_Time` returns the current date/time as plain text in the format selected in `Current_Date_Format`.

```

< Bodies of dynamic pages: Ada 4f > ≡
  function AJAX_Time
    (From_Request : HTTP.Request_P) return Dynamic.Dynamic_Response'Class is
    Result : Dynamic.Dynamic_Response (From_Request);
    function Format return GNAT.Calendar.Time_IO.Picture_String;
    function Format return GNAT.Calendar.Time_IO.Picture_String is
    begin
      case Current_Date_Format is

```

```

        when ISO => return GNAT.Calendar.Time_IO.ISO_Date;
        when US => return GNAT.Calendar.Time_IO.US_Date;
        when European => return GNAT.Calendar.Time_IO.European_Date;
        when Locale => return "%c";
    end case;
end Format;
begin
    Dynamic.Set_Content_Type (Result, To => Types.Plain);
    Dynamic.Set_Content
        (Result,
         GNAT.Calendar.Time_IO.Image (Ada.Calendar.Clock, Format));
    return Result;
end AJAX_Time;
◇

```

Fragment defined by [4f](#), [10c](#), [14e](#), [18d](#), [20d](#).

Fragment referenced in [26](#).

Defines: [AJAX\\_Time 4d](#).

Uses: [Current\\_Date\\_Format 4e](#).

## 2.2 Changing the time display format

The choice of time format is implemented in HTML in a table data cell containing a form `fTimeFormat` containing a drop-down list, with the value associated with each option being that of the corresponding value of the Ada `Date_Format` (this makes it easy for the Ada code to determine which value has been sent, using `Date_Format'Value`).

*< Change the time display format: HTML 5 >* ≡

```

<td>
    Using select/options
</td>
<td>
    <form method="POST" name="fTimeFormat" id="fTimeFormat">
        <select name="format">
            <option value="iso" selected="true">ISO</option>
            <option value="us">US</option>
            <option value="european">European</option>
            <option value="locale">Local</option>
        </select>
    </form>
</td>
◇

```

Fragment referenced in [3a](#).

Defines: [fTimeFormat 6a](#), [17](#).

The form `fTimeFormat` nominally POSTs the request, but this is overridden using `postChange`. The selected option is sent as a query in the form `timeformat=iso`, `timeFormat=us` etc.



```

< Set up to send time format: JavaScript 6a > ≡
document.fTimeFormat.format.onChange = function () {
  for (var o = document.fTimeFormat.format.options, i = 0;
       i < o.length;
       i++) {
    if (o[i].selected) {
      postChange.start("timeFormat=" + o[i].value);
      break;
    }
  }
};
◇

```

Fragment referenced in [25](#).

Uses: [fTimeFormat 5](#), [postChange 20a](#).

In [AJAX\\_Change](#), check whether it has been called to change the time format; a query `foo=bar` can be retrieved from the `Request` as the property "foo" with value "bar" (if the property isn't present in the request, the empty string is returned).

```

< Checks for changed properties: Ada 6b > ≡
declare
  Property : constant String
    := EWS.HTTP.Get_Property ("timeFormat", From_Request.all);
begin
  if Property /= "" then
    Put_Line ("saw timeFormat=" & Property);
    Current_Date_Format := Date_Format'Value (Property);
  end if;
end;
◇

```

Fragment defined by [6b](#), [9a](#), [12c](#).

Fragment referenced in [20d](#).

Uses: [Current\\_Date\\_Format 4e](#), [Date\\_Format 4e](#).

## Chapter 3

# Radio buttons

Note, with radio buttons the `value` identifies which radio button has been pressed, and does not change; it's the `checked` field which changes, and only one can be `true` at a time.

Using radio buttons	Forward Light	<input type="radio"/> Red	<input checked="" type="radio"/> Blue
	Aft Light	<input checked="" type="radio"/> Red	<input type="radio"/> Blue

Figure 3.1: Using radio buttons

Figure 3.1 shows the part of the example that relates to radio buttons. There are two lights, Forward and Aft, each of which can show Red or Blue. The HTML is implemented in a form with

( *Radio buttons: HTML 7* ) ≡

```
<tr>
  <td>
    Using radio buttons
  </td>
  <td>
    <form method="PUT" name="lights" id="lights">
      <table border="1">
        <tr>
          <td id="forward-light">Forward Light</td>
          <td>
            <input
              type="radio"
              name="forward"
              value="red"
              checked="true">Red</input>
          </td>
          <td>
            <input
              type="radio"
              name="forward"
              value="blue">Blue</input>
          </td>
        </tr>
        <tr>
          <td id="aft-light">Aft Light</td>
          <td>
```



```

< Checks for changed properties: Ada 9a > ≡
  declare
    Property : constant String
      := EWS.HTTP.Get_Property ("forward-light", From_Request.all);
  begin
    if Property /= "" then
      Put_Line ("saw forward-light=" & Property);
      Forward_Light := Light_State'Value (Property);
    end if;
  end;
  declare
    Property : constant String
      := EWS.HTTP.Get_Property ("aft-light", From_Request.all);
  begin
    if Property /= "" then
      Put_Line ("saw aft-light=" & Property);
      Aft_Light := Light_State'Value (Property);
    end if;
  end;
  ◇

```

Fragment defined by 6b, 9a, 12c.

Fragment referenced in 20d.

Uses: Aft\_Light 8b, Forward\_Light 8b, Light\_State 8b, aft-light 7, forward-light 7.

Because the server can be accessed by more than one web client, and all the other clients need to show changes, the current light state is retrieved every second via a `CyclicHttpRequest` to the URL `lightState.xml`. The response is expected to be XML:

```

<lights>
  <forward-light>lmp</forward-light>
  <aft-light>lmp</aft-light>
</lights>

```

where `lmp` specifies a colour (will be red or blue).

```

< Cyclic interactions: JavaScript 9b > ≡
  var lightStateRequest = new CyclicHttpRequest
    ("lightState.xml",
    function (r) {
      var xml = r.responseXML;
      document.getElementById("forward-light").style.color =
        xml.getElementsByTagName("forward-light")[0].firstChild.nodeValue;
      document.getElementById("aft-light").style.color =
        xml.getElementsByTagName("aft-light")[0].firstChild.nodeValue;
    },
    1000);
  ◇

```

Fragment defined by 4a, 9b.

Fragment referenced in 25.

Defines: `lightStateRequest` 9c.

Uses: aft-light 7, forward-light 7, lightState.xml 10b.

The cyclic interaction needs to be started when the page is loaded.

```

< Start getting cyclic data: JavaScript 9c > ≡
  lightStateRequest.start();
  ◇

```

Fragment defined by 4b, 9c.

Fragment referenced in 25.

Uses: `lightStateRequest` 9b.

The Ada code which receives the cyclic `lightState.xml` request is in the function `AJAX_Light_State`.

*< Specs of dynamic pages: Ada 10a >* ≡

```
function AJAX_Light_State
  (From_Request : HTTP.Request_P)
  return Dynamic.Dynamic_Response'Class;
```

◇

Fragment defined by 4c, 10a, 14c, 18b, 20b.

Fragment referenced in 26.

Defines: AJAX\_Light\_State 10b.

The function is registered with the server, to be called to respond to the URL `lightState.xml`.

*< Register dynamic pages: Ada 10b >* ≡

```
Dynamic.Register (AJAX_Light_State'Unrestricted_Access, "/lightState.xml");
```

◇

Fragment defined by 4d, 10b, 14d, 18c, 20c.

Fragment referenced in 26.

Defines: `lightState.xml` 9b.

Uses: AJAX\_Light\_State 10ac.

*< Bodies of dynamic pages: Ada 10c >* ≡

```
function AJAX_Light_State
  (From_Request : HTTP.Request_P)
  return Dynamic.Dynamic_Response'Class is
  Result : Dynamic.Dynamic_Response (From_Request);
begin
  Dynamic.Set_Content_Type (Result, To => Types.XML);
  Dynamic.Append (Result, "<lights>");
  Dynamic.Append_Element
    (Result,
     "forward-light",
     Ada.Strings.Fixed.Translate
       (Forward_Light'Img,
        Ada.Strings.Maps.Constants.Lower_Case_Map));
  Dynamic.Append_Element
    (Result,
     "aft-light",
     Ada.Strings.Fixed.Translate
       (Aft_Light'Img,
        Ada.Strings.Maps.Constants.Lower_Case_Map));
  Dynamic.Append (Result, "</lights>");
  return Result;
end AJAX_Light_State;
```

◇

Fragment defined by 4f, 10c, 14e, 18d, 20d.

Fragment referenced in 26.

Defines: AJAX\_Light\_State 10b.

Uses: Aft\_Light 8b, Forward\_Light 8b, aft-light 7, forward-light 7, lights 7.

## Chapter 4

# Checkboxes

Using checkboxes	Port Lamp	<input checked="" type="checkbox"/>
	Starboard Lamp	<input type="checkbox"/>

Figure 4.1: Using checkboxes

Figure 4.1 shows the part of the example that relates to checkboxes. There are two Lamps, Port and Starboard, which are separately switched.

*< Checkboxes: HTML 11 >* ≡

```
<tr>
  <td>
    Using checkboxes
  </td>
  <td>
    <form method="PUT" name="lamps" id="lamps">
      <table border="1">
        <tr>
          <td>Port Lamp</td>
          <td>
            <input
              type="checkbox"
              name="lamp"
              value="port"/>
          </td>
        </tr>
        <tr>
          <td>Starboard Lamp</td>
          <td>
            <input
              type="checkbox"
              name="lamp"
              value="starboard"/>
          </td>
        </tr>
      </table>
    </form>
  </td>
</tr>
```

◇

Fragment referenced in 23c.

Defines: lamp 12ac, 17, 18d, lamps 12a, 17.

The onclick action is a function whose source is, for example,

```
postChange.start('lamp=0&value=port&checked='+document.lamps.lamp[0].checked);
```

Note that this sends the clicked checkbox's index (`document.lamps.lamp[0]` is the first) as well as the value (corresponding to the internal name of the box); the receiving Ada code presently uses the index, though it would obviously be better to use the value.

```
<Set up the checkboxes: JavaScript 12a> ≡
  for (var c = document.lamps.lamp, i = 0; i < c.length; i++) {
    c[i].onclick = new Function(
      "postChange.start('lamp=" + i
      + "&value=" + document.lamps.lamp[i].value
      + "&checked=" + document.lamps.lamp[" + i + "].checked);"
    );
  }
}
```

◇

Fragment referenced in 25.

Uses: lamp 11, lamps 11, postChange 20a.

The Ada code which receives the lamp property changes updates global data.

```
<Global data: Ada 12b> ≡
  Lamps : array (0 .. 1) of Boolean := (others => True);
```

◇

Fragment defined by 4e, 8b, 12b.

Fragment referenced in 26.

Defines: Lamps 12c, 18d.

```
<Checks for changed properties: Ada 12c> ≡
  declare
    Lamp : constant String
      := EWS.HTTP.Get_Property ("lamp", From_Request.all);
  begin
    if Lamp /= "" then
      declare
        Checked : constant String
          := EWS.HTTP.Get_Property ("checked", From_Request.all);
        Value : constant String
          := EWS.HTTP.Get_Property ("value", From_Request.all);
      begin
        Put_Line ("saw lamp=" & Lamp
          & " value=" & Value
          & " checked=" & Checked);
        Lamps (Natural'Value (Lamp)) := Boolean'Value (Checked);
      end;
    end if;
  end;
```

◇

Fragment defined by 6b, 9a, 12c.

Fragment referenced in 20d.

Uses: Lamps 12b, lamp 11.

## Chapter 5

# File upload

Figure 5.1 shows the file upload dialog. Figure 5.2 shows the alert box displayed when the upload is complete.

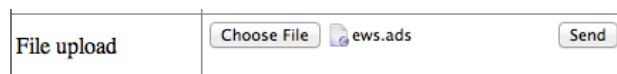


Figure 5.1: The file upload dialog

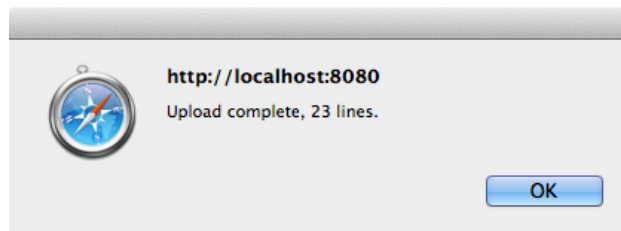


Figure 5.2: Upload completed

The file upload dialog is in a form, in a table cell. The form contains two elements: a file selector (`datafile`) and a submit button.

When the form is submitted, the content of the selected file is sent as a multipart attachment to the URL `/fileInput` (the form's `action`).

The other components of this example program used explicit JavaScript methods to send the data to the server. In the case of a file upload, the protocol is complex enough that it's best left to the browser. When the browser has submitted the request, it expects to get a new page in response; we don't want that, so we use the form's `target` attribute to direct the response to an invisible frame `iFrame` so that the current page remains displayed.



*< File upload: HTML 14a >* ≡

```
<tr>
  <td>
    File upload
  </td>
  <td>
    <form method="POST"
      enctype="multipart/form-data"
      name="fileInput"
      action="fileInput"
      target="iFrame">
      <input type="file" name="datafile" size="128">
      <input type="submit" name="send" value="Send">
    </form>
  </td>
</tr>
◇
```

Fragment referenced in 23c.

Uses: fileInput 14d, iFrame 14b.

*< File upload's target iFrame: HTML 14b >* ≡

```
<iframe name="iFrame" id="iFrame" src="about:blank" width="0" height="0">
</iframe>
◇
```

Fragment referenced in 23c.

Defines: iFrame 14a.

The Ada code which receives the fileInput request is in the function File\_Input.

*< Specs of dynamic pages: Ada 14c >* ≡

```
function File_Input
  (From_Request : HTTP.Request_P)
  return Dynamic.Dynamic_Response'Class;
◇
```

Fragment defined by 4c, 10a, 14c, 18b, 20b.

Fragment referenced in 26.

Defines: File\_Input 14d.

The function is registered with the server, to be called to respond to the URL fileInput.

*< Register dynamic pages: Ada 14d >* ≡

```
Dynamic.Register (File_Input'Unrestricted_Access, "/fileInput");
◇
```

Fragment defined by 4d, 10b, 14d, 18c, 20c.

Fragment referenced in 26.

Defines: fileInput 14ae.

Uses: File\_Input 14ce.

The response generated is a page containing an alert box (5.2), which is displayed even though the iFrame swallows the HTML content.

*< Bodies of dynamic pages: Ada 14e >* ≡

```
function File_Input
  (From_Request : HTTP.Request_P) return Dynamic.Dynamic_Response'Class is
  < Upload_Result, calculates file input result: Ada 15, ... >
  C : HTTP.Cursor;
  Lines : Natural := 0;
  Line : String (1 .. 1024);
  Last : Natural;
  Attachments : constant HTTP.Attachments
    := HTTP.Get_Attachments (From_Request.all);
begin
  Put_Line ("saw fileInput with attachment length"
    & HTTP.Get_Content (Attachments)'Length'Img);
```

```

if HTTP.Get_Content (Attachments)'Length /= 0 then
  begin
    HTTP.Open (C, Attachments);
    while not HTTP.End_Of_File (C) loop
      Lines := Lines + 1;
      Put (Lines'Img & ": ");
      HTTP.Get_Line (C, Line, Last);
      Put_Line (Line (1 .. Last));
    end loop;
    HTTP.Close (C);
    return Upload_Result
      ("Upload complete," & Lines'Img & " lines.");
  exception
    when E : others =>
      begin
        HTTP.Close (C);
      exception
        when others => null;
      end;
      return Upload_Result
        ("Upload failed: " & Ada.Exceptions.Exception_Message (E));
  end;
else
  declare
    Result : Dynamic.Dynamic_Response (From_Request);
  begin
    Dynamic.Set_Content_Type (Result, To => Types.Plain);
    Dynamic.Set_Content (Result, "null");
    return Result;
  end;
end if;
end File_Input;

```

◇

Fragment defined by [4f](#), [10c](#), [14e](#), [18d](#), [20d](#).

Fragment referenced in [26](#).

Defines: [File\\_Input](#) [14d](#).

Uses: [Upload\\_Result](#) [15](#), [16](#), [fileInput](#) [14d](#).

```

⟨Upload_Result, calculates file input result: Ada 15⟩ ≡
  function Upload_Result (Message : String)
    return Dynamic.Dynamic_Response'Class;
  
```

◇

Fragment defined by [15](#), [16](#).

Fragment referenced in [14e](#).

Defines: [Upload\\_Result](#) [14e](#).

If the **Message** to be returned contains multiple lines, they have to be translated to the `\n` that the JavaScript `alert()` function expects.

```

⟨Upload_Result, calculates file input result: Ada 16⟩ ≡
function Upload_Result (Message : String)
return Dynamic.Dynamic_Response'Class is
  Result : Dynamic.Dynamic_Response (From_Request);
begin
  Dynamic.Set_Content_Type (Result, To => Types.HTML);
  Dynamic.Append (Result, "<body onload=\"alert('");
  for C in Message'Range loop
    case Message (C) is
      when ASCII.CR | ASCII.NUL => null;
      when ASCII.LF => Dynamic.Append (Result, "\n");
      when others =>
        Dynamic.Append (Result, String'(1 => Message (C)));
    end case;
  end loop;
  Dynamic.Append (Result, "'")>");
  return Result;
end Upload_Result;

```

◇  
 Fragment defined by [15](#), [16](#).  
 Fragment referenced in [14e](#).  
 Defines: Upload\_Result [14e](#).

## Chapter 6

# Retrieving the initial state

When a new client connects to the server, it must retrieve the current state and set the page elements accordingly.

The server responds to the URL `state.xml` with the current state in XML format,

```
<state>
  <time-format>fmt</time-format>
  <forward-light>lmp</forward-light>
  <aft-light>lmp</aft-light>
  <lamp>bool</lamp>
  <lamp>bool</lamp>
</state>
```

where

`fmt` can be `iso`, `us`, `european` or `locale`,

`lmp` can be `red` or `blue`, and

`bool` can be `false` or `true`.

and the first `lamp` element is for the starboard lamp and the second is for the port lamp.

⟨*Retrieving the initial state: JavaScript 17*⟩ ≡

```
var stateRequest = new OneshotHttpInteraction
("state.xml",
 null,
 function (r) {
   var x = r.responseXML;
   var value = x.getElementsByTagName("time-format")[0].firstChild.nodeValue;
   for (var o = document.fTimeFormat.format.options, i = 0;
        i < o.length;
        i++) {
     o[i].selected = (o[i].value == value);
   }
   value = x.getElementsByTagName("forward-light")[0].firstChild.nodeValue;
   for (var o = document.lights.forward, i = 0;
        i < o.length;
        i++) {
     o[i].checked = (o[i].value == value);
   }
   value = x.getElementsByTagName("aft-light")[0].firstChild.nodeValue;
   for (var o = document.lights.aft, i = 0;
```

```

        i < o.length;
        i++) {
    o[i].checked = (o[i].value == value);
}
var lamps = x.getElementsByTagName("lamp");
for (var c = document.lamps.lamp, i = 0; i < c.length; i++) {
    c[i].checked = lamps[i].firstChild.nodeValue == "true";
}
});

```

◇

Fragment referenced in 25.

Defines: `stateRequest` 18a.

Uses: `aft-light` 7, `fTimeFormat` 5, `forward-light` 7, `lamp` 11, `lamps` 11, `lights` 7, `state.xml` 18c.

When the page is opened, request the initial state.

```

<Request the initial state: JavaScript 18a> ≡
    stateRequest.start();

```

◇

Fragment referenced in 25.

Uses: `stateRequest` 17.

The Ada code which receives the `state.xml` request is in the function `AJAX_Status`.

```

<Specs of dynamic pages: Ada 18b> ≡
    function AJAX_Status
        (From_Request : HTTP.Request_P)
        return Dynamic.Dynamic_Response'Class;

```

◇

Fragment defined by 4c, 10a, 14c, 18b, 20b.

Fragment referenced in 26.

Defines: `AJAX_Status` 18c.

The function is registered with the server, to be called to respond to the URL `state.xml`.

```

<Register dynamic pages: Ada 18c> ≡
    Dynamic.Register (AJAX_Status'Unrestricted_Access, "/state.xml");

```

◇

Fragment defined by 4d, 10b, 14d, 18c, 20c.

Fragment referenced in 26.

Defines: `state.xml` 17.

Uses: `AJAX_Status` 18bd.

```

<Bodies of dynamic pages: Ada 18d> ≡

```

```

    function AJAX_Status
        (From_Request : HTTP.Request_P)
        return Dynamic.Dynamic_Response'Class is
        Result : Dynamic.Dynamic_Response (From_Request);
    begin
        Dynamic.Set_Content_Type (Result, To => Types.XML);
        Dynamic.Append (Result, "<state>");
        Dynamic.Append_Element
            (Result,
             "time-format",
             Ada.Strings.Fixed.Translate
                (Current_Date_Format'Img,
                 Ada.Strings.Maps.Constants.Lower_Case_Map));
        Dynamic.Append_Element
            (Result,
             "forward-light",
             Ada.Strings.Fixed.Translate
                (Forward_Light'Img,
                 Ada.Strings.Maps.Constants.Lower_Case_Map));
    end AJAX_Status;

```

```

Dynamic.Append_Element
  (Result,
   "aft-light",
   Ada.Strings.Fixed.Translate
     (Aft_Light'Img,
      Ada.Strings.Maps.Constants.Lower_Case_Map));
for L in Lamps'Range loop
  Dynamic.Append_Element
    (Result,
     "lamp",
     Ada.Strings.Fixed.Translate
       (Lamps (L)'Img,
        Ada.Strings.Maps.Constants.Lower_Case_Map));
end loop;
Dynamic.Append (Result, "</state>");
return Result;
end AJAX_Status;

```

◇

Fragment defined by [4f](#), [10c](#), [14e](#), [18d](#), [20d](#).

Fragment referenced in [26](#).

Defines: [AJAX\\_Status](#) [18c](#).

Uses: [Aft\\_Light](#) [8b](#), [Current\\_Date\\_Format](#) [4e](#), [Forward\\_Light](#) [8b](#), [Lamps](#) [12b](#), [aft-light](#) [7](#), [forward-light](#) [7](#), [lamp](#) [11](#).

# Chapter 7

## Utilities

### 7.1 Notifying the server of changes

`postChange` is a one-shot interaction; it sends a request to the URL `ajaxChange`. If `postChange.start()` is called with a parameter (for example, `"foo=bar"`, the parameter is sent to the URL as a query. No specific response is expected.

```
< Generalised change action request: JavaScript 20a > ≡  
    var postChange = new OneshotHttpRequest  
        ("ajaxChange",  
         null,  
         function (r) { });  
    ◇
```

Fragment referenced in 25.

Defines: `postChange` 6a, 12a, 21.

Uses: `ajaxChange` 20c.

The Ada code which receives the `ajaxChange` request is in the function `Ajax_Change`.

```
< Specs of dynamic pages: Ada 20b > ≡  
    function AJAX_Change  
        (From_Request : HTTP.Request_P)  
        return Dynamic.Dynamic_Response'Class;  
    ◇
```

Fragment defined by 4c, 10a, 14c, 18b, 20b.

Fragment referenced in 26.

Defines: `AJAX_Change` 20c.

The function is registered with the server, to be called to respond to the URL `ajaxChange`.

```
< Register dynamic pages: Ada 20c > ≡  
    Dynamic.Register (AJAX_Change'Unrestricted_Access, "/ajaxChange");  
    ◇
```

Fragment defined by 4d, 10b, 14d, 18c, 20c.

Fragment referenced in 26.

Defines: `ajaxChange` 20a.

Uses: `AJAX_Change` 20bd.

```
< Bodies of dynamic pages: Ada 20d > ≡  
    function AJAX_Change  
        (From_Request : HTTP.Request_P)  
        return Dynamic.Dynamic_Response'Class is  
        Result : Dynamic.Dynamic_Response (From_Request);  
    begin  
        Put_Line ("AJAX_Change called.");  
    end;
```

```

    < Checks for changed properties: Ada 6b, ... >
    Dynamic.Set_Content_Type (Result, To => Types.Plain);
    Dynamic.Set_Content (Result, "OK");
    return Result;
end AJAX_Change;

```

Fragment defined by [4f](#), [10c](#), [14e](#), [18d](#), [20d](#).

Fragment referenced in [26](#).

Defines: [AJAX\\_Change 20c](#).

## 7.2 Set up radio buttons

`Set up radio buttons` utility is a `nuweb` parameterised fragment. There's no indication in a parameterised fragment's name that it is parameterised; when invoked with parameters, occurrences of `@n` are replaced by the `n`'th parameter.

The first parameter (`@1`) is the name of the buttons to be set up (they all have the same name): e.g. `document.formName.buttonName`.

The second parameter (`@2`) is the property name that is passed to `postChange`.

When one of the buttons is clicked, a one-shot `ajaxChange` interaction is invoked, posting the query `property=value` where `property` is the name passed in the second parameter and `value` is the `value` attribute of the button.

```

< "Set up radio buttons" utility: JavaScript 21 > ≡
  for (var j = 0; j < @1.length; j++) {
    @1[j].onclick = new Function(
      "postChange.start('" + @2 + "=" + @1[j].value + "');");
  };

```

Fragment referenced in [8a](#).

Uses: [postChange 20a](#).



## Chapter 8

# HTML pages

```
"ajax.html" 22a≡
  <HTML licence header 2d>
  <html>
  <head>
  <HTML header 22b>
  </head>
  <body bgcolor="white">
  <Page heading 23a>
  <Introductory material 23b>
  <p><hr>
  <The demonstrations 23c>
  <Author link 24>
  </body>
  </html>
  ◇
<HTML header 22b> ≡
  <title>EWS: AJAX demonstration</title>

  <!-- NB,for Internet Explorer you mustn't use the empty-element
    syntax. For Safari, you have to close the element. -->
  <script type="text/javascript"src="HttpInteraction.js"></script>
  <script type="text/javascript"src="ajax.js"></script>

  <style type="text/css">
    div#demos table { margin : 0.2em 1em;
                      font-size : 100%;
                      border-collapse : collapse; }
    div#demos th,td { padding : 0.2em; }
  </style>
  ◇
Fragment referenced in 22a.
```

⟨ *Page heading 23a* ⟩ ≡

```
<table width="100%">
  <tr>
    <td><h1>Embedded Web Server: AJAX demonstration</h1></td>
    <td align="right">
      <a href="http://sourceforge.net"> </a>
    </td>
  </tr>
</table>
◇
```

Fragment referenced in [22a](#).

⟨ *Introductory material 23b* ⟩ ≡

```
<p>EWS is a web server construction kit, designed for embedded
applications using the <a
href="http://libre.adacore.com/">GNAT</a> Ada
compiler.

<p>The project is hosted on SourceForge. You can see
the <a
href="http://sourceforge.net/project/showfiles.php?group_id=95861">releases</a>,
and you
can <a
href="http://embed-web-srvr.hg.sourceforge.net/hgweb/embed-web-srvr/embed-web-srvr/">browse</a>
the <a href="http://mercurial.selenic.com">Mercurial</a> repository</a>.

<p>EWS comes with a demonstration of its facilities. The available
facilities are described in <a href="ews.pdf">this document</a>, which
also acts as the source code for the demonstration using the <a
href="http://www.literateprogramming.com/">Literate Programming</a>
facilities of <a
href="https://sourceforge.net/projects/nuweb-py.nuweb.p/">nuweb.py</a>.
◇
```

Fragment referenced in [22a](#).

⟨ *The demonstrations 23c* ⟩ ≡

```
<p>Below are demonstrations of
various <a href="http://www.amazon.co.uk/exec/obidos/ASIN/0471777781/qid%3D1146719450/203-6928631-0011916">AJAX</a>
technologies:

<div id="demos" align="center">
  <table border="1">
    <tr>
      <td>⟨ Cyclic updating and select/options: HTML 3a ⟩
    </td>
    <td>⟨ Radio buttons: HTML 7 ⟩
    </td>
    <td>⟨ Checkboxes: HTML 11 ⟩
    </td>
    <td>⟨ File upload: HTML 14a ⟩
    </td>
  </table>
</div>
⟨ File upload's target iFrame: HTML 14b ⟩
◇
```

Fragment referenced in [22a](#).

*Author link 24* ≡  
 <hr>  
 <i>  
 <address>  
 <a href="mailto:simon@pushface.org">Simon Wright</a>  
 </address>  
 </i>  
 ◊  
 Fragment referenced in [22a](#).

## Chapter 9

# JavaScript

This script, loaded by `ajax.html`, relies on the utility `HttpInteraction.js` having been already loaded by the page.

```
"ajax.js" 25≡  
    < JavaScript licence header 2c >  
    < Retrieving the initial state: JavaScript 17 >  
    < Cyclic interactions: JavaScript 4a, ... >  
    < Generalised change action request: JavaScript 20a >  
  
    /**  
     * Assign event handlers and begin fetching.  
     */  
    window.onload = function () {  
        < Request the initial state: JavaScript 18a >  
        < Start getting cyclic data: JavaScript 4b, ... >  
        < Set up to send time format: JavaScript 6a >  
        < Set up the radio buttons: JavaScript 8a >  
        < Set up the checkboxes: JavaScript 12a >  
    };  
    ◇
```

# Chapter 10

## Ada

### 10.1 Ada code

"ews\_demo.adb" 26≡

```

    < Ada licence header 2b >
with Ada.Calendar;
with Ada.Exceptions;
with Ada.Strings.Fixed;
with Ada.Strings.Maps.Constants;
with Ada.Text_IO; use Ada.Text_IO;
with EWS.Dynamic;
with EWS.HTTP;
with EWS.Server;
with EWS.Types;
with GNAT.Calendar.Time_IO;
with GNAT.Command_Line;

with EWS_Htdocs;

procedure EWS_Demo is

    use EWS;

    < Specs of dynamic pages: Ada 4c, ... >

    < Global data: Ada 4e, ... >

    < Bodies of dynamic pages: Ada 4f, ... >

    Verbose : Boolean := False;

begin
    begin
        loop
            case GNAT.Command_Line.Getopt ("v") is
                when 'v' =>
                    Verbose := True;
                when ASCII.NUL =>
                    exit;
            end case;
        end loop;
    end begin;
end;
```

```

        when others =>
            null; -- never taken
        end case;
    end loop;
exception
    when GNAT.Command_Line.Invalid_Switch =>
        Put_Line (Standard_Error,
            "invalid switch -" & GNAT.Command_Line.Full_Switch);
        return;
end;

end;

< Register dynamic pages: Ada 4d, ... >

Server.Serve (Using_Port => 8080,
    With_Stack => 40_000,
    Tracing => Verbose);

delay 1_000_000.0;

end EWS_Demo;
◇

```

## 10.2 GNAT Project

```

"ews_demo.gpr" 27≡
with "../ews";
with "xmlada";
project EWS_Demo is

    for Main use ("ews_demo.adb");
    for Exec_Dir use ".";
    for Source_Dirs use ".";
    for Object_Dir use ".build";

    package Builder is
        for Default_Switches ("ada") use ("-g");
    end Builder;

    package Compiler is
        for Default_Switches ("ada") use
            (
                "-gnat12",
                "-O2",
                "-gnatqQafoy"
            );
    end Compiler;

    package Binder is
        for Default_Switches ("ada") use ("-E");
    end Binder;

end EWS_Demo;
◇

```

# Appendix A

## About this document

This document is prepared using [nuweb](#), a language-agnostic [Literate Programming](#) tool. The actual variant used is [nuweb.py](#).

# Appendix B

## Index

### B.1 Files

"`ajax.html`" Defined by [22a](#).

"`ajax.js`" Defined by [25](#).

"`ews_demo.adb`" Defined by [26](#).

"`ews_demo.gpr`" Defined by [27](#).

### B.2 Macros

- `<"Set up radio buttons" utility: JavaScript 21>` Referenced in [8a](#).
- `<Ada licence header 2b>` Referenced in [26](#).
- `<Author link 24>` Referenced in [22a](#).
- `<Bodies of dynamic pages: Ada 4f, 10c, 14e, 18d, 20d>` Referenced in [26](#).
- `<Change the time display format: HTML 5>` Referenced in [3a](#).
- `<Checkboxes: HTML 11>` Referenced in [23c](#).
- `<Checks for changed properties: Ada 6b, 9a, 12c>` Referenced in [20d](#).
- `<Copyright 2a>` Referenced in [2bcd](#).
- `<Cyclic interactions: JavaScript 4a, 9b>` Referenced in [25](#).
- `<Cyclic updating and select/options: HTML 3a>` Referenced in [23c](#).
- `<Display the current time: HTML 3b>` Referenced in [3a](#).
- `<File upload's target iFrame: HTML 14b>` Referenced in [23c](#).
- `<File upload: HTML 14a>` Referenced in [23c](#).
- `<Generalised change action request: JavaScript 20a>` Referenced in [25](#).
- `<Global data: Ada 4e, 8b, 12b>` Referenced in [26](#).
- `<HTML header 22b>` Referenced in [22a](#).
- `<HTML licence header 2d>` Referenced in [22a](#).
- `<Introductory material 23b>` Referenced in [22a](#).
- `<JavaScript licence header 2c>` Referenced in [25](#).
- `<Page heading 23a>` Referenced in [22a](#).
- `<Radio buttons: HTML 7>` Referenced in [23c](#).
- `<Register dynamic pages: Ada 4d, 10b, 14d, 18c, 20c>` Referenced in [26](#).
- `<Request the initial state: JavaScript 18a>` Referenced in [25](#).
- `<Retrieving the initial state: JavaScript 17>` Referenced in [25](#).
- `<Set up the checkboxes: JavaScript 12a>` Referenced in [25](#).
- `<Set up the radio buttons: JavaScript 8a>` Referenced in [25](#).
- `<Set up to send time format: JavaScript 6a>` Referenced in [25](#).
- `<Specs of dynamic pages: Ada 4c, 10a, 14c, 18b, 20b>` Referenced in [26](#).
- `<Start getting cyclic data: JavaScript 4b, 9c>` Referenced in [25](#).



⟨The demonstrations 23c⟩ Referenced in 22a.  
⟨Upload\_Result, calculates file input result: Ada 15, 16⟩ Referenced in 14e.

## B.3 Definitions

AJAX\_Change: defined in 20bd, used in 20c.  
AJAX\_Light\_State: defined in 10ac, used in 10b.  
AJAX\_Status: defined in 18bd, used in 18c.  
AJAX\_Time: defined in 4cf, used in 4d.  
Aft\_Light: defined in 8b, used in 9a, 10c, 18d.  
Current\_Date\_Format: defined in 4e, used in 4f, 6b, 18d.  
Date\_Format: defined in 4e, used in 6b.  
File\_Input: defined in 14ce, used in 14d.  
Forward\_Light: defined in 8b, used in 9a, 10c, 18d.  
Lamps: defined in 12b, used in 12c, 18d.  
Light\_State: defined in 8b, used in 9a.  
Upload\_Result: defined in 15, 16, used in 14e.  
aft-light: defined in 7, used in 9ab, 10c, 17, 18d.  
ajaxChange: defined in 20c, used in 20a.  
ajaxTime: defined in 4d, used in 4a.  
fTimeFormat: defined in 5, used in 6a, 17.  
fileInput: defined in 14d, used in 14ae.  
forward-light: defined in 7, used in 9ab, 10c, 17, 18d.  
iFrame: defined in 14b, used in 14a.  
lamp: defined in 11, used in 12ac, 17, 18d.  
lamps: defined in 11, used in 12a, 17.  
lightState.xml: defined in 10b, used in 9b.  
lightStateRequest: defined in 9b, used in 9c.  
lights: defined in 7, used in 10c, 17.  
postChange: defined in 20a, used in 6a, 12a, 21.  
state.xml: defined in 18c, used in 17.  
stateRequest: defined in 17, used in 18a.  
timeDisplay: defined in 3b, used in 4a.  
timeRequest: defined in 4a, used in 4b.