

Embedded Web Server

Simon Wright
simon@pushface.org

9.viii.2022

Contents

1	Introduction	1
2	Ada main program	2
2.1	Ada code	2
2.2	GNAT Project	4
3	Copyright and Licencing	5
4	Cyclic updating and select/options	7
4.1	Displaying the current time	7
4.2	Changing the time display format	9
5	Radio buttons	11
6	Checkboxes	15
7	File upload	17
8	Retrieving the initial state	21
9	Utilities	24
9.1	Notifying the server of changes	24
9.2	Set up radio buttons	25
10	HTML pages	26
11	JavaScript	28
A	About this document	29
B	Index	30
B.1	Files	30
B.2	Macros	30
B.3	Definitions	31

List of Figures

1.1	The dynamic part of the example page	1
4.1	Displaying the current time, and controlling the format	7
5.1	Using radio buttons	11
6.1	Using checkboxes	15
7.1	The file upload dialog	17
7.2	Upload completed	17

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_generator`) 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	<table border="1"><tr><td>Forward Light</td><td><input type="radio"/> Red</td><td><input checked="" type="radio"/> Blue</td></tr><tr><td>Aft Light</td><td><input checked="" type="radio"/> Red</td><td><input type="radio"/> Blue</td></tr></table>	Forward Light	<input type="radio"/> Red	<input checked="" type="radio"/> Blue	Aft Light	<input checked="" type="radio"/> Red	<input type="radio"/> Blue
Forward Light	<input type="radio"/> Red	<input checked="" type="radio"/> Blue					
Aft Light	<input checked="" type="radio"/> Red	<input type="radio"/> Blue					
Using checkboxes	<table border="1"><tr><td>Port Lamp</td><td><input checked="" type="checkbox"/></td></tr><tr><td>Starboard Lamp</td><td><input type="checkbox"/></td></tr></table>	Port Lamp	<input checked="" type="checkbox"/>	Starboard Lamp	<input type="checkbox"/>		
Port Lamp	<input checked="" type="checkbox"/>						
Starboard Lamp	<input type="checkbox"/>						
File upload	<input type="button" value="Choose 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`.

Chapter 2

Ada main program

This chapter describes the Ada main program and the [GNAT Project](#) used to build it.

2.1 Ada code

The main program (EWS_Demo).

```
"ews_demo.adb" 2≡  
  ⟨ Ada licence header 5b ⟩  
  
  ⟨ Main program standard context: Ada 3a ⟩  
  
  ⟨ Main program generated context: Ada 3b ⟩  
  
  procedure EWS_Demo is  
  
    use EWS;  
  
    ⟨ Specs of dynamic pages: Ada 8c, ... ⟩  
  
    ⟨ Global data: Ada 8e, ... ⟩  
  
    ⟨ Bodies of dynamic pages: Ada 8f, ... ⟩  
  
    Verbose : Boolean := False;  
  
  begin  
  
    begin  
      loop  
        case GNAT.Command_Line.Getopt ("v") is  
          when 'v' =>  
            Verbose := True;  
          when ASCII.NUL =>  
            exit;  
          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;

    < Register dynamic pages: Ada 8d, ... >

    Put_Line ("Connect to ews_demo using e.g. http://localhost:8080");

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

    delay 1_000_000.0;

end EWS_Demo;
◇

```

Users: EWS_Demo in 4.

This is the context required for the main program, with the exception of that for the code generated from the site file tree.

```

< Main program standard context: Ada 3a > ≡
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;
◇

```

Fragment referenced in 2.

This is the output of the program `ews-make_htdocs`, which converts a file tree into static Ada source code.

```

< Main program generated context: Ada 3b > ≡
with EWS_Htdocs;
◇

```

Fragment referenced in 2.

2.2 GNAT Project

```
"ews_demo.gpr" 4≡
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";
    for Create_Missing_Dirs use "true";

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

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

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

end EWS_Demo;
◇
Uses: EWS_Demo 2.
```

Chapter 3

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 5a > ≡  
  Copyright 2013-2022, Simon Wright <simon@pushface.org>  
  ◇
```

Fragment referenced in [5bc](#), [6](#).

In Ada,

```
< Ada licence header 5b > ≡  
  -- < Copyright 5a >  
  --  
  -- 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 [2](#).

In JavaScript,

```
< JavaScript licence header 5c > ≡  
  /*  
   * < Copyright 5a >  
   *  
   * 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 [28](#).

In HTML,

$\langle \textit{HTML licence header 6} \rangle \equiv$

```
<!--
```

```
   $\langle \textit{Copyright 5a} \rangle$ 
```

```
  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 [26a](#).

Chapter 4

Cyclic updating and select/options

This part of the demonstration, shown in Figure 4.1, 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 4.1: Displaying the current time, and controlling the format

< Cyclic updating and select/options: HTML 7a > ≡

```
<tr>
  < Display the current time: HTML 7b >
</tr>
<tr>
  < Change the time display format: HTML 9a >
</tr>
```

◇

Fragment referenced in 27c.

4.1 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 7b > ≡

```
<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 7a.

Users: `timeDisplay` in 8a.

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 8a > ≡
  var timeRequest = new CyclicHttpRequest
    ("ajaxTime",
      function (r) {
        document.getElementById("timeDisplay").innerHTML = r.responseText;
      },
      1000);

```

Fragment defined by [8a](#), [13b](#).

Fragment referenced in [28](#).

Users: `timeRequest` in [8b](#).

Uses: `ajaxTime` [8d](#), `timeDisplay` [7b](#).

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

```

< Start getting cyclic data: JavaScript 8b > ≡
  timeRequest.start();

```

Fragment defined by [8b](#), [13c](#).

Fragment referenced in [28](#).

Uses: `timeRequest` [8a](#).

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

```

< Specs of dynamic pages: Ada 8c > ≡
  function AJAX_Time
    (From_Request : HTTP.Request_P)
  return Dynamic.Dynamic_Response'Class;

```

Fragment defined by [8c](#), [14a](#), [18c](#), [22b](#), [24b](#).

Fragment referenced in [2](#).

Users: `AJAX_Time` in [8d](#).

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 8d > ≡
  Dynamic.Register (AJAX_Time'Unrestricted_Access, "/ajaxTime");

```

Fragment defined by [8d](#), [14b](#), [18d](#), [22c](#), [24c](#).

Fragment referenced in [2](#).

Users: `ajaxTime` in [8a](#).

Uses: `AJAX_Time` [8cf](#).

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

```

< Global data: Ada 8e > ≡
  type Date_Format is (ISO, US, European, Locale);
  Current_Date_Format : Date_Format := ISO;

```

Fragment defined by [8e](#), [12b](#), [16b](#).

Fragment referenced in [2](#).

Users: `Current_Date_Format` in [8f](#), [10](#), [22d](#), `Date_Format` in [10](#).

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 8f > ≡

  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
    Result.Set_Content_Type (To => Types.Plain);
    Result.Set_Content
        (GNAT.Calendar.Time_IO.Image (Ada.Calendar.Clock, Format));
    return Result;
end AJAX_Time;

```

Fragment defined by [8f](#), [14c](#), [18e](#), [22d](#), [24d](#).

Fragment referenced in [2](#).

Users: [AJAX_Time](#) in [8d](#).

Uses: [Current_Date_Format](#) [8e](#).

4.2 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 9a) \equiv

```

<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 [7a](#).

Users: `fTimeFormat` in [9b](#), [21](#).

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 9b) \equiv

```

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 [28](#).

Uses: `fTimeFormat` [9a](#), `postChange` [24a](#).

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 10) \equiv

```
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 [10](#), [13a](#), [16c](#).

Fragment referenced in [24d](#).

Uses: `Current_Date_Format` [8e](#), `Date_Format` [8e](#).

Chapter 5

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 5.1: Using radio buttons

Figure 5.1 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 11) \equiv

```
<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 13a > ≡
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 10, 13a, 16c.

Fragment referenced in 24d.

Uses: Aft_Light 12b, Forward_Light 12b, Light_State 12b, aft-light 11, forward-light 11.

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 13b > ≡
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 8a, 13b.

Fragment referenced in 28.

Users: `lightStateRequest` in 13c.

Uses: aft-light 11, forward-light 11, lightState.xml 14b.

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

```

< Start getting cyclic data: JavaScript 13c > ≡
  lightStateRequest.start();

```

Fragment defined by 8b, 13c.

Fragment referenced in 28.

Uses: `lightStateRequest` 13b.

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

< Specs of dynamic pages: Ada 14a > ≡

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

◇

Fragment defined by 8c, 14a, 18c, 22b, 24b.

Fragment referenced in 2.

Users: AJAX_Light_State in 14b.

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

< Register dynamic pages: Ada 14b > ≡

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

◇

Fragment defined by 8d, 14b, 18d, 22c, 24c.

Fragment referenced in 2.

Users: lightState.xml in 13b.

Uses: AJAX_Light_State 14ac.

< Bodies of dynamic pages: Ada 14c > ≡

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

◇

Fragment defined by 8f, 14c, 18e, 22d, 24d.

Fragment referenced in 2.

Users: AJAX_Light_State in 14b.

Uses: Aft_Light 12b, Forward_Light 12b, aft-light 11, forward-light 11, lights 11.

Chapter 6

Checkboxes

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

Figure 6.1: Using checkboxes

Figure 6.1 shows the part of the example that relates to checkboxes. There are two Lamps, Port and Starboard, which are separately switched.

< Checkboxes: HTML 15 > \equiv

```
<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 [27c](#).

Users: `lamp` in [16ac](#), [21](#), [22d](#), `lamps` in [16a](#), [21](#).

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 16a > ≡
  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 [28](#).

Uses: `lamp` [15](#), `lamps` [15](#), `postChange` [24a](#).

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

```
< Global data: Ada 16b > ≡
  Lamps : array (0 .. 1) of Boolean := (others => True);
```

◇

Fragment defined by [8e](#), [12b](#), [16b](#).

Fragment referenced in [2](#).

Users: `Lamps` in [16c](#), [22d](#).

```
< Checks for changed properties: Ada 16c > ≡
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 [10](#), [13a](#), [16c](#).

Fragment referenced in [24d](#).

Uses: `Lamps` [16b](#), `lamp` [15](#).

Chapter 7

File upload

Figure 7.1 shows the file upload dialog. Figure 7.2 shows the alert box displayed when the upload is complete.

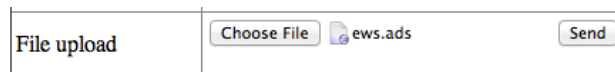


Figure 7.1: The file upload dialog

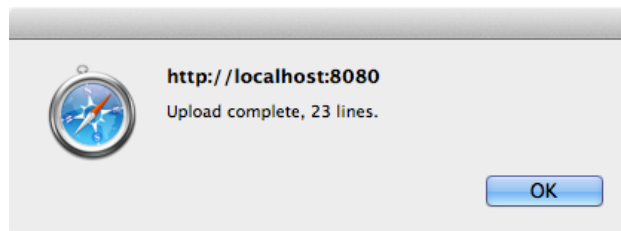


Figure 7.2: Upload completed

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 use 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 18a > ≡

```

<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 27c.

Uses: fileInput 18d, iFrame 18b.

< File upload's target iFrame: HTML 18b > ≡

```

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

```

Fragment referenced in 27c.

Users: iFrame in 18a.

The Ada code which receives the fileInput request is in the function File_Input.

< Specs of dynamic pages: Ada 18c > ≡

```

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

```

Fragment defined by 8c, 14a, 18c, 22b, 24b.

Fragment referenced in 2.

Users: File_Input in 18d.

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

< Register dynamic pages: Ada 18d > ≡

```

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

```

Fragment defined by 8d, 14b, 18d, 22c, 24c.

Fragment referenced in 2.

Users: fileInput in 18ae.

Uses: File_Input 18ce.

The response generated is a page containing an alert box (7.2), which is displayed even though the iFrame swallows the HTML content.

< Bodies of dynamic pages: Ada 18e > ≡

```

function File_Input
  (From_Request : HTTP.Request_P) return Dynamic.Dynamic_Response'Class is
  <Upload_Result, calculates file input result: Ada 19, ...>
  C : HTTP.Cursor;
  Lines : Natural := 0;
  Line : String (1 .. 1024);
  Last : Natural;
  Attachments : constant HTTP.Attachments
    := HTTP.Get_Attachments (From_Request.all);
  Content : constant HTTP.Contents
    := HTTP.Get_Content (From => Attachments);
begin

```

```

Put_Line ("saw fileInput with attachment length"
          & Content'Length'Image);
if Content'Length /= 0 then
begin
  case HTTP.Get_Content_Kind (Content) is
    when HTTP.Text =>
      HTTP.Open (C, Content);
      while not HTTP.End_Of_File (C) loop
        Lines := Lines + 1;
        Put (Lines'Image & ": ");
        HTTP.Get_Line (C, Line, Last);
        Put_Line (Line (Line'First .. Last));
      end loop;
      HTTP.Close (C);
      return Upload_Result
        ("Upload complete," & Lines'Image & " lines.");
    when others =>
      return Upload_Result
        ("Upload complete," & Content'Length'Image & " bytes.");
  end case;
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
  return Upload_Result ("Upload complete, zero bytes.");
end if;
end File_Input;

```

◇

Fragment defined by [8f](#), [14c](#), [18e](#), [22d](#), [24d](#).

Fragment referenced in [2](#).

Users: `File_Input` in [18d](#).

Uses: `Upload_Result` [19](#), [20](#), `fileInput` [18d](#).

$\langle \text{Upload_Result, calculates file input result: Ada 19} \rangle \equiv$

```

function Upload_Result (Message : String)
  return Dynamic.Dynamic_Response'Class;

```

◇

Fragment defined by [19](#), [20](#).

Fragment referenced in [18e](#).

Users: `Upload_Result` in [18e](#).

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 20⟩ ≡
function Upload_Result (Message : String)
  return Dynamic.Dynamic_Response'Class is
    Result : Dynamic.Dynamic_Response (From_Request);
begin
  Result.Set_Content_Type (To => Types.HTML);
  Result.Append ("<body onload=""alert('");
  for C in Message'Range loop
    case Message (C) is
      when ASCII.CR | ASCII.NUL => null;
      when ASCII.LF => Result.Append ("\n");
      when others =>
        Result.Append (String'(1 => Message (C)));
    end case;
  end loop;
  Result.Append ("')"">");
  return Result;
end Upload_Result;

```

◇
 Fragment defined by [19](#), [20](#).
 Fragment referenced in [18e](#).
 Users: Upload_Result in [18e](#).

Chapter 8

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 21*⟩ ≡

```
var stateRequest = new OneshotHttpRequest
("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 28.

Users: `stateRequest` in 22a.

Uses: `aft-light` 11, `fTimeFormat` 9a, `forward-light` 11, `lamp` 15, `lamps` 15, `lights` 11, `state.xml` 22c.

When the page is opened, request the initial state.

< Request the initial state: JavaScript 22a > ≡
`stateRequest.start();`

◇

Fragment referenced in 28.

Uses: `stateRequest` 21.

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

< Specs of dynamic pages: Ada 22b > ≡
`function AJAX_Status`
`(From_Request : HTTP.Request_P)`
`return Dynamic.Dynamic_Response'Class;`

◇

Fragment defined by 8c, 14a, 18c, 22b, 24b.

Fragment referenced in 2.

Uses: `AJAX_Status` in 22c.

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

< Register dynamic pages: Ada 22c > ≡
`Dynamic.Register (AJAX_Status'Unrestricted_Access, "/state.xml");`

◇

Fragment defined by 8d, 14b, 18d, 22c, 24c.

Fragment referenced in 2.

Uses: `state.xml` in 21.

Uses: `AJAX_Status` 22bd.

< Bodies of dynamic pages: Ada 22d > ≡

```

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

```

```

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

```

Fragment defined by [8f](#), [14c](#), [18e](#), [22d](#), [24d](#).

Fragment referenced in [2](#).

Users: [AJAX_Status](#) in [22c](#).

Uses: [Aft_Light](#) [12b](#), [Current_Date_Format](#) [8e](#), [Forward_Light](#) [12b](#), [Lamps](#) [16b](#), [aft-light](#) [11](#), [forward-light](#) [11](#), [lamp](#) [15](#).

Chapter 9

Utilities

9.1 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 24a > ≡  
var postChange = new OneshotHttpRequest  
  ("ajaxChange",  
   null,  
   function (r) { });
```

◇

Fragment referenced in 28.

Users: `postChange` in 9b, 16a, 25.

Uses: `ajaxChange` 24c.

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

```
< Specs of dynamic pages: Ada 24b > ≡  
function AJAX_Change  
  (From_Request : HTTP.Request_P)  
  return Dynamic.Dynamic_Response'Class;
```

◇

Fragment defined by 8c, 14a, 18c, 22b, 24b.

Fragment referenced in 2.

Users: `AJAX_Change` in 24c.

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

```
< Register dynamic pages: Ada 24c > ≡  
Dynamic.Register (AJAX_Change'Unrestricted_Access, "/ajaxChange");
```

◇

Fragment defined by 8d, 14b, 18d, 22c, 24c.

Fragment referenced in 2.

Users: `ajaxChange` in 24a.

Uses: `AJAX_Change` 24bd.

```
< Bodies of dynamic pages: Ada 24d > ≡  
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.");
```

```

    < Checks for changed properties: Ada 10, ... >
    Result.Set_Content_Type (To => Types.Plain);
    Result.Set_Content ("OK");
    return Result;
end AJAX_Change;

```

Fragment defined by 8f, 14c, 18e, 22d, 24d.
 Fragment referenced in 2.
 Users: AJAX_Change in 24c.

9.2 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 25 > ≡
  for (var j = 0; j < @1.length; j++) {
    @1[j].onclick = new Function(
      "postChange.start(' " + @2 + "=" + @1[j].value + "');"");
  };

```

Fragment referenced in 12a.
 Uses: `postChange` 24a.

Chapter 10

HTML pages

```
"ajax.html" 26a≡
< HTML licence header 6 >
<html>
<head>
< HTML header 26b >
</head>
<body bgcolor="white">
< Page heading 27a >
< Introductory material 27b >
<p><hr>
< The demonstrations 27c >
< Author link 27d >
</body>
</html>
◇

< HTML header 26b > ≡
<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 26a.
```

```

< Page heading 27a > ≡
<table width="100%">
<tr>
<td><h1>Embedded Web Server: AJAX demonstration</h1></td>
<td align="right">
<a href="https://github.com">

</a>
</td>
</tr>
</table>

```

Fragment referenced in 26a.

```

< Introductory material 27b > ≡
<p>EWS is a web server construction kit, designed for embedded
applications using the GNAT Ada compiler.

<p>The project is hosted on <a
href="https://github.com/simonjwright/ews">Github</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://github.com/simonjwright/nuweb.py">nuweb.py</a>.

```

Fragment referenced in 26a.

```

< The demonstrations 27c > ≡
<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>< i>Cyclic updating and select/options: HTML 7a</i>
  </td>
  <td>< i>Radio buttons: HTML 11</i>
  </td>
  <td>< i>Checkboxes: HTML 15</i>
  </td>
  <td>< i>File upload: HTML 18a</i>
  </td>
  </tr>
</table>
</div>
< i>File upload's target iFrame: HTML 18b</i>

```

Fragment referenced in 26a.

```

< Author link 27d > ≡
<hr>
<i>
<address>
<a href="mailto:simon@pushface.org">Simon Wright</a>
</address>
</i>

```

Fragment referenced in 26a.

Chapter 11

JavaScript

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

```
"ajax.js" 28≡  
  ⟨ JavaScript licence header 5c ⟩  
  
  ⟨ Retrieving the initial state: JavaScript 21 ⟩  
  
  ⟨ Cyclic interactions: JavaScript 8a, ... ⟩  
  
  ⟨ Generalised change action request: JavaScript 24a ⟩  
  
  /**  
   * Assign event handlers and begin fetching.  
   */  
  window.onload = function () {  
  
    ⟨ Request the initial state: JavaScript 22a ⟩  
    ⟨ Start getting cyclic data: JavaScript 8b, ... ⟩  
    ⟨ Set up to send time format: JavaScript 9b ⟩  
    ⟨ Set up the radio buttons: JavaScript 12a ⟩  
    ⟨ Set up the checkboxes: JavaScript 16a ⟩  
  };  
  ◇
```

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 [26a](#).

"[ajax.js](#)" Defined by [28](#).

"[ews_demo.adb](#)" Defined by [2](#).

"[ews_demo.gpr](#)" Defined by [4](#).

B.2 Macros

- [〈"Set up radio buttons" utility: JavaScript 25〉](#) Referenced in [12a](#).
- [〈Ada licence header 5b〉](#) Referenced in [2](#).
- [〈Author link 27d〉](#) Referenced in [26a](#).
- [〈Bodies of dynamic pages: Ada 8f, 14c, 18e, 22d, 24d〉](#) Referenced in [2](#).
- [〈Change the time display format: HTML 9a〉](#) Referenced in [7a](#).
- [〈Checkboxes: HTML 15〉](#) Referenced in [27c](#).
- [〈Checks for changed properties: Ada 10, 13a, 16c〉](#) Referenced in [24d](#).
- [〈Copyright 5a〉](#) Referenced in [5bc, 6](#).
- [〈Cyclic interactions: JavaScript 8a, 13b〉](#) Referenced in [28](#).
- [〈Cyclic updating and select/options: HTML 7a〉](#) Referenced in [27c](#).
- [〈Display the current time: HTML 7b〉](#) Referenced in [7a](#).
- [〈File upload's target iFrame: HTML 18b〉](#) Referenced in [27c](#).
- [〈File upload: HTML 18a〉](#) Referenced in [27c](#).
- [〈Generalised change action request: JavaScript 24a〉](#) Referenced in [28](#).
- [〈Global data: Ada 8e, 12b, 16b〉](#) Referenced in [2](#).
- [〈HTML header 26b〉](#) Referenced in [26a](#).
- [〈HTML licence header 6〉](#) Referenced in [26a](#).
- [〈Introductory material 27b〉](#) Referenced in [26a](#).
- [〈JavaScript licence header 5c〉](#) Referenced in [28](#).
- [〈Main program generated context: Ada 3b〉](#) Referenced in [2](#).
- [〈Main program standard context: Ada 3a〉](#) Referenced in [2](#).
- [〈Page heading 27a〉](#) Referenced in [26a](#).
- [〈Radio buttons: HTML 11〉](#) Referenced in [27c](#).
- [〈Register dynamic pages: Ada 8d, 14b, 18d, 22c, 24c〉](#) Referenced in [2](#).
- [〈Request the initial state: JavaScript 22a〉](#) Referenced in [28](#).
- [〈Retrieving the initial state: JavaScript 21〉](#) Referenced in [28](#).
- [〈Set up the checkboxes: JavaScript 16a〉](#) Referenced in [28](#).
- [〈Set up the radio buttons: JavaScript 12a〉](#) Referenced in [28](#).
- [〈Set up to send time format: JavaScript 9b〉](#) Referenced in [28](#).
- [〈Specs of dynamic pages: Ada 8c, 14a, 18c, 22b, 24b〉](#) Referenced in [2](#).
- [〈Start getting cyclic data: JavaScript 8b, 13c〉](#) Referenced in [28](#).

⟨ The demonstrations 27c ⟩ Referenced in 26a.
⟨ Upload_Result, calculates file input result: Ada 19, 20 ⟩ Referenced in 18e.

B.3 Definitions

AJAX_Change: defined in 24bd, used in 24c.
AJAX_Light_State: defined in 14ac, used in 14b.
AJAX_Status: defined in 22bd, used in 22c.
AJAX_Time: defined in 8cf, used in 8d.
Aft_Light: defined in 12b, used in 13a, 14c, 22d.
Current_Date_Format: defined in 8e, used in 8f, 10, 22d.
Date_Format: defined in 8e, used in 10.
EWS_Demo: defined in 2, used in 4.
File_Input: defined in 18ce, used in 18d.
Forward_Light: defined in 12b, used in 13a, 14c, 22d.
Lamps: defined in 16b, used in 16c, 22d.
Light_State: defined in 12b, used in 13a.
Upload_Result: defined in 19, 20, used in 18e.
aft-light: defined in 11, used in 13ab, 14c, 21, 22d.
ajaxChange: defined in 24c, used in 24a.
ajaxTime: defined in 8d, used in 8a.
fTimeFormat: defined in 9a, used in 9b, 21.
fileInput: defined in 18d, used in 18ae.
forward-light: defined in 11, used in 13ab, 14c, 21, 22d.
iFrame: defined in 18b, used in 18a.
lamp: defined in 15, used in 16ac, 21, 22d.
lamps: defined in 15, used in 16a, 21.
lightState.xml: defined in 14b, used in 13b.
lightStateRequest: defined in 13b, used in 13c.
lights: defined in 11, used in 14c, 21.
postChange: defined in 24a, used in 9b, 16a, 25.
state.xml: defined in 22c, used in 21.
stateRequest: defined in 21, used in 22a.
timeDisplay: defined in 7b, used in 8a.
timeRequest: defined in 8a, used in 8b.