

# WASH/CGI

Dynamic web pages with Haskell

Xavier Pardonnet  
Martin Matusiak

Wednesday 21.03.07



Introducing WASH/CGI

Diving in

A more substantial example

Concluding remarks



## Common Gateway Interface

- ▶ The first technology (1993) for dynamic web content, by using dynamic **scripts** in place of static **pages**.
- ▶ Defines the way a user application can interact with the web server to provide dynamic content.
- ▶ Scripts typically written in c (at first) or perl (commonly).

*The way CGI works from the Web server's point of view is that certain locations (e.g. `http://www.example.com/wiki.cgi`) are defined to be served by a CGI program. Whenever a request to a matching URL is received, the corresponding program is called, with any data that the client sent as input. Output from the program is collected by the Web server, augmented with appropriate headers, and sent back to the client.*

- Wikipedia



## A CGI scripting framework in Haskell

- ▶ WASH/CGI is an **embedded DSL**,
- ▶ with a rich interface for **programming** CGI scripts **as applications**.
- ▶ A **single point of entry** approach sidesteps the common issue of **argument passing** between scripts,
- ▶ so form processing is **encapsulated**.
- ▶ HTML documents treated in a **structured** way, giving **valid** code.

Created by **Peter Thiemann** (University of Freiburg, Germany).



## A CGI scripting framework in Haskell

- ▶ CGI is superbly **supported** and **portable** (albeit binaries must be recompiled).
- ▶ Support for **session handling** and typed input **validation**.
- ▶ A lean and mean library? **No**. 79 modules, 28k lines of code
  - CGI
  - HTML generation
  - Email dispatch
  - DB connectivity



## Matthias Neubauers Lieblingsrezept

**Griessbrei ist das Beste überhaupt!!!**

### Zutaten

1 Liter	Milch
175 Gramm	Griess
3 Essloeffel	Zucker
1 Priesse	Salz
1	unbehandelte Zitrone
nach belieben	Rosinen

70 lines of Haskell code, 70 lines of HTML  
XHTML 1.0 Transitional – VALID



# The first demo : view source

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
3 ><!-- generated by WASH/HTML 0.11
4 --><html xmlns="http://www.w3.org/1999/xhtml"><head><title>Matthias
5   Neubauers Lieblingsrezept</title>
6 ></head>
7 ><body><h1>Matthias Neubauers Lieblingsrezept</h1>
8 ><form enctype="application/x-www-form-urlencoded" target="_self"
9   onsubmit="return OnSubmit();" method="post"
10  action="/cgi-bin/WASH/bin/ex1"><h2 style="color: yellow;
11  background: red; ">Griessbrei ist das Beste uberhaupt!!!</h2>
12 ><h3 style="color: blue; background: yellow; ">Zutaten</h3>
13 ><table cellpadding="2" border="1"><tr><td style="color: yellow;
14  background: blue; ">1 Liter</td>
15 ><td style="color: red; background: blue; ">Milch</td>
16 ></tr>
17 ><tr><td style="color: yellow; background: blue; ">175 Gramm</td>
18 ><td style="color: red; background: blue; ">Griess</td>
19 ></tr>
20 ><tr><td style="color: yellow; background: blue; ">3 Essloeffel</td>
21 ><td style="color: red; background: blue; ">Zucker</td>
22 ></tr>
23 <!-- snip -->
```



## WASH/CGI at work

1. **My eyes!!!**
2. WASH/CGI doesn't **prevent** me from creating horrid pages. :(
3. **Styles** should have been declared once and reused.
4. All pages come pre-wrapped in a **form**, whether dynamic or not.  
\*suspicious\* All forms submitted with **POST**, ie. forward/back buttons in browser rendered useless.
5. Document is **well formed** and **valid**, as promised.
6. Output could use pretty printing.



Introducing WASH/CGI

Diving in

A more substantial example

Concluding remarks



# Hello World

```
1 module Main where
2
3 import WASH.CGI.CGI hiding (div, head, map, span)
4
5 first :: CGI ()
6 first = ask $ standardPage "Hello World" $ (p $ text "Ma, I'm on tv!")
7
8 main = run $ first
```

## Hello World

Ma, I'm on tv!

Hm, that wasn't too bad.



```
5 second :: CGI ()
6 second = ask $ standardPage "Second" $ do
7     p (text "Second page yay")
8     p $ do
9         text "A list"
10        ul $ do
11            li (text "Steve")
12            li (text "Stevie")
13            li (text "Steve-O")
```

## Second

Second page yay

A list

- Steve
- Stevie
- Steve-O



# Let's add some styles!

```
5 fgRed = "color" ::= "red"
6 bgGreen = "background" ::= "yellow"
7 styleImportant = fgRed ^: bgGreen
8
9 important = using styleImportant
10
11 styled :: CGI ()
12 styled = ask $ standardPage "Styled" $
13     important p (text "Important")
```

## Styled

Important



# A simple form

```
5 greeting :: String -> CGI ()
6 greeting name =
7     standardQuery "Hello" $ do
8         text "Hello "
9         text name
10        text ". This is my first interactive CGI program!"
11
12 main = run $ standardQuery "What's your name?" $
13     p $ do
14         text "Hi there! What's your name? "
15         activate greeting textInputField empty
```

## What's your name?

Hi there! What's your name?

## Hello

Hello Terry. This is my first interactive CGI program!



Introducing WASH/CGI

Diving in

**A more substantial example**

Concluding remarks



## AntWars

Enter ant files to upload:

NOTE: Simulation may take a few minutes.

## Battle results

### Home

xavier	203
bastiaan_sample	0

### Away

bastiaan_sample	0
xavier	337

### Overall

xavier	540
bastiaan_sample	0



# Under the hood 1

```
7 main =
8   run $
9   standardQuery "AntWars" $
10  do text "Enter ant files to upload: "
11     fileA <- checkedFileInputField refuseUnnamed empty
12     fileB <- checkedFileInputField refuseUnnamed empty
13     submit (F2 fileA fileB) display (fieldVALUE "To arms!")
14     text "NOTE: Simulation may take a few minutes."
15
16 refuseUnnamed :: FileReference -> FileReference
17 refuseUnnamed mf =
18   do FileReference {fileReferenceExternalName=frn} <- mf
19     if null frn then fail "" else mf
```

- ▶ fileA, fileB : **file input** fields that refuse empty input
- ▶ submit : creates the submit button, takes as arguments the **arguments** passed on from this form, the **action** to be performed, and **attributes** of the HTML tag
- ▶ F2 : FX are simple constructors for passing on the given number of arguments



## Under the hood 2

```
21 display :: F2 (InputField FileReference) (InputField FileReference)
22     VALID -> CGI ()
23 display (F2 fileA fileB) =
24     let fileRefA = value fileA
25         fileRefB = value fileB
26         pathA = fileReferenceName fileRefA
27         nameA = fileReferenceExternalName fileRefA
28         pathB = fileReferenceName fileRefB
29         nameB = fileReferenceExternalName fileRefB
30     in do
31         io $ mv pathA nameA
32         io $ mv pathB nameB
33         runbattle nameA nameB
34
35 mv :: FilePath -> String -> IO ()
```

- ▶ `display` : receive two `FileReferences`, extract their paths and move them to where we want them. Then call `runbattle` to run the simulation.
- ▶ `value` : extract the value of an `InputField` into its type



# Under the hood 3

```
17 runbattle :: String -> String -> CGI ()
18 runbattle a b = do
19     res <- io $ runsim a b
20     ask $ simpage res
21
22 simpage res = standardPage "Battle results"
23     (do displaymatch "Home" (fst res)
24         displaymatch "Away" (snd res)
25         displaybattle "Overall" res)
```

- ▶ `runbattle` : accepts two filepaths, runs the simulator and passes the scores to `simpage`
- ▶ `runsim` : the function which runs the simulation (omitted as it does not concern WASH/CGI)
- ▶ `io` : our get-out-of-jail card, lifts the IO computation to the CGI monad



# Under the hood 4

```
38 displaymatch match (rounds, world,  
39   (Red, redteam, redscore),  
40   (Black, blackteam, blackscore)) = do  
41   table $ do  
42     tr $ th $ do attr "colspan" "2"  
43         text match  
44     tr $ do (td $ text redteam)  
45             (td $ text $ show redscore)  
46     tr $ do (td $ text blackteam)  
47             (td $ text $ show blackscore)
```

- ▶ `displaymatch` : builds up the table to display the scores from one match.
- ▶ `attr` : sets an attribute for a tag, usually the first computation after the tag has been opened



Introducing WASH/CGI

Diving in

A more substantial example

**Concluding remarks**



## Flaws and problems

- ▶ **Styles** are a complete afterthought. WASH/CGI neither does encourage nor provide for a structured application of styles onto pages.
- ▶ Internal **types** are very complicated, making code hard to debug (this aspect omitted from slides on purpose). The library is so large that looking up definitions is not trivial either.
- ▶ **No document** found in sync with current library version (`WashNGo-2.10`), making it hard to follow outdated examples and **changing types**.
- ▶ Current library version makes it entirely possible to produce **invalid** documents (eg. nesting `table` under `ul` is not prevented).



# Is the syntax convenient?

The syntax leaves us wanting. One would like to be able to write the following, which is congruent with HTML, rather than the latter. WASH/CGI obfuscates clear HTML structure.

```
1 table
2   (tr (th (colspan 2)
3         match))
4   (tr (td redteam)
5       (td $ show redscore))
6   (tr (td blackteam)
7       (td $ show blackscore))
```

```
10 table $ do
11   tr $ th $ do attr "colspan" "2"
12             text match
13   tr $ do (td $ text redteam)
14           (td $ text $ show redscore)
15   tr $ do (td $ text blackteam)
16           (td $ text $ show blackscore)
```



# Is it practical?

## Issues

- ▶ **CGI** is portable, but deploying **binaries** is not,
- ▶ and webhosts **do not trust** binaries for fear of malware.
- ▶ The **learning curve** of WASH/CGI is such that it takes a [decent] Haskell coder to use it.
- ▶ From our modest research and use, we **cannot** see how embedding WASH/CGI in Haskell gives substantial benefit toward building websites in a functional manner.
- ▶ CGI is basically **deprecated**.

