IGB

Leibniz-Institute of
Freshwater Ecology
and Inland Fisheries

Run Perl in the Browser with WebPerl!

Hauke Dämpfling (haukex), IGB Berlin

08.08.2019

PerlCon 2019 in Rīga, Latvia

These slides are available at: goo.gl/QPvUb9

Outline

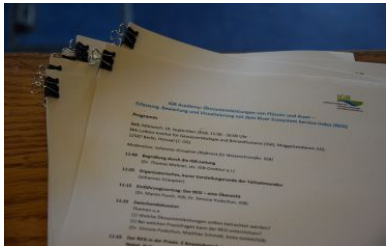
WebPerl

- Introduction
- Technical Background
- Architecture, API, Environment
- Pros & Cons, Status
- Perl 6 Support
- Examples

Research for the Future of our Freshwaters

IGB is Germany's largest, and one of the leading international research centres for freshwaters.

www.igb-berlin.de



Research – crossing borders, bridging disciplines

Broad range of topics: basic research on freshwaters and aquatic organisms; impacts of land use, climate change and pollution on freshwaters; conservation of freshwater biodiversity; sustainable aquaculture and fisheries.

Promote – dedicated teams, international perspectives

We are actively involved in teaching: international master's programme in Fish Biology, Fisheries and Aquaculture at the Humboldt-Universität zu Berlin; 10 joint professorships with 4 Universities.

Share – objective information, open exchange

One of IGB's core tasks is to provide science-based consulting to society's stakeholders, and information to the interested public.

Many Thanks To:

- My employer, the IGB Berlin
- Open Source Developers
  - Larry Wall, the Perl 5 Porters, and the Perl 6 developers
  - The Emscripten Team
  - Paweł Murias (Rakudo.js, 6pad)
  - Prior Art: A few people have compiled *microperl* to JS
    - Harsha: https://github.com/moodyharsh/plu
    - Shlomi Fish: https://github.com/shlomif/perl5-for-JavaScript--take2
    - FUJI Goro: https://github.com/gfx/perl.js

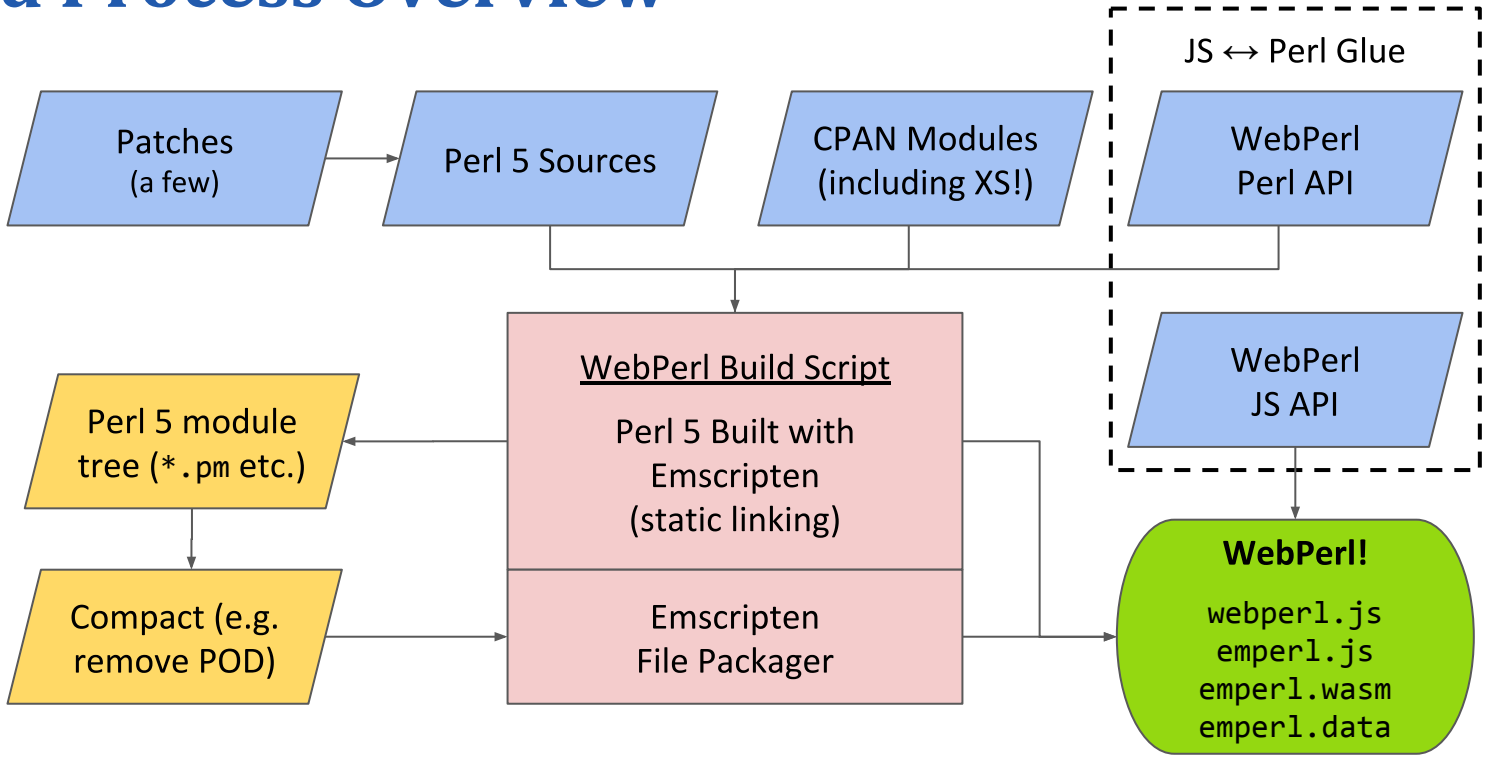
Motivation

```
<html>
<head>
<title>Hello, Perl World!</title>
<script src="webperl.js"></script>
<script type="text/perl">
print "Hello, Perl World!\n";
</script>
</head>
<body>
</body>
</html>
```

Technical Background

- **JavaScript** (1995)
  - Steady progress in standardization, adoption, optimization, etc.
- **`asm.js`** (2013; asmjs.org)
  - Strict subset of JS that supporting browsers can compile and optimize, and therefore run faster than JS (example use: http://bellard.org/jslinux)
- **WebAssembly** (2017; webassembly.org and MDN web docs)
  - Bytecode format for browsers, targeting the same VM as JS
  - Unlike `asm.js`, does not require aligned memory access
- **Emscripten** (emscripten.org)
  - Compiler based on LLVM / `clang` to compile C/C++ to `asm.js` and WebAssembly
  - Provides virtual environment for C/C++ code (system calls, file system, ...)

Build Process Overview



Glue Code

- **`webperl.js`** (https://webperl.zero-g.net/using.html#webperljs)
  - Looks for `<script type="text/perl">` tags, and if found, `joins` and runs them, otherwise, you can use the JS "`Perl`" object to control the interpreter
  - Loads `emperl.js`, which loads `emperl.wasm` and `emperl.data` (async)
  - `STDOUT/ERR` goes to JS console by default, unless redirected (JS `Perl.output`)

- **`WebPerl.pm`** (https://webperl.zero-g.net/using.html#webperlpm)
  - Provides `js($javascript_code)` and `WebPerl::JSObject` to Perl
  - Uses `WebPerl.xs`, which includes JavaScript code to interact with `webperl.js`

API: `js()`

`js("javascript_code")`* return values		
JavaScript Value	⇒	**Perl Value**
`undefined`		`undef`
Booleans		`!0` / `!1`
Numbers and strings		copied to Perl as numbers and strings
`function`s, objects (hashes), and arrays		“reference” is wrapped in `WebPerl::JSObject` proxy objects

* `js( [1,2,3] )` and `js( {foo=>"bar"} )` is also supported: deep copies Perl to JS, creates a new JS object, and returns a `WebPerl::JSObject`

API: WebPerl::JSObject

Perl Proxy Object ⇒	JavaScript Code
`$jso->{"bar"}`	`obj["bar"]`
`$jso->[42]`	`obj[42]`
`$jso->("arg", ...)`	`obj("arg", ...)`
`$jso->bar("arg", ...)`	`obj.bar("arg", ...)`
`$jso->methodcall( "can", "arg", ...)`	`obj.can("arg", ...)`
`$jso->toperl()`	deep copy of JS data structure to Perl, with JS `function`s as Perl coderefs

`WebPerl::JSObject`s are memory-managed via Perl's `DESTROY`,
so JS can garbage-collect its objects as appropriate

API: JS ↔ Perl

A new JS object is created and a `WebPerl::JSObject` is returned to Perl

(currently) deep copied to JS using `Cpanel::JSON::XS`

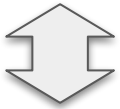
```
my $jsobject = js( {hello=>"world"} );

js("function (a,b,c,d,e) {}")->( "foo", [1,2,3], {bar=>42},
    sub {}, $jsobject );
```

JS gets the original JS object

- JS gets a `function` that, when called, calls the Perl `sub`
- Arguments and return values to/from the `sub` are supported
- **Important:** Because JS doesn't have equivalent of `DESTROY`, *anonymous* subs passed from Perl to JS must be **explicitly freed** using `WebPerl::unregister($coderef)`!

Example

```
<script>
    document.getElementById('my_button')
        .addEventListener('click', function () {
            window.alert("You clicked the button!");
        } );
</script>
```



```
<script src="webperl.js"></script>
<script type="text/perl">
    js('document')->getElementById('my_button')
        ->addEventListener('click', sub {
            js('window')->alert("You clicked the button!");
        } );
</script>
```

The Perl Interpreter's Environment

- **Single-process environment**: No `system`, backticks (`qx`), piped `open`, `fork`, multithreading, `kill`, `wait`, `waitpid`, signals (except `SIGALRM`), and related functions

- Currently **no blocking I/O**, because it would block the browser
  - Virtual FS works fine, but not `<STDIN>` or blocking network I/O
  - Workarounds may be possible, I'm thinking about it `:-)`

- Emscripten provides a **virtual file system** that resembles a *NIX FS
  - Is fixed when WebPerl is built, lives only in the browser's memory, all changes are lost
  - WebPerl mounts a special IndexedDB file system at `/mnt/idb` *(see docs for details!)*
  - Sandboxed: to access the user's files, use the browser's file upload / download features, or use HTTP calls to the web server to access files there

Perl Interpreter Lifecycle

```
#!/usr/bin/perl
END { ... }
while (<>) {
    chomp;
    ...
    print;
}
exit(0);
__END__
```

```
use Tk;
$mw = MainWindow->new;
$mw->Button(
    -text => "Close",
    -command => sub {
        $mw->destroy();
    } )->pack();
MainLoop;
__END__
```

```
use WebPerl;
END { ... }
...->addEventListener(
    'click', sub {
        ...
    } );
exit(0);
__END__
```

1. Interpreter shuts itself down, `END` blocks run, global destruction
2. C `main()` ends
3. Process ends

1. Interpreter does **not** shut itself down
2. C `main()` ends, process is “suspended”
3. Control returns to browser’s main loop

- Browser window can be closed **anytime**
- You could end the Perl interpreter with `WebPerl::end_perl()`, but it can’t be (easily) re-started

Advantages & Disadvantages

- **It's Perl!** `:-)`
- One language for server and client!
- Runs anywhere that WASM is supported (including `node.js`)
- Sandboxed Perl with XS support
- Can take full advantage of existing JavaScript frameworks (jQuery etc.), AJAX, etc.

➢ IMO, good for UIs / long-running single-page apps

- Sandboxed, single-process environment (no `fork`, `qx`, signals, blocking I/O, etc.)
- Fairly large download: currently ~4MB gzip compressed, 16MB uncompressed
- Not as fast as plain JS or native Perl
  - WebPerl is roughly 3-4x slower than native (Linux, Firefox)
  - Many strings copied back & forth
- Interpreter can only run once (workaround possible with `<iframe>`s)

Status

- WebPerl is still **beta** because it needs more tests!
  - The tests I have been able to run manually look ok, but:
  - Running Perl's core test suite directly is very difficult due to WebPerl's limitations: single-process environment, but many tests use `qx` (e.g. `runperl` in `t/test.pl`), intermixed with tests that don't require `qx`
- Many ideas, not enough time `;-)`
  - https://webperl.zero-g.net/notes.html#todos
  - Solidify Perl 6 integration
  - Support for Web Workers → possible solution for test suite and maybe even blocking I/O?
- Your input is very welcome!

Perl 6 Support

- **Rakudo.js** is a JavaScript backend in the Rakudo compiler, by Paweł Murias et al.
  - **It transpiles Perl 6 to JavaScript** (does not use WebAssembly)
  - Rakudo is written in NQP (quasi-subset of Perl 6), so it can transpile itself to JS
- **WebPerl's support** is experimental and must be patched in (see the "Quick Start")
  - It's currently a direct copy of the Rakudo.js build from "6pad" by Paweł Murias
  - You can build your own, see Rakudo's "`js`" backend and these links
- WebPerl provides **Perl 6 functionality** similar to Perl 5:
  - `<script type="text/perl6">` tags are run automatically,
  - or you can control the interpreter via the JS "`Raku`" object
- **Brief Example:**

```
my $window = EVAL(:lang<JavaScript>, 'return window');
$window.alert("Hello, World!");
```

Example: Getting Started

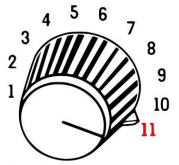
https://webperl.zero-g.net/#quick-start

https://webperl.zero-g.net/perl6.html#quick-start

```
$   git clone https://github.com/haukex/webperl.git && cd webperl
$   wget https://github.com/haukex/webperl/releases/\
    download/v0.09-beta/webperl_prebuilt_v0.09-beta.zip
$   unzip -j webperl_prebuilt_v0.09-beta.zip '*/emperl.*' -d web
$   cpanm --installdeps .
$   plackup web/webperl.psgi &
$   x-www-browser http://localhost:5000/webperl_demo.html
```

More Examples

- Building your own WebPerl to add more CPAN modules
  - https://webperl.zero-g.net/building.html
  - Build is almost entirely automated via a script
- WebPerl Regex Tester (written in Perl)
  - https://webperl.zero-g.net/regex.html
- WebPerl Embeddable Code Demo Editor
  - https://webperl.zero-g.net/democode/index.html
- Perl 6
  - Perl 5 and Perl 6 calling each other through JavaScript: https://github.com/haukex/webperl/blob/fe8e030/experiments/p6/test6.html



perl11.org


http://webperl.zero-g.net/
Slides: goo.gl/QPvUb9
Questions?
Thank You!

More Information

- I’ve written some more about WebPerl on PerlMonks:
  - Run Perl 5 in the Browser!
  - WebPerl Regex Tester (beta)
  - Embedding WebPerl Code Demos into Other Pages
  - WebPerl now has Experimental Perl 6 Support!
- More info on Rakudo.js:
  - http://blogs.perl.org/users/pawel_murias/