Let’s Wub

Wub tutorial

jos.decoste@gmail.com
What is Wub?

• HTTP 1.1 Webserver
• Written by Colin McCormack
• Successor of tclhttpd
• 100% Tcl Web application framework
• Heavy user of recent Tcl features, which pushed development of Tcl 8.6 forward:
  – dicts
  – coroutines
  – tclOO
  – Zlib
• Domain based (packages grouping functionality)
Wub in action!

- Tcler’s wiki runs on Wub since 2007
  [http://wiki.tcl.tk](http://wiki.tcl.tk)
- Wubchain (web interface to Tcler’s chat)
- Intranet application to manage tool releases
Requirements?

- Tcl 8.6 (dicts, coroutines, tclOO, zlib)
- Tcllib 1.11
- Unix or Windows
Getting Wub

• Wub’s hosted at Google
  http://code.google.com/wub

• Releases at
  http://code.google.com/p/wub/downloads

• Checkout via SVN:
  ```
  svn checkout http://wub.googlecode.com/svn/trunk Wub
  ```

• Available together with required Tcllib modules as part of WubWikit
  http://code.google.com/p/wubwikit
Documentation

• Wub’s wiki page:
  http://wiki.tcl.tk/Wub

• Wub documentation:
  http://wiki.tcl.tk/_wub/

• This Tutorial
Tutorial goals

• Overview different parts of Wub
• Step-by-step introduction of File, Mason, Direct, Nub, Ini, jQuery, ... domains
• Wub API: utilities to generate HTML, cache, handle queries, convert response types, forms, ...
Tutorial examples

- Get examples at:
  
  http://code.google.com/p/wubwikit/downloads

- Run as:
  
  tclsh ex.tcl (make sure it can find tcllib and Wub)

  ..:/Wub and ..:/tcllib are added to auto_path by each example

- Point browser to:
  
  http://localhost:8080
Wub’s architecture
Wub’s architecture

- Httpd module converts HTTP client request into *request* dicts.
- Caching and blocking based on *request* dict
- *Request* dicts are transformed into *response* dicts by the web application using Wub utilities and domains.
- *Response* dicts are send back to the client by the Httpd module (protocol / non-protocol fields in dict)
Example 1: Wub out-of-the-box

• Running the default setup, start the webserver with these lines:

```package require Site
Site start home .```

• What’s being served?
  – Files from html, images, css and scripts subdirectories of ./docroot
Example 1 (continued)

• How are these file served:
  – On port 8080
  – Using caching
  – The html, images, css and scripts subdirectories are served as root in the URL

<table>
<thead>
<tr>
<th>On server</th>
<th>On client</th>
<th>Expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>./docroot/css/file.css</td>
<td><a href="http://myurl:8080/css/file.css">http://myurl:8080/css/file.css</a></td>
<td>tomorrow</td>
</tr>
</tbody>
</table>
Example 1a: adding some Nub

• Nub is a configuration utility for Wub

• Specify redirect from root URL to ./docroot/html/ex.html in ex.nub:

  redirect / /html/ex.html

• Start Wub with these lines:

  package require Site
  Site start home . nubs ex.nub
Example 1a: Nub redirect syntax

```
redirect <from url> <to url>
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>from url</td>
<td>URL for which to send a redirect.</td>
</tr>
<tr>
<td>to url</td>
<td>URL to redirect to</td>
</tr>
</tbody>
</table>
Example 2: the File domain

• Map URL’s to file system hierarchy

• Making File domains explicit in ex.nub:

```plaintext
domain /css/ {File css} root [file join . docroot css] expires tomorrow nodir 1
domain /images/ {File images} root [file join . docroot images] expires "next week" nodir 1
domain /scripts/ {File scripts} root [file join . docroot scripts] expires tomorrow nodir 1
domain /html/ {File html} root [file join . docroot html] expires tomorrow nodir 1
```

• Add new File domain:

```plaintext
domain /disk/ {File disk} 
  root / 
  indexfile index.* 
  hide {^([.].*)|(.~)|(#.*$)} 
  redirdir 1 
  expires tomorrow 
  dateformat "%Y %b %d %T" 
  nodir 0
```

• Add Rewrite to ex.nub:

```plaintext
rewrite {/[\^/]\+[.].html} {/html/[file tail [dict get $r -path]]}
```
Example 2: Nub domain syntax

domain <url> <list of domain_name and name> <args>

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Url to be processed with the specified domain</td>
</tr>
<tr>
<td>domain_name</td>
<td>name of domain to use (File, Mason, Direct, jQ, …)</td>
</tr>
<tr>
<td>name</td>
<td>Name of this domain usage</td>
</tr>
<tr>
<td>args</td>
<td>Domain specific arguments</td>
</tr>
</tbody>
</table>
### Example 2: File domain arguments

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>File-system root directory of File domain.</td>
<td></td>
</tr>
<tr>
<td>indexfile</td>
<td>Name of the file which stands for a directory, such as index.html. The contents of that file will be returned in stead of the directory listing.</td>
<td>index.*</td>
</tr>
<tr>
<td>hide</td>
<td>A regular expression to hide temp and other uninteresting files (default hides .* <em>~ and #</em>).</td>
<td>^([.].*)</td>
</tr>
<tr>
<td>redirdir</td>
<td>Should references to directories be required to have a trailing /?</td>
<td>1</td>
</tr>
<tr>
<td>expires</td>
<td>A tcl clock expression indicating when contents expires.</td>
<td>0</td>
</tr>
<tr>
<td>dateformat</td>
<td>A tcl clock format for displaying dates in directory listings.</td>
<td>%Y %b %d %T</td>
</tr>
<tr>
<td>nodir</td>
<td>Don't allow the browsing of directories (default: 0 - browsing allowed).</td>
<td>0</td>
</tr>
</tbody>
</table>
Example 2: Nub rewrite and redirect syntax

```
rewrite <regexp> <script>
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>regexp</td>
<td>Regular expression to select an URL to be transformed</td>
</tr>
<tr>
<td>script</td>
<td>Script to be called to transform the URL (evaluated in Nub namespace)</td>
</tr>
</tbody>
</table>
Example 3: Mason domain

- A File like domain
- Mapping URL’s to file system hierarchy
- Provides templating by applying [subst] on .tml files
- Pre/post filtering of request and responses
- Adding Mason domain to ex.nub

```
domain /mason/ {Mason mason} root [file join . docroot mason]
```
### Example 3: Mason arguments

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>File-system root for Mason domain.</td>
<td></td>
</tr>
<tr>
<td>ctype</td>
<td>Default content type of returned values.</td>
<td>x-text/html-fragment</td>
</tr>
<tr>
<td>hide</td>
<td>Regular expresion to detect files to hide.</td>
<td>^([^].*)</td>
</tr>
<tr>
<td>indexfile</td>
<td>Name of the file which stands for a directory, such as index.html. The contents of that file will be returned in stead of the directory listing.</td>
<td>index.html</td>
</tr>
<tr>
<td>expires</td>
<td>A tcl clock expression indicating when contents expires.</td>
<td></td>
</tr>
<tr>
<td>functional</td>
<td>File extension indicating which files will be evaluated.</td>
<td>.tml</td>
</tr>
<tr>
<td>notfound</td>
<td>Template to be evaluated when requested file can’t be found.</td>
<td>.notfound</td>
</tr>
</tbody>
</table>
### Example 3: Mason arguments (cont.)

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>wrapper</td>
<td>Template to be evaluated with successful response.</td>
<td>.wrapper</td>
</tr>
<tr>
<td>auth</td>
<td>Template to be evaluated before processing requested file.</td>
<td>.auth</td>
</tr>
<tr>
<td>nodir</td>
<td>Don't allow the browsing of directory listings.</td>
<td>0</td>
</tr>
<tr>
<td>dateformat</td>
<td>A tcl clock format for displaying dates in directory listings.</td>
<td>%Y %b %d %T</td>
</tr>
</tbody>
</table>
Example 3: template file

- Is evaluated using \([\text{subst}]\)
- Result of evaluation is returned as content
- Alternatively set \(-\text{content field of response dict available in variable \text{response}}\)
- Use the response dict to access the request information (e.g. query information)
- Other files with same name but different extension are also matched to template file, which allows for one template file to provide multiple formats (e.g. test.html, test.txt and test.tml)
Example 3: Pre/Post/Not found filter

• Pre filter (.auth): return code != 200 or set response dict –code to value != 200 to deny access
• Post filter (.wrapper): transforms responses after they have been processed (e.g. set content type)
• Not found (.notfound): is ran when request can’t be resolved
• If no filter file found in requested directory, Mason goes up in file system hierarchy until it finds one or reaches its root directory.
Example 4: Direct domain

- Dispatch a URL request to:
  - Commands within a namespace
  or
  - Methods within an TclOO object

- Proc/method names must start with a /

- Adding direct domain to nub:

```
domain /directns/ Direct namespace MyDirectDomain
domain /directoo/ Direct object $::oodomain
```
Example 4: Direct arguments

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctype</td>
<td>Default content type of returned values</td>
<td>text/html</td>
</tr>
<tr>
<td>wildcard</td>
<td>A proc/method to be used when the request URL doesn’t match any of the proc’s/methods</td>
<td>/default</td>
</tr>
</tbody>
</table>
Example 4: basic Direct proc

namespace eval MyDirectDomain {  
    proc /test { req } {  
        dict set req -content "Test for MyDirectDomain"  
        dict set req content-type x-text/html-fragment  
        dict set req -title "MyDirectDomain: test with query"  
        return $req  
    }  
}
Example 4: basic Direct method

```plaintext
::class create MyOODomain {
    constructor {args} {}
    method /test {req args} {
        dict set req -content "Test for MyOODomain"
        dict set req content-type x-text/html-fragment
        dict set req -title "MyOODomain: test"
        return $req
    }
}

set oodomain [MyOODomain new]
```
Example 4: Query arguments

• Specify query arguments as proc/method arguments with same name as used in the query
• Wub will decode and assign the query arguments.
• Arguments missing in the request passed as empty string
• Use utilities in `Query` package to handle queries
Example 4: armouring

- HTML special characters (<, ‘, ...) need to be armoured.
- Use built-in command armour.
Example 4: conversions

• Wub converts all content to something permitted by the accept request field (e.g. test/html, text/plain, ...)
• The Convert domain has a table of possible conversions
• Conversion notation:
  .<from>.<to>
• Basic built-in conversion:
  .x-text/html-fragment.text/html
Example 4: Content type x-text/html-fragment

<table>
<thead>
<tr>
<th>Response dict key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-content</td>
<td>Content to be sent to client</td>
</tr>
<tr>
<td>-title</td>
<td>Title field in header</td>
</tr>
<tr>
<td>-headers</td>
<td>List of HTML statements puts verbatim in header</td>
</tr>
<tr>
<td>-script</td>
<td>List of script source and script arguments, converted into <code>&lt;script&gt;</code> statements and placed in the header</td>
</tr>
<tr>
<td>-style</td>
<td>List of style source and style arguments, converted into <code>&lt;stylesheet&gt;</code> statements and placed in the header</td>
</tr>
<tr>
<td>-preload</td>
<td>List of HTML <code>&lt;script&gt;</code> statement to be put in header</td>
</tr>
<tr>
<td>-postscript</td>
<td>List of script source and script arguments, converted into <code>&lt;script&gt;</code> statements and placed at end of body</td>
</tr>
<tr>
<td>-postload</td>
<td>List of HTML <code>&lt;script&gt;</code> statement to be put at end of body</td>
</tr>
</tbody>
</table>
Example 4: custom conversions

• Add proc in `conversions` namespace
• Name contains content type from which conversion starts and content type it generates: `.<from>.<to>`

```python
namespace eval ::conversions {
    proc .x-unarmoured-text/html-fragment.x-text/html-fragment { rsp } {
        set rspcontent [dict get $rsp -content]
        if {[string match "<!DOCTYPE*" $rspcontent]} {
            # the content is already fully HTML
            set content $rspcontent
        } else {
            set content [armour $rspcontent]
        }
        return [Http Ok $rsp $content x-text/html-fragment]
    }
}
```
Example 4: other content types

- Other content types can be returned by setting the `content-type` response dict value

```python
dict set req content-type text/plain
```
Example 4: generating html

- Wub offers HTML generation commands in **Html package**:

  `<command> ?key value ...? contents`

- Is converted into HTML statement:

  `<command key='value' key2='value2' ...> contents</command>`

<table>
<thead>
<tr>
<th><code>&lt;h1&gt;</code></th>
<th><code>&lt;h2&gt;</code></th>
<th><code>&lt;h3&gt;</code></th>
<th><code>&lt;h4&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;p&gt;</code></td>
<td><code>&lt;b&gt;</code></td>
<td><code>&lt;i&gt;</code></td>
<td><code>&lt;tt&gt;</code></td>
</tr>
<tr>
<td><code>&lt;ul&gt;</code></td>
<td><code>&lt;ol&gt;</code></td>
<td><code>&lt;li&gt;</code></td>
<td><code>&lt;span&gt;</code></td>
</tr>
<tr>
<td><code>&lt;table&gt;</code></td>
<td><code>&lt;tr&gt;</code></td>
<td><code>&lt;td&gt;</code></td>
<td><code>&lt;div&gt;</code></td>
</tr>
<tr>
<td><code>&lt;author&gt;</code></td>
<td><code>&lt;description&gt;</code></td>
<td><code>&lt;copyright&gt;</code></td>
<td><code>&lt;generator&gt;</code></td>
</tr>
<tr>
<td><code>&lt;keywords&gt;</code></td>
<td><code>&lt;meta&gt;</code></td>
<td><code>&lt;link&gt;</code></td>
<td><code>&lt;script&gt;</code></td>
</tr>
</tbody>
</table>
Example 5: Http commands

• Found in Http package
• Send Http responses to client:
  
  `Http <error-name> <response dict> ?arguments?`

• Supported error-names (with codes):
  
  – Ok (200)
  – Moved (301), Redirect (302),
    RedirectReferer (302), Found (302),
    SeeOther (303), Relocated (307)
  – Bad (400), Forbidden (403), NotFound (404)
  – ServerError
Example 5: Caching

- Use **Http commands** to enable/disable caching
  - `NoCache` : indicate response contents can not be cached
  - `Cache` : indicate response contents may be cached
  - `Dcache` : indicate response contents may be cached but revalidation is required
Example 5: Caching Age

• The Cache and Dcache take age argument:
  – Specified as integer: the age at which the contents expires, expressed in seconds
  – Specified as clock scan string: the point in time until which the contents remain valid (tomorrow, next week, 7 June 2009, ...)
Example 5: Clear cache

- **Cache clear**: Clear the complete cache. All cached contents is removed from the cache.
- **Cache delete <url>**: Remove the contents cached for the specified URL from the cache.
Example 6: Forms

- Create form with HTML generation commands from Form package
- Specify namespace proc or object method as form action
- Use post or get as form method
- As with queries, form entries are translated by Wub to proc/method arguments based on entry names

```
<form>
  <input>
  <button>
  <hidden>
  <textarea>
  <option>
  <optgroup>
  <select>
```

<table>
<thead>
<tr>
<th>&lt;form&gt;</th>
<th>&lt;input&gt;</th>
<th>&lt;button&gt;</th>
<th>&lt;hidden&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;textarea&gt;</td>
<td>&lt;option&gt;</td>
<td>&lt;optgroup&gt;</td>
<td>&lt;select&gt;</td>
</tr>
</tbody>
</table>
Example 7: jQuery

• jQuery is a JavaScript library
  http://jquery.com/

• Wub makes it easy to use jQuery and some of its plugins by wrapping it into a File like domain.

• To add jQuery to your application add this statements:

  set req [jQ jquery $req]
Example 7: jQuery

• To run a scripts when jQuery is loaded:
  set req [jQ ready $req <script>]

• To use jQuery plugin:
  set req [jQ <plugin> <selector> <plugin arguments>]

• List of wrapped plugins at
  http://wiki.tcl.tk/_wub/docs/Domains/JQ

<table>
<thead>
<tr>
<th>jframe</th>
<th>jtemplates</th>
<th>history</th>
<th>datepicker</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeentry</td>
<td>hint</td>
<td>boxtoggle</td>
<td>tablesorter</td>
</tr>
<tr>
<td>multifile</td>
<td>containers</td>
<td>tabs</td>
<td>accordion</td>
</tr>
<tr>
<td>resizable</td>
<td>draggable</td>
<td>droppable</td>
<td>sortable</td>
</tr>
<tr>
<td>selectable</td>
<td>autogrow</td>
<td>autoscale</td>
<td>tooltip</td>
</tr>
<tr>
<td>hoverimage</td>
<td>galleria</td>
<td>gallery</td>
<td>editable</td>
</tr>
<tr>
<td>form</td>
<td>validate</td>
<td>autofill</td>
<td>confirm</td>
</tr>
<tr>
<td>ingrid</td>
<td>map</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 8: Ini parameters

• Configure the webserver
• Uses ini-file syntax
• Per ini section, a dict is create in the Site namespace, keys and values of this dict are taken from the ini section, except wub section keys which become variable in Site namespace
• Pre-defined sections and keys
• User-defined sections and keys
### Example 8: Ini parameters

<table>
<thead>
<tr>
<th>Section</th>
<th>Key</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wub</td>
<td>home</td>
<td>[file dirnam [info script]]</td>
<td>Home of application</td>
</tr>
<tr>
<td>ini</td>
<td>site.ini</td>
<td></td>
<td>Ini file</td>
</tr>
<tr>
<td>globaldocroot</td>
<td>0</td>
<td></td>
<td>????:?</td>
</tr>
<tr>
<td>cmdport</td>
<td>8082</td>
<td></td>
<td>Console port</td>
</tr>
<tr>
<td>application</td>
<td>&quot;&quot;</td>
<td></td>
<td>Package to require</td>
</tr>
<tr>
<td>local</td>
<td>local.tcl</td>
<td></td>
<td>Post-init script</td>
</tr>
<tr>
<td>vars</td>
<td>vars.tcl</td>
<td></td>
<td>Pre-init script</td>
</tr>
<tr>
<td>listener</td>
<td>-port</td>
<td>8080</td>
<td>Wub listener port</td>
</tr>
<tr>
<td>cache</td>
<td>maxsize</td>
<td>204800</td>
<td>Maximum size of object in cache</td>
</tr>
<tr>
<td></td>
<td>high</td>
<td>100</td>
<td>High water mark</td>
</tr>
<tr>
<td></td>
<td>low</td>
<td>90</td>
<td>Low water mark</td>
</tr>
<tr>
<td></td>
<td>weight_age</td>
<td>0.02</td>
<td>Age weight for replacement</td>
</tr>
<tr>
<td></td>
<td>weight_hits</td>
<td>-2.0</td>
<td>Hits weight for replacement</td>
</tr>
</tbody>
</table>
## Example 8: Ini parameters

<table>
<thead>
<tr>
<th>Section</th>
<th>Key</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nub</td>
<td>nubs</td>
<td>nub.nub bogus.nub</td>
<td>List of nub files to process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>httpd</td>
<td>logfile</td>
<td>“wub.log”</td>
<td>Log file name</td>
</tr>
<tr>
<td></td>
<td>max_conn</td>
<td>20</td>
<td>Maximum number of connections per IP</td>
</tr>
<tr>
<td></td>
<td>no_really</td>
<td>30</td>
<td>How many time to complain about max_conn exceeded</td>
</tr>
<tr>
<td></td>
<td>retry_wait</td>
<td>20</td>
<td>How long to advise client to wait on exhaustion (in seconds)</td>
</tr>
<tr>
<td></td>
<td>timeout</td>
<td>60000</td>
<td>Idle time tolerated (in milli-seconds)</td>
</tr>
</tbody>
</table>
### Example 9: Nub

- Configure Wub (interactively)
- Map URL’s to domains

<table>
<thead>
<tr>
<th>Nub command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain &lt;url&gt; &lt;domain&gt; &lt;args&gt;</td>
<td>Process specified URL with given domain</td>
</tr>
<tr>
<td>redirect &lt;from&gt; &lt;to&gt;</td>
<td>Redirect &lt;from&gt; URL to &lt;to&gt; URL</td>
</tr>
<tr>
<td>rewrite &lt;regexp&gt; &lt;script&gt;</td>
<td>Transform URL selected by the regexp into the one calculated by the script</td>
</tr>
<tr>
<td>code &lt;url&gt; &lt;script&gt;</td>
<td>Send result of evaluating the script to client requesting specified URL</td>
</tr>
<tr>
<td>literal &lt;url&gt; &lt;contents&gt;</td>
<td>Send literal contents to client accessing specified URL</td>
</tr>
<tr>
<td>block &lt;list of url’s&gt;</td>
<td>Block IP addresses trying to access specified (glob matched) URL’s</td>
</tr>
</tbody>
</table>
Example 10: Suspend/Resume

• Suspend request until response is available:
  return [Httpd Suspend $req]

• Resume suspended request when response became available:
  Httpd Resume [Http Ok [/test_return_result $req]]

• Used as basis for e.g. Wubchain where request for new chat messages from client is suspended until a new chat message arrives from another chatter
Example 11: Cookies

- Cookies are stored in the request dict
- The Cookies package can be used to manipulate the cookies as stored in the request dict:

```
set cdict [dict get $req -cookies]
set vdict [Cookies fetch $cdict -name my_cookie]
set value [dict get $vdict -value]
```
Example 12: Command port

- Make connection to running Wub using telnet from localhost
- Stop, configure or query running server
- Set command port with cmdport ini parameter in wub section
More domains

- CGI
- Coco: co-routine domain
- Commenter: parse Tcl file and show comments
- Dub: database domain (based on metakit)
- Honeypot: catch spiders and link harvesters
- Login: simple login account handling
- RAM: convert contents of an array into a domain
- Repo: file repository (e.g. half bakery)
- Session: session handling
- Tie: mapping of namespace variables to URLs
- Tub: direct domain to store arbitrary data
- Woof: Wub’s interface to Woof (Web oriented object framework by Ashok P. Nadkarni)
More utilities

- Auth: authentication (rfc 2617)
- captcha: create captcha’s using convert
- Color: color manipulation
- Debug: debug info logger
- Report: convert a dict or csv data into an HTML table
- scgi: scgi interface
- Sitemap: create google sitemap from dict
- UA: user-agent detection
- Url: URL manipulation
Reporting bugs, feature request, ...

• Bugs and feature request for Wub are best reported at
  http://code.google.com/p/wub/issues

• Bugs and feature request for this tutorial is best reported at
  http://code.google.com/p/wubwikit/issues
Conclusion

- Wub is a new environment for Web application development using Tcl/Tk
- Reliable (see Tcler’s wiki)
- Under active development
- Offering lots of useful domains and utilities to make a developer’s life easier
- When looking for a Tcl solution and thinking about using tclHTTPd, consider using Wub