A ***T***cl ***W***eb ***F***ramework
(***A***rnulf's ***T***cl ***W***eb ***F***ramework)

History

- Start was about 2007
- Collecting ideas in Tcl wiki entry “ToW Tcl on WebFreeWay“
- Discussion about architecture
- „substify“ proc from Jean-Claude Wippler

Goal of the Tool

- Have something like Ruby on Rails for Tcl
- Use of MVC architecture
- Serverside parts for building html/xml output
- Mix in data from DB
- Controllers, actions and views parts can be built from different persons
- Allow enduser to use default parts, but allow to overwrite all defaults with own code

Step 1

- Use of apache server with rivet on server side
- Use wub
- Use of cgi-bin
- Use of meta layer for DB access
- Implementation of some actions functionality including DB access functionality

Step 2

- Inspired from a payed work project new strategy to use ideas from PHP Zend framework
- Implementation start of some small controller and action parts
- Use of „substify“ proc for view part
- Use of tdbc for DB access

Step 3

- As most of the implemented stuff was very similar to Zend strategy, change to „convert“ Zend to Tcl
- Nearly 1 to 1 conversion of PHP code to Tcl
- Use of itcl-ng for corresponding PHP classes and objects

Step 4

- Use of Tcl dicts instead of PHP arrays
- Use of tdbc instead of PDO adaptors
- Use of Tcl namespaces instead of PHP simulated namespaces
- Use of „substify“ proc for generating final html or xml output
- Use of the corresponding directory structure model from Zend for an application
- Use of the module/controller/action model from Zend

General Directory Structure

- application
  - configs
  - controllers
  - models
  - layouts
  - views
  - modules
    - module1
- public

Structure of Url's

- http://...../<module name>/<controller name>/<action name>?<parameters>
- **<module name>**
- the name of the module (optional, default empty)
- **<controller name>**
- the name of the controller (optional, default index)
- **<action name>**

  the name of the action (optional, default index)
- **<parameters>**

  0..n key value pairs separated by “&” and “=” or by “/”

MVC Parts (1)

- Example url 1: http://localhost/people/list/id/123
- Example url 2: http://localhost/people/list?id=123&company=xy
- No module name in these examples
- Controller name is people
- Action name is list

MVC Parts (2)

- models directory represents data to be shown
- DbTable subdirectory contains model classes for DB tables
- layouts directory contains layout information like css info etc.
- views contains templates for building html/xml output, with scripts subdirectory
- scripts subdirectory normally contains a subdirectory for each controller with the name of that controller
- here are templates for building the html/xml output

MVC Parts (3)

- A dispatcher dispatches a request calls controller and action and returns a response
- A controller class has a method for every action
- An action gets and prepares the needed data using the models to get i.e. DB data
- DB data are fetched using DbTable adapters
- An action stores the prepared data in the view object and calls the view renderer there
- The view renderer builds the html/xml output using „substify“ proc and template in scripts subdirectory

Resources

Standard Resources (Plugins)

- Db
- FrontController
- Layout
- Locale
- Modules
- Navigation
- Resource
- Router
- Session
- Translate
- View

Dispatcher Steps(1)

setRequest

setBaseUrl

setResponse

for every plugin call method setRequest

for every plugin call method setResponse

initialize router

router setParams

initialize dispatcher

dispatcher setParams

dispatcher setResponse

Dispatcher Steps(2)

```
while not dispatched
    for every plugin call method routeStartup
    $router route
    for every plugin call method routeShutdown
    for every plugin call method dispatchLoopStartup
    while to be dispatched
        setDispatched true
        for every plugin call method preDispatch
        if request is not completely dispatched continue
        $dispatcher dispatch $request
        notify plugins of dispatch completion call postDispatch
        if request is dispatched break
    end of to be dispatched
    for every plugin call method dispatchLoopShutdown
    if $returnResponse return the response
    else: $response sendResponse
end of not dispatched
```

Config Example

```
[app]
webhost          = wiedeman-pri.de
title            = Arnulf's Tcl Web Framework
includePaths.library  = @APPLICATION_PATH@/../library/ATWF/
includePaths.models   = @APPLICATION_PATH@/models/
@APPLICATION_PATH@/models/DbTable/
bootstrap.path       = @APPLICATION_PATH@/Bootstrap.tcl
bootstrap.class      = ::Bootstrap
resources.frontController.controllerDirectory = @APPLICATION_PATH@/controllers
resources.frontController.moduleDirectory = @APPLICATION_PATH@/modules2
resources.frontController.moduleDirectory = @APPLICATION_PATH@/modules
resources.layout.layoutpath  = @APPLICATION_PATH@/layouts
resources.db.params.host     = localhost
resources.db.params.username = arnulf
resources.db.params.port     = 3306
resources.db.params.dbname   = atwf_test
resources.db.adapter         = MYSQL
resources.modules
```

Bootstrap Example

```
::itcl::extendedclass Bootstrap {
    inherit ::ATWF::Application::Bootstrap::Bootstrap
    constructor {application} { chain $application} {}
    public method bootstrap {{resource {}}}
    public method run {}
}
::itcl::body Bootstrap::constructor {application} {
}
::itcl::body Bootstrap::bootstrap {{resource {}}} {
    next $resource
    return $this
}
::itcl::body Bootstrap::run {} {
    next
}
```

Controller Example

```
::itcl::class Module1::ModuleController {
    inherit ::ATWF::Controller::Action
    constructor {request response {invokeArgs {}}} {chain $request $response
$invokeArgs} {}
    public method init {}
    public proc moduleAction {}
}
::itcl::body Module1::ModuleController::constructor {request response {invokeArgs {}}} {}
::itcl::body Module1::ModuleController::init {} {
    set contextSwitch [$_helper getHelper ContextSwitch]
    $contextSwitch addActionContext module xml
    $contextSwitch initContext
}
::itcl::body Module1::ModuleController::moduleAction {} {
    set people_obj [Module1::Model::DbTable::People #auto]
    set rows [$people_obj getRows [list "id > ?" 27 "ID < ? " 36]]
    $view VarSet var1 "hallo Arnulf!";        $view VarSet rows $rows
}
```

Model Example

```
::itcl::extendedclass ::Module1::Model::DbTable::People {
    inherit ::ATWF::Db::Table
    constructor {} {  set _name people ;  chain} {}
    public method getRows {where} {
        return [fetchAll $where]
    }
}
```

View Example

```
<?xml version="1.0" encoding="UTF-8" ?>\n
% set rows [$this VarGet rows]
% if {$rows eq ""} { set cnt 0 } else { set cnt [$rows count]}
% set row_num 0
  <rows>
% while {$row_num < $cnt} {
%   set row [$rows getRow $row_num]
    <row>
      <rownum>$row_num</rownum>\n
%   foreach name [list idnum name created modified info] {
      <field>
        <name>$name</name>
        <value>[$row __get $name]</value>
      </field>\n
%   } ;   incr row_num
    </row>\n
% }
  </rows>
```

Status

- Conversion of about 50.000 lines of Zend framework
- Time frame February to May 2010
- Use of itcl::extendedclass from itcl-ng
- A big part of Zend basic functionality (V1.9.3)
- Dispatching of “normal” requests including redirecting etc. is implemented
- DB Adapter for mysql is running
- Other DB adapters can be easily added using tdbc functionality

Todos

- Test suite
- Complete integration into the 3 server types (Apache, cgi-bin, wub)
- Look for parts, which can be made more Tcl'ish
- Decide what other parts of Zend are needed/interesting
- Prepare alpha version
- Take care of feedback to that version
- Documentation
- Demos