

WebFu
The Art of Dynamic Webpages

Matthew Estes

March 25, 2006

Contents

1	Introduction	1
1.1	Prerequisites	1
1.2	What is the WebFu Framework?	1
1.3	Component Diagram	3
1.4	File Layout	4
1.5	Licensing Issues	5
1.6	Credits	5
2	WebFu IDE	7
3	WebFunction Compiler	9
3.1	What is a Web Function	9
3.2	Role Based Security	9
3.3	Command Class	10
3.4	The WebFunction Language	10
3.5	PHP Support Library	12
4	Workflow Compiler	13
5	Form Creator	15
6	Bishop Compiler	17
A	Web Development Basics	19
B	HTTP	21
B.1	Church and State	21
B.2	HTTP Commands	21
C	HTML, XHTML, XML, and HTML Forms	23

D	CSS	25
E	Javascript	27
F	PHP	29
	F.1 Object Model	29
G	Class Library	31
	G.1 Account	31
	G.2 AuthUser	31
	G.3 CommandFactory	31
	G.4 Ctl*	31
	G.5 DB*	32
	G.6 ParserWiki*	32
	G.7 PostVars	32
	G.8 TypeDB*	32
	G.9 TypeForm*	33
	G.10 Utilities	33

Chapter 1

Introduction

1.1 Prerequisites

The WebFu framework is a system for developing database backed, dynamic websites in PHP 5, with a focus of utilizing Javascript to enhance usability on the web based clients. We assume that you know how to program in PHP, and are interested in using PHP 5 if you don't already. We assume that you understand the basic of HTML, HTTP, and Javascript. It would be nice if you understand Cascading Stylesheets.

While there is no need to be an "expert" to use any of this framework, and the framework is designed to be learnable in small chunks. In fact, some parts may only truly be valuable in complex systems, while other parts are useful in websites of any size.

The tools in the framework are based on Java 1.5, and a 1.5 JVM is required to run the tools. The PHP class library is licensed so that you can build royalty free commercial websites if you have purchased the framework(see the license included in the source for more details). The tools and the IDE are not open source, but the generated source code is yours to use.

1.2 What is the WebFu Framework?

WebFu consists of 7 distinct pieces, which can be used independently(to lesser or greater affect).

- **Site configuration Management System** Code to set up a standard layout of files, manage database backed sessions, provide a consistent way to configure code that is deployed to multiple web servers/environments. This code also overloads error handling to integrate error reporting into a bug management system for production web sites. Most of the code for this subsystem is in the /lib directory.
- **Class Library** A set of PHP classes to perform common tasks needed to develop web apps. This includes various authorization methods, login, user account management code, create

graphs, connect to databases, among other things. Most of the files in the `/stdclasses/` folder are a part of this library.

- **Web Functions** This is a little language which helps secure sites by processing and validating form data, and providing a security layer to create a role-based permissions system. The type system for form validation is user extensible. See the `TypeForm*` classes.
- **Form Creator** A HTML-to-PHP compiler that creates PHP functions out of reusable forms. This also includes a library of AJAX controls and form elements to simplify creation of Javascript heavy forms.
- **Workflow Language** This is a little language for specifying workflows. It consists of compiler which transparently integrates in to the Web Functions language, and a JVM based server for managing workflow state. This server uses a simple sockets based protocol, and can be used to integrate a web-app with non-web workflows with the accompanying Java package for connecting to the server. This tool also generates flow charts of the workflows suitable for end users.
- **Bishop Database Language** A procedural database language that creates code for SQLite and PostgreSQL and other database backends. It uses built-in constraint enforcement when available and emulates it in the generated PHP when its not available. This is NOT an Object-Relational Mapper(ORM). This is a SQL replacement that does not use a least-common-denominator approach for supporting multiple databases. This makes a reasonable replacement for implementing a Model in a MVC system. Other backends besides PHP are planned. This tool also generates E-R diagrams for your database schema, and other tools for managing your database.
- **WebFu IDE** The final piece is the WebFu Integrated Development Environment which provides syntax-highlighting editors, integrated support for version control packages, and version control backed multiple deployment configurations to deploy different versions of your site to different system(e.g. Testing, Development, Production). The IDE also provides context sensitive help for the different packages in addition to support for writing PHP.

Together, you can consider this collection to be the WebFu framework. You might consider this a cross between a framework for rapid development of CRUD(Create, Report, Update, Delete)/MVC(Model-View-Controller)/N-tier(you don't want to know...) websites. Now, that the acronym soup is out of the way, the overarching design of the pieces is based on letting you drive the site. Very few pieces are required, and none of the generated code should ever need to be edited by hand(if you do, let us know, we consider this a major issue and is the major contributing reason why many people distrust code generation techniques, we'll help you figure out another solution or fix the bug).

1.4 File Layout

In a typical WebFu site, the following standard layout is suggested to help structure your code. These are the defaults for any new project.

```
/classes
/genclasses
/lib
/resources
/stdresources
/stdclasses
command.php
index.php
site.php
xmlhttp.php
```

The `/classes` directory is where your project specific classes are stored. While PHP does not enforce a certain layout of files, we highly recommend that you write ONE class per `.php` file in this folder, and name the PHP file the same name as the sole class in that file. This is similar to the way Java requires you to write your code. While this may seem unnecessary, PHP 5 lets you write a "magic function" called `__autoload()`, which can dynamically load classes that cannot be found in the search path. Our framework provides an `__autoload()` function which looks in a user specified(usually `/classes`) folder for classes which cannot be found.

The `/genclasses` directory is where tool-generated classes are stored. There is no reason to edit the code that is stored in this folder, since it will be overwritten the next time you rerun one of the compilers.

The `/lib` directory is where the "core" code for the framework can be found. The file `/lib/prefix.php` should be required at the BEGINNING of any file that you wish to use the framework in, and `/lib/postfix.php` should be required at the END of the file. `prefix.php` loads the file `site.php` to look for configuration defines that define your website. It also overloads the built-in error handler if your site is in production mode (`SITE_DEBUG = false`) with an error handler that emails you about PHP errors. If custom session handling is on, it loads up the database backend for storing sessions. It also starts output buffering so that you can more flexibly generate HTML, headers, and cookies.

The `/resources` directory is the suggested place for storing your `.css` and image files used by your site. Depending on how Javascript heavy your site is, you might want to store your custom Javascript here, or another directory(like `/js` or `/javascript`). Part of the WebFu framework is the strong suggestion that following good conventions will lead to better code. While we don't impose this on you, its a good practice, and will help other developers(particularly in the WebFu framework) understand your project at a glance.

The `/stdresources` directory is where the Javascript code, image files, and css used by Form Creator controls is placed. More can be found about this in the chapter on the Form Creator.

The `/stdclasses` directory is where the framework provided class library is placed. These files are doxygen documented, and contain more documentation in the section of the book dedicated to the class library. Depending on the number of WebFu sites sharing a web host, it is possible to move this folder to a single location that all sites can use the same copy.

The `command.php` file is the default target for all forms and all HTTP **POST** requests to target. This returns a redirection to a url to retrieve via HTTP **GET** and performs the state change required by the request. This file should not be changed, but may be a good example to consider if you need similar behaviour.

The `index.php` file provided is an example. Consider this a "template" that calls `Commands`(in the `WebFunction` compiler sense) to get pieces of output. The provide example should be a template for any request to be handled by HTTP **GET**. Other uses would be code that outputs MIME-type headers to return generated images, or user uploaded files. This where the "look" and "style" is applied.

The `site.php` file is where site specific configuration constants are stored. This file will be automatically managed by the WebFu IDE so that the proper file is generated based on the Server Profile for the site you are uploading too.

`xmlhttp.php` is similar to `command.php`, except it makes no assumptions about **GET** or **POST**(it uses the PHP synthesized `_REQUEST` hash), and is intended to be used to process automated requests(like from AJAX components), and many controls used by the form generator target this file. It should not be changed, but does provide a template for files which have similar needs.

1.5 Licensing Issues

At present, the WebFu framework is not open source. You must acquire a license suitable to your needs from Metanotion Software. However, we do believe in the value of open source, and our work does build on several open source technologies. The PHP class libraries written by Metanotion Software can be used in any commercial work. In addition, the open source PHP and Javascript code that we have included has been vetted and should be safe for use in a commercial work as well. If we are distributing your open source project in contravention of your license, contact us and we will address your issues.

1.6 Credits

Some of the open source products, and commercial products we have used, and which we list here for your convenience and to give credit to the fine work of our fellow programmers:

- **Dev-CPP** This is a fine open source Windows IDE for gcc.
- **JavaCC** An excellent open source parser generator for Java.
- **JCreatorLE** This is an excellent free Java IDE.
- **JTB** Java Tree Builder, an add on to JavaCC which is indispensable
- **Mozilla/Firefox**
- **PEAR::JSON** The PEAR JSON module is included here.
- **PHP** Without which, this product would have to target something icky like Active Server Pages(ASP).

Chapter 2

WebFu IDE

The WebFu IDE is a Java Swing application. As such, it should run on any platform a 1.5 JVM is available for. While the IDE is not required to use the framework, it can greatly simplify invoking the compilers. The compilers

Chapter 3

WebFunction Compiler

3.1 What is a Web Function

There are several patterns that nearly every nontrivial web application must perform.

- Process HTML form data to the appropriate type
- Map URL's and query strings to code
- Provide a security model to determine who can do what.

There are others, but these three have common threads, and are the problems that the WebFunction language provides a partial solution.

3.2 Role Based Security

There are many "schemes" for implementing the security strategy of a system. Perhaps the most promising avenue is that of the language E which is based on the principle of least authority (POLA) and is studied as a part of Object-Capability systems. No matter what your architecture, you have to have a way to connect it to a user interface (in this case, a web site), and you have to be able to easily audit and understand the whole of the system. Role based security is a reasonably easy to understand system, and the security handling of the WebFunctions compiler is based on this strategy. It is not a complete solution, but instead a tool which allows you to easily build a complete solution that is both easy to audit and easy to use.

In the WebFunction language a role is just a label. The meaning of the label is up to the application. In addition, WebFunctions is not concerned with **how** you are authorized to assume a certain role, only with **what** actions you can perform. Some examples of common roles include *user*, *admin*, *customer*, *cashier*, *game_master*. Another common role, which we typically label

NULL represents a person who is not authorized in any role(the non-role), you can use another name if you want, but we happen to think *NULL* is a good name for the non-role role.

In addition, roles can perform commands. Commands might include *login*, *playGame*, *addToShoppingCart*, *saveDocument*, *createAccount*. Most object oriented languages use verbs as a prefix for method names and Camel Case for the rest of the name like *actionMustBeLikeThis*. Again, you may choose another convention, but this one has worked well for us.

Also, commands typically require parameters, like *Username* which is a string, *birthday* which might be a date, or even *employeeIDs* which might be an array of integers. You might have noted that most languages "type" the data sent through a web form: as a string. This is certainly "well typed", but not typed well.

Finally, a command executed by a role will require actual code to implement the command. In object oriented parlance this is a "delegate", and actually does the work. This would be a method of an object like *Account::createNew* or *Game::makeChoice*.

These represent the pieces of a what is contained in a WebFunction specification.

3.3 Command Class

The WebFunction compiler creates a set of classes, one for each role. Each class is prefixed with "Cmd" followed by the name of the role. The *CommandFactory* class checks the value of the `$_SESSION['cmdcls']` variable to determine which command class to instantiate when you call the *CommandFactory::getCommand* method.

In PHP each "command class" implements a method for each command a role can perform. For the other commands it generates a place holder function which returns an error code. This way, your front end code doesn't have to check for which commands it is allowed to execute, just delegate to the command class you're given. Likewise, on a rough scale, the back end code doesn't have to check who is allowed to do what either. Now, you still have to check to make sure a user is only trying to delete HIS document instead of someone else's, but that is considerably less effort than checking everything all the time.

Furthermore, the command class, when it delegates to your code, passes paremeters that have been processed for the proper type in the function call. You will never get a date string when you expected an integer(you might get a 0 if someone tries). Your data will not be "valid", but it will be of the proper type. You can even add new types by extending the TypeForm class.

3.4 The WebFunction Language

Now that you have some idea how it all works, here's the structure of a WebFunction file:

```
[ NULL, User , Admin ]
```

```

/*      Comment? */
GET loginForm() [ NULL ] Account::loginForm;
POST login(Username STRING,
           Password STRING) [ NULL ] Account::login;
GET mainPage() [ NULL ] MyApp::marketing
               [ User ] MyApp::userMenu
               [ Admin ] MyApp::controlPanel;
GET somethingCool(
    start DATE,
    someNumber ARRAY[INTEGER])
               [ User / Admin ] Cool::widget;

```

Whitespace is ignored(new lines, tabs, everything), and C-style comments are allowed anywhere whitespace is too. The first section is a role list, enclosed in square braces('[', ']') and separated by commas. Role names must start with a letter or underscore, and may consist of letters, underscore, and numbers.

After the role list, there is a command list. A command consists of 4 parts followed by a semicolon. The command type, the name of the command, the parameter list, and the role-delegate list.

Command types are permitted to be **GET**, **POST**, **XMLHTTP**. A **GET** command assumes the delegate should be executed only by an HTTP **GET** request, and typically returns HTML. A **POST** command assumes the delegate should be executed only by an HTTP **POST** request. A **POST** command should return a URL, which will be used to redirect the browser to a **GET**. Finally, **XMLHTTP** types do not assume either **GET** or **POST** and does form processing from the PHP `$_REQUEST` array. **XMLHTTP** can return anything, including HTML, although if you make use of the XMLHTTP request object implemented in most web browsers, you might find JSON to be a common output as well. In the *Cmd* classes, methods are prefixed with a *p_* for a **POST**, *g_* for a **GET** and a *xh_* for **XMLHTTP**. Make sure you remember this if you call *Cmd* methods directly.

All the support code in the framework assumes that the command name will be part of a URL, and is typically stored by the *cmd* parameter of a querystring, or a hidden form input named *cmd*.

The parameter list consists of a comma delimited list of parameter names, which correspond to parameter names in the form or querystring, and a type for each parameter. A type can optionally have a shape. The shape is enclosed in square braces([,]), and is a comma separated list of "type parameters". Each parameter has an optional label(e.g. *name* : this comes first), followed by either a number, a string(enclosed in double quotes), or another type(which can also have a shape).

The Role Delegate list takes the form of a role list, enclosed in square braces, followed by an object/method pair. To use the same delegate for multiple roles, the role list should be delimited

by a forward slash. There is no separator other whitespace in the role delegate list.

Finally, each command spec must be terminated with a semicolon(';').

3.5 PHP Support Library

The generated *Cmd* classes make use of the *PostVars* class in the standard library. This class understands the "type" processing and knows how to instantiate the classes implementing types. All form types must extend the *TypeForm* class. This has one method, which processes the input given to it and returns the appropriate PHP value. The standard types that come with the system include:

- **Array** - Array is a generic type, and its "shape" is another type(such as Integer), it assumes that there are a series of form variables that match the form *name_n*, and return a PHP array, whose elements of the type of its "shape". If you wish to implement a more complex type, the source to this class makes a good example of what can be done.
- **Boolean** - Processes checkboxes and returns *true* or *false*.
- **Date** - Gives a date string, using PHP's *strftime* and *strtotime* functions. The "shape" of this type is a format string for *strftime*. If no date is passed in, this gives a blank. Note, this function is *MAGIC_QUOTES* aware, and strips unnecessary slashes before processing.
- **Integer** - Simply calls PHP's *intval* function
- **JSON** - This uses the PEAR JSON library to process a form family into a PHP object. Obviously, this is a pretty flexible type, and useful for XMLHTTP enabled apps, however, you might consider that a more "unfolded" combination of types and parameters will give you some useful guarantees.
- **String** - This simply passes through the string value untouched, unless *MAGIC_QUOTES_GPC* is on. If *MAGIC_QUOTES_GPC* is on, this undoes the effect of *MAGIC_QUOTES*.

Chapter 4

Workflow Compiler

Simple web applications often do not need workflows; however, more complex applications often develop several needs that the Workflow language solves, some of which include:

- Initiate multiple, long running, parallel "background tasks"
- Wait on several tasks to complete before starting another task
- Synchronize and sequence a complex series of parallel user and background tasks.

The workflow language is based on the simple idea of multiple processes which communicate via messages. These processes are executed inside a lightweight Java based server. The messages are defined in terms of Web Functions, and a Web Function script must exist before a Workflow script can be created.

Chapter 5

Form Creator

Chapter 6

Bishop Compiler

Appendix A

Web Development Basics

If you are new to web development, need a refresher on HTTP, HTML, Javascript, CSS, or PHP 5 then the chapters in this part of the manual are for you. This is not a complete tutorial on any of these technologies or an introduction in how to program. We do not advertise WebFu as a silver bullet for nonprogrammers to write webapplications(although we certainly think it makes it easier for everyone, beginners included). However, we still want to help you along, and there are corners of these technologies that we explain here because of their relevance to the design patterns in the WebFu framework. Furthermore, while web technologies are very well designed and work well, they are not always intuitive, and the WebFu code, as well as your code, must work with the grain of web technologies.

Appendix B

HTTP

HTTP is the foundation of the world wide web. HTTP is the network protocol most often used by Web browsers to talk to Web servers. It stands for **Hyper Text Transfer Protocol**. Its basic operation is simple. A URL(*Universal Resource Identifier*) is entered into a web browser(either by typing in the address bar, clicking a link, or requested by a program written in javascript, etc.)

The resource identifier(everything after the third '/') is sent to the web server, which then identifies the resource as either a file on disk, or a program to be executed. If its a file, it sends the contents of the file, and if its a program, it executes the program and returns the output.

B.1 Church and State

One thing to mention about the HTTP protocol is by its very nature it "state-less". Every request is independent of the previous request, and the server doesn't(normally) stay connected to the client. Even more importantly, all contact between a web server and web browser is **initiated** by the web browser.

This is not a problem. But it can have a profound effect on how you structure your application.

Let that sink in a minute.

B.2 HTTP Commands

The HTTP protocol defines several commands. The two most basic are **GET** and **POST**. There is a very important difference. **GET** returns a resource, and is supposed to be "idempotent". This is a nice mathematical word that means it always does returns the same thing. Executing two **GET**'s to the same resource back-to-back should result in the same output from the web server. Almost. The exception is that if something(else) changes things while we're gone, that

might not be the case, but nothing should be *caused* by issuing that first **GET**, to change the output of the second identical **GET**.

You see, **GET**'s are supposed to be cache'able and "safe". A while back Google, following the HTTP standard, released a "Web Accelerator". While you were browsing the web it would follow the links(which are retrieved by **GET** commands) and save the results so that when you followed the link, the page might already be downloaded and so you could view it much faster.

Unfortunately, several many web developers had created applications in which "delete" commands were accessed via hyperlinks that web browsers would **GET** instead of **POST**. Google's Web Accelerator then proceeded to **GET** all the delete links in the website, leaving a puzzled and angry user(who, I assure you, created a puzzled and angry web developer while leaving a support request).

If following the link changes something, use **POST**. Do not use **GET**, after all, one day Google may visit you with a Web Accelerator and your users will hate you.

Appendix C

HTML, XHTML, XML, and HTML Forms

Appendix D

CSS

Appendix E

Javascript

This should be titled ECMAScript. This is documented everywhere, but its useful to repeat facts so that people will remember them. Originally, Netscape came up with a browser scripting language called LiveScript. Netscape observed that Sun was doing a tremendous marketing effort to associate Java as the programming language of the internet, and strategically decided to change the name of LiveScript to JavaScript to benefit from the free advertising. Both Java and JavaScript look similar, falling into the set of C-inspired "curly brace" languages. Java is a Class based, object oriented programming(OOP) language with a huge class library which is compiled to byte code to run on a Java Virtual Machine(JVM). JavaScript is a Prototype based, object oriented programming language which barely has a standard library, which depends on its hosting environment to provide I/O, and is not compiled to any sort of standard bytecode, much less Java bytecodes. You could compile it to Java bytecodes. That still wouldn't make it Java though. Prototype based OOP can be very different class based OOP. At some point, someone standardized JavaScript with the European standards body, ECMA, and they decided to try and reduce confusion by increasing it and renaming the language ECMAScript.

JavaScript is executed on the web browser, and is limited tremendously by the security of the web browser(well, actually, there probably is a web browser out there somewhere which doesn't limit it, but this is not a Good Thing(TM)).

E.1 XMLHTTP Request Object

Appendix F

PHP

We use the 5th version of the PHP programming language. PHP has evolved heavily, and version 5 contains a totally reworked object model. Many parts of the WebFu framework depend on the PHP 5 object model, and it is more likely that we would port it to other web development languages than earlier versions of PHP. Earlier versions are not bad, but many features of PHP 5 make it a vast improvement for our purposes.

F.1 Basic Syntax

F.2 Object Model

PHP 5 has a different, slightly backwards-compatible, object model than PHP 4. This is probably the major source of incompatibility between PHP 4 and 5. The object model bears some resemblance to Java's.

F.2.1 Structural Subtyping/Duck Typing

Appendix G

Class Library

G.1 Account

Functions to implement commonly needed user account management.

Suggested Command Code:

POST goAdmin()	[User]	Account::goAdmin;
POST goUser()	[Admin]	Account::goUser;
GET myAccount()	[User / Admin]	Account::edit;
POST updateAccount()	[User / Admin]	Account::update;
GET createAccount()	[NULL]	Account::newAccountForm;
GET forgotPassword()	[NULL / User / Admin]	Account::forgotPasswordForm;

G.2 AuthUser

A library to provide a form based, database-backed login authentication.

G.3 CommandFactory

This is part of the Web Functions PHP support library. It uses the current session to determine which generated class to use for providing app security. If you need access to a command class, instantiate a *CommandFactory* object and call the *getCommand()* method.

G.4 Ctl*

These are classes used by the Form Creator to create elements of various controls. While they are named similarly to distinguish them, there is no inheritance hierarchy among them, and

each one is ad-hoc. There is nothing specific to the Form Creator about them, and they may be freely reused outside of the Form Creator.

G.5 DB*

The primary class is *DB*, which is a singleton for establishing connections to the database. The *DB** classes provide an abstraction over basic database functionality. They can be used directly, but the *DB* class provides equivalent methods which delegate to the appropriate class based on the *DB_TYPE* define. While this class does not implement any form of connection pooling, it provides an abstraction for getting a *SINGLE* database connection during the run of script without the need for passing around the database connection. The *AuthUser* and *Account* classes depend on these classes.

G.6 ParserWiki*

The Metanotion Wiki language is our dialect of Wiki. We have tried to be faithful to the idea of Wiki syntax, but since there is no unified Wiki language standard(at this time), we have implemented our own compromise for ease of use and ease of coding. The Wiki dialect used is documented in the chapter on the Wiki classes. In addition, the Rich Text edit control, which uses *HTMLarea* which is available in IE 5.5 or higher(shipped with Windows 95B), Mozilla 1.3 or higher, and Firefox can be switched into Wiki text mode and uses this Wiki dialect. This provides a convenient way to have user formatted text while avoiding cross site scripting attacks.

G.7 PostVars

This is part of the Web Functions PHP support library. It is used internally by the generated *Cmd** classes. You can call it directly if you wish, but it is undocumented and meant for internal use only.

G.8 TypeDB*

These classes sanitize inputs for SQL, and provide a convenient manner of avoiding SQL injection attacks. Some of the more "complex" types(e.g. JSON, Wiki) are formatted as a string. These are used by the *DB** classes and the Bishop generated code. You can call them directly.

To add your own database "type" extend the abstract *TypeDB* class and implement the abstract methods defined in it(*toDB* and *fromDB*).

G.9 TypeForm*

These class process inputs from HTML forms, and are used by the Web Function generated code. While these don't "validate" all form elements, they do provide guarantees on their output to avoid more tedious checks (patterns like *if(isset(\$_POST{'var'}))\$v = intval(\$_POST{'var'});*).

To implement your own "type" extend *TypeForm* and implement the abstract methods defined in it. For an example of what kind of complex form processing is possible you may want to look at the *TypeFormArray* class.

G.10 Utilities