DATALVS 2.0 BETA 1
Developer Manual

Abbey Hawk Sparrow

April 23, 2010

1 The Basics

When approaching other frameworks my criticism is often that it forces me to work within the confines of it's preferred method. Sometimes, I want a full framework, sometimes I want less, and I like it when the things I don't need get out of my way. I'm not sure if other's don't work this way, or if forcing the user to use all or nothing is part of some framework developing methodology I wasn't let in on. But I just don't believe in it.

So my central goal with DATALUS.PHP is to provide a robust suite of tools which function harmoniously, but independently.

1.1 Architecture

1.2 History

DATALUS began life as a templating engine I wrote in 1999 as my first foray into PHP called 'Catalyst', which used recursive HTML templates with an escape charater sequence and properties files to create a session with permissions and to generate a page, so I could spend my time tooling around with SQL, Forms and the wheel-reinvention normally associated with web development. The idea was I would be able to maintain a common code-base between all the sites I managed. The truth of it, was I maintained slightly different versions for each site, which helped, but didn't exactly solve the problem.

In late 2003 I decided to scrap the old idea in favor of a new design which would hide everything but getting, setting and the initiation of loading and storing. Any other interaction would have to either be automated or prompted to the user. During this time I basically kept a running sheaf of papers refining what I thought would work and doing various proof-of-concepts.

At some point in 2006 I grew frustrated with my own templating language and discovered XSL, this in combination with ideas that come from a Java library for making generative desktop interfaces I was also writing gave me this model. The java variant had grown out of a project to make a binary data editor configured by XML files, but was now built to generate and store data in a generic way to a variety of datasources, and build displays for editing them. This lasted for 2 years in both a java and php form with slightly different dialects, until I began a ground up rewrite one more time in 2008.

The goal of this last rewrite was to use datalus objects throughout the core of the project and to further segment the data handling code from the GUI visualization code, to make that data core more portable. In addition I had successfully 'inverted' Smarty in a previous project which led to an extremely intuitive recursive panel structure which allowed me to have views which could display their own data, catch their own forms and coupling this with the existing data layer made and extremely clean MVC abstraction.

2 Model

2.1 XML

2.1.1 benefits in a world of heterogeneous data

2.1.2 the definition is the container

2.1.3 nodes

2.1.3.1 object

2.1.3.2 instant

2.1.3.3 integer

2.1.3.4 link

2.1.3.5 linkgroup

2.1.3.6 permission

2.1.3.7 string

- panel this is where a subpanel is rendered as well as a test is defined. In order to render a panel:

```
{panel name="path/relative/to/panel/root"}
```

It also accepts a 'params' argument which is an associative array of the parameters being passed in. To define a test on any given panel, just add the test panels with the format <identifier>_test such as:

```
{panel name="path/relative/to/panel/root" my_test="anotherpath/relative/to/panel/roo
```

then in order to measure conversions you simply use the convert macro on the page you wish to convert on:

```
{convert group="path/relative/to/panel/root"}
```

(make sure to convert using the name of the *main* group, not one of the test groups)

Yeah, that's it... wait for the hits and conversions to roll in.

- page this is where we set the main data for the page macro: title: this is the actual title tag content heading: page heading, used as a subsection label, but can be used for most anything. meta: text for the meta tag wrapper: this is the wrapper that we will render
- breadcrumb this is a macro to take care of breadcrumb trails in a simplified way (any of levels) an example is best:

```
        {breadcrumb name="trail"
            crumb1="Home" link1="index"
            crumb2="Company" link2="company"
            crumb3="About Us" link3="company"
        }
```

- oo This is for handling variables as objects from within smarty... it supports pulling a value from a function:

```
{oo object=$object function="func"}
```

or as a field off the object itself:

```
{oo object=$object field="fieldname"}
```

- convert this macro generates the conversions for an AB test, usage is covered in the panel section above
- txt access text bundle

```
{txt key="some.thing"}
```

accesses the 'thing' entry in the Strings/some.properties file inside the text bundle

```
{txt key="thing"}
```

accesses the file body of the Blocks/thing.txt file inside the text bundle

- ll the local link tag serves as a passthru point for link rewrites to occur, this can allow us to support cookieless sessions and conditionally rewrite areas of the site to various domain mappings.
- enumeration this renders dropdowns from the .properties files that exist in the Enumerations directory

```
{enumeration name="enum_name" identifier="form_name" value="default"}
```

- primitive this allows the internal primitives (such as an EntityAttribute's value) to be displayed:

```
{primitive type="mailing_address" value=$value mode="display"}
```

or displayed in an editor:

```
{primitive type="mailing_address" value=$value mode="edit"}
```

- wrapper_variable this allows you to pass a value to the wrapper from a panel

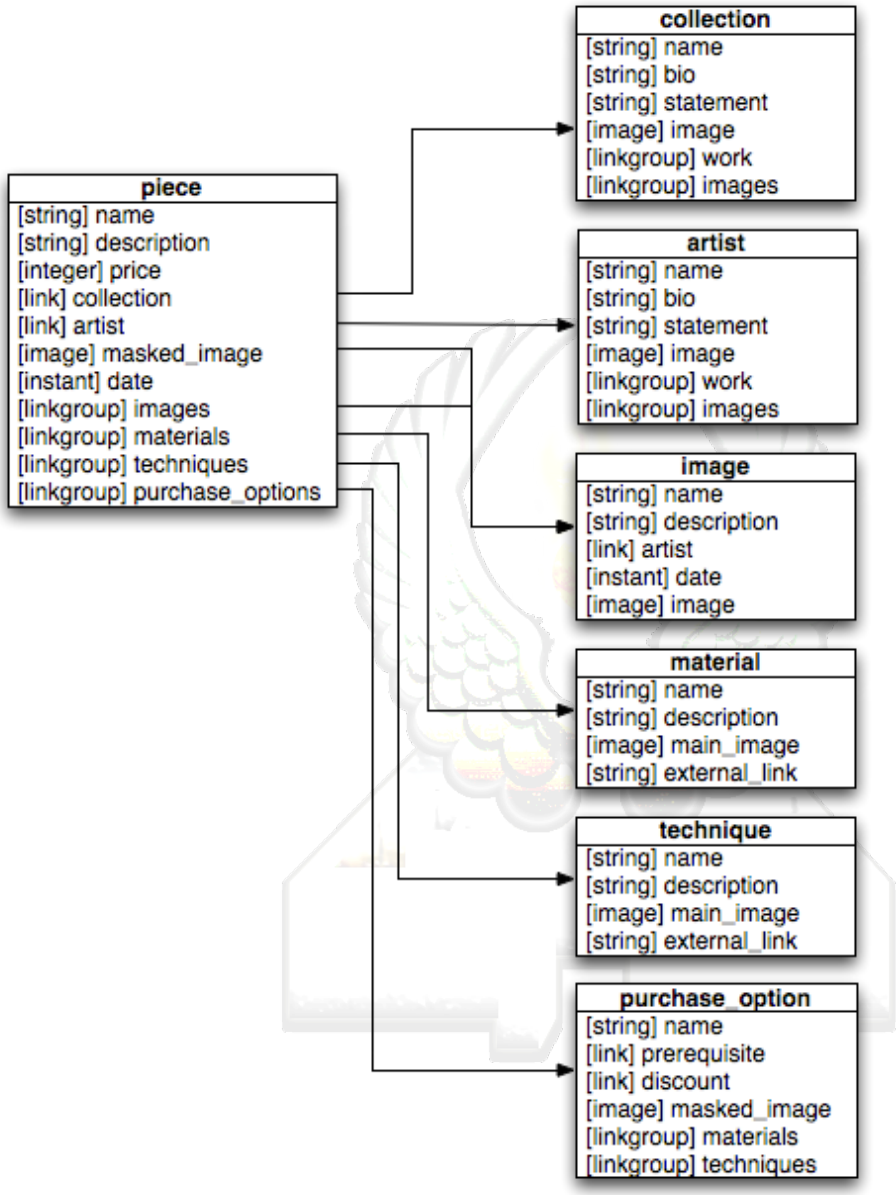
```
{wrapper_variable name="var_name" value=$value}
```


As an example we're going to assemble a web interface for an artist's web presence, assuming a fresh download of the DATALVS project.

7.3.1 Designing your site

TODO: Warn design is important
TODO: Discuss Commerce Objects

Knowing this, we now need to map out how the different data objects are going to relate with one another. The best way to do this is to lay out a diagram of how they interrelate, but keeping it as general as possible, without limiting what you'll be able to accomplish on the site you are working on. In my case, I may be working on a site to promote art jewelry, but I will keep it general enough, that the data configuration could be used for a wide range of artists. The basic organization is a bunch of 'pieces' made of 'materials' and 'techniques', but we want to cover variants, images for all these objects and for pieces to be part of a 'collection'. Given this criteria, I came up with:


piece
[string] name
[string] description
[integer] price
[link] collection
[link] artist
[image] masked_image
[instant] date
[linkgroup] images
[linkgroup] materials
[linkgroup] techniques
[linkgroup] purchase_options
collection
[string] name
[string] bio
[string] statement
[image] image
[linkgroup] work
[linkgroup] images
artist
[string] name
[string] bio
[string] statement
[image] image
[linkgroup] work
[linkgroup] images
image
[string] name
[string] description
[link] artist
[instant] date
[image] image
material
[string] name
[string] description
[image] main_image
[string] external_link
technique
[string] name
[string] description
[image] main_image
[string] external_link
purchase_option
[string] name
[link] prerequisite
[link] discount
[image] masked_image
[linkgroup] materials
[linkgroup] techniques

This autoloader takes care of classloading dynamicly as well as maintaining scope and registration. All you have to do to use it is use the java style _import
EX: _import('path.to.package.*');
or the path based syntax
EX: Autoloader::register_directory('./path/to/package/');
multiple copies of a identically named class can exist, but only one can be loaded during execution, so while this can be helpful when dealing with libraries which have colliding namespaces, it is no panacea.

11 Automaton

Automaton is an XML scriptable web scraper. It may also be extended by implementing a base class

- grab: pull a single value from a block of text
- sift: pull a set of results from the page
- pull: fetch a web page and put the result in the buffer
- set: set a form variable
- submit: submit a form and fetch the response
- respond: dump the last_response buffer to a registered callback function

12 UserLocale

An abstraction which handles geolocation as well as currency conversion using a variety of pluggable sources

Currency Converters:

- XE Currency Exchanger: A purchase-only data provider
- XRates.com: A free, though dated data provider

Geolocators:

- IP2Location: Commercial geolocation data provider
- IP2C: Commercial geolocation data provider
- PHP.net: Free geolocation data provider

13 ImageBooth

A Partial Implementation of Photoshop 3 in PHP. This class abstracts image manipulations functions into an Object Oriented abstraction layer. The purpose of this is to make working with images in code equivalent to using your favorite image editor. You can expect, for the near term my priority will be to implement most of what older releases of photoshop provide.

- Layers: Creation, Selection, Naming
- Sizing: Image and Canvas
- Filters: blur, threshold, greyscale, emboss, lines and edges
- Transformations: rotate
- Operations: color replace, brightness/contrast, negative
- Output: Inline/File (PNG, JPEG, GIF)

14 Grapher

An implementation independent charting abstraction.

- gRaphael: pull a set of results from the page
- Similie Timeplot: pull a set of results from the page
- Similie Line: fetch a web page and put the result in the buffer
- Fusion Charts: pull a single value from a block of text
- SWF Graphs: pull a single value from a block of text