Building Data-centric Websites with Dataface

Steve Hannah
Faculty of Applied Sciences
Simon Fraser University
shannah@sfu.ca
What does Data-centric Mean?

- Centred around the data of the site.
- Use a database to store data
- Separate the Logic from the Presentation
- Through-the-web editing of data
Examples of Data-centric Apps

- Weblogs
- Course management Apps (e.g., WebCT)
- Event / Volunteer Registration Apps
- Research profile systems

Any web site can be created in a Data-centric way by storing the content in a database, and using templates to display the content.
Advantages of Data-centric Design

- Reusability
- Maintainability
- Separates responsibility (Designers, Developers, and Content Owners)
- Improved information collaboration and sharing.
- Inherent benefits of database (searching, sorting, relationships, etc...)

Disadvantages of Data-centric Design

- More complicated to develop (HTML vs MySQL, PHP)
- May be difficult to add features after original development cycle.
- Portability. (Requires Database and Libraries to be installed)
Conventional Data-centric Development Procedures

- **Step 1:** Design database
- **Step 2:** Create forms for users to enter/edit data in the database
- **Step 3:** Create web pages which draw content from the database.
Which steps are harder?

- **Designing the database** is “easy” (i.e., very little “grunt” work)

- **Creating web pages** that draw data from the database is “easy” (e.g., you can reuse templates from site to site).

- **Building forms** to administer the application is “hard”. (repetitive – yet important!! Poorly designed forms can cause irreparable damages).
Enter Dataface

- Most of the administrative functionality of data-centric websites is the same from site to site.

- Dataface is a web application framework that factors out all of this common functionality.
Dataface Core Technologies

- PHP 4/5, MySQL 3.23/4
- Uses PEAR class libraries:
  - HTML_QuickForm
  - SQL_Parser
  - XML_Serializer, and more..
- Smarty Template Engine
- FCKEditor (HTML Editor)
Data-centric Design with Dataface

- **Step 1**: Design database
- **Step 2**: Decorate database application
- **Step 3**: Design web pages which draw content from database.
Example Dataface Application

- A Group Content Management System.
- Researchers maintain personal profile and publication lists.
- Research groups maintain projects, websites, publications, and members.
Example Application: Finished Product

- Login page
Example Application: Finished Product (2)

- Super-user's Homepage
Example Application: Finished Product (3)

- Regular user's homepage
Example Application: Finished Product (4)

- Edit Personal Profile
Example Application: Finished Product (5)

- Manage Publications
A Simple Dataface Application

```php
<?
require_once '/pub_html/dataface/dataface-public-api.php';
df_init(__FILE__, 'http://fas.sfu.ca/fas/dataface');

$app =& Dataface_Application::getInstance();
$app->display();

?>
```
require_once '/pub_html/dataface/dataface-public-api.php'
// Load the Dataface Public API

df_init($__FILE__, 'http://fas.sfu.ca/fas/dataface');
// Initialize the application

$app =& Dataface_Application::getInstance();
// Obtain a reference to the Application object

$app->display();
// Displays the application
What about the database connection info? (e.g., Host name, Username, Password, etc...) – We include a file called `conf.ini` in the same directory.
Example *conf.ini* file

```
[_database]
host = localhost
user = root
password = mypass
name = simple_app_db

[_tables]
Profiles = Profiles
Addresses = Addresses
Appointments = Appointments
```
conf.ini file Annotated

- **[_database]** section contains connection information for connecting to the MySQL database.

- **[_tables]** section lists the tables that should be included in the application navigation menu.
  - TableName = Table Label
Building an Application

- We will build a simple single-table application to manage user profiles.

- Profiles will need to contain data such as usernames, password, birth dates, personal profile blurbs, modification dates, etc...

- We won't deal with security yet.
Step 1: Designing the Database

- First step is always to design the database
- Pick your poison (PHPMyAdmin, Direct SQL commands, other DB Admin apps).
- Example using PHPMyAdmin:
Step 1: Designing the Database (SQL)

- Resulting SQL for PHPMyAdmin table creation:

```sql
CREATE TABLE `Profile` (
    `ProfileID` INT(11) NOT NULL AUTO_INCREMENT,
    `Username` VARCHAR(64) NOT NULL,
    `Password` VARCHAR(64) NOT NULL,
    `Email` VARCHAR(128),
    `BirthDate` DATE,
    `Blurb` TEXT,
    `Gender` ENUM('Male', 'Female') NOT NULL,
    `LastModified` TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    PRIMARY KEY (`ProfileID`),
    INDEX (`Email`),
    UNIQUE ( `Username` )
) TYPE = MyISAM ;
```
Step 2: Make Web App

- We will make a directory for our application. We'll call it `simple_app`.

- Create `index.php` and `conf.ini` files inside our `simple_app` directory.

- Add `.htaccess` file to `simple_app` directory to make sure that the `conf.ini` file is not served to the web. THIS IS IMPORTANT.
Step 2: Make Web App
(index.php file)

- The index.php file serves as the access point for the application.

```php
<?
require_once '../dataface/dataface-public-api.php';
df_init(__FILE__, 'http://localhost/~shannah/dataface');

$app =& Dataface_Application::getInstance();
$app->display();

?>
```
Step 2: Make Web App (conf.ini file)

- The *conf.ini* file goes in the same directory as the *index.php* file.

```
[_database]
host = localhost
user = root
password =
name = simple_app

[_tables]
Profile = Profile
```
Step 2: Make Web App (.htaccess file)

- The `conf.ini` file contains sensitive database connection information and should not be served by the web server.
- Include an .htaccess file in the `simple_app` directory to tell Apache NOT to serve .ini files:

```<FilesMatch "\.ini$"> Deny from all </FilesMatch>```
Using the Web App

- The application is now ready to use! Let's take a look.
- Point browser to the *index.php* file:
What now?

- The application looks kind of boring. What can we do with it?
  - Create new records
  - Edit and delete records
  - Search for records
  - Browse through the records

- First let's create a new record.
Creating a new record (1)

- Select *new record* from the *actions to be performed* menu.
Creating New Record (2)
Step 3: Decorating the Application

- This basic application is functional, but we can do better.

- Goals of Decorating:
  - More user friendly
  - More secure
  - More features
Step 3: Decorating cont'd (The *tables* directory)

- Follow these steps (we'll explain and generalize later):
  - 1. Create a directory named *tables* inside the *simple_app* directory.
  - 2. Create a directory named *Profile* inside the *tables* directory.
  - 3. Create a file named *fields.ini*, and place it inside the *Profile directory*. 
Step 3: Decorating (cont'd)
Application Directory Structure

- Your application directory structure should now look like this*:

  ![Image of directory structure]

  * There is also an .htaccess file that is not shown in this image because OSX finder hides files beginning with ".".

- Note the naming convention. The **Profile** directory is named after the **Profile** table. All files inside this directory will pertain to the **Profile** table.
- If we wanted to decorate a table named **foo** we would create a directory named **foo**.
The fields.ini file (in the Profile directory) contains additional settings for the Profile table and its fields.

Let's change the label for the BirthDate field from “BirthDate” to “Birth Date” (i.e., put a space between “Birth” and “Date” so it reads better by adding the following to the fields.ini file:

```
[BirthDate]
widget:label = Birth Date
```
Step 3: Decorating (cont'd)
Changing Field Label

- The BirthDate field before:
  ![BirthDate field before]

- The BirthDate field after:
  ![BirthDate field after]

  - Notice the space between “Birth” and “Date”
Step 3: Decorating (cont'd)
Adding Field Descriptions

- Let's add a note to the field so the user knows how to format the date.
- Modify the fields.ini file so it now contains:
  
  ```ini
  [BirthDate]
  widget:label = Birth Date
  widget:description = "YYYY-MM-DD"
  ```

- Now the BirthDate field looks like:
Step 3: Decorating (cont'd) Customizing the Widget type

- On second thought, let's use pull-down menus to select the date, rather than a text field.
- Change the fields.ini file so it contains:

  [BirthDate]
  widget:label = Birth Date
  widget:type = date

- Now the BirthDate field looks like:
Available (Simple) Widget Types

- **Text** – text field
- **Textarea** – Text Area
- **Htmlarea** – HTML Editor (FCKEditor)
- **Date** – Month/Day/Year pull-down lists
- **Select** – Select List*
- **Checkbox** – Either a single checkbox or a checkbox group
- **Autocomplete** – Textfield the automatically completes input based on a value list.
- **Hidden** – A hidden field
- **Static** – Uneditable field
[Username]
widget:description = "Unique user name to log into the system."

[Password]
widget:description = "Minimum 6 characters"

[Email]
widget:description = "e.g., john_doe@foo.com"

[BirthDate]
widget:label = Birth Date
widget:type = date

[Blurb]
widget:type = htmlarea

[LastModified]
widget:type = static
Vocabularies and Value lists

- Valuelists provide vocabularies that can be used by selection fields (e.g., checkbox groups, and select lists).
- Valuelists are defined in the `valuelists.ini` file.
- Each table folder may have its own `valuelists.ini` file.
- You may place a global `valuelists.ini` file in the site's directory to define valuelists to be used by all tables.
Let's define some valuelists for the Profile table in our application. Do the following:
1. create a file named valuelists.ini inside the Profile directory. Your application directory structure will now look like:
Adding Valuelists to Table (Example) cont'd...

- Place the following in the `valuelists.ini` file:

  ```ini
  [Colors]
  red = Red
  green = Green
  blue = Blue
  ```

- This defines a value-list named `Colors` that can be used as a vocabulary for select, checkbox, and autocomplete fields.
Adding Favourite Colour Field

- We want to add a field to the Profile table for the user to enter his favourite colour.

- Steps:
  1. Add field to table using PHPMyAdmin (or SQL).
     
     ```sql
     ALTER TABLE `Profile` ADD `FavouriteColour` VARCHAR(32) AFTER `Email`
     ```

- The FavouriteColour field now looks like:
Using the select Widget

- We want the user to choose his favourite colour from a list of the colors in the Colors value-list.

- Add the following to the fields.ini file:

  ```ini
  [FavouriteColour]
  widget:type = select
  vocabulary = Colors
  ```

  The vocabulary attribute means that the select list should use the value list named Colors for its options.
Using the `select` Widget (cont'd)

- Now the `FavouriteColour` field looks like:

- The HTML source generated for this `select` list is:

```html
<select class="default" id="FavouriteColour" name="FavouriteColour">
  <option value="">Please Select...</option>
  <option value="red">Red</option>
  <option value="green">Green</option>
  <option value="blue">Blue</option>
</select>
```
Dynamic Value-lists

- Add a 'SupervisorID' field to the 'Profile' table to track who supervises who.

- We want to be able to select a Profile from a list of available profiles in the 'SupervisorID' field.

- Use a dynamic value-list whose values are drawn from the database.

- Add the following to the valuelists.ini file:

  ```ini
  [Profiles]
  __sql__ = "SELECT ProfileID, Username FROM Profile ORDER BY Username"
  ```
Dynamic Value-lists (cont'd)

- Add following to fields.ini file:

  [SupervisorID]
  widget:type = select
  vocabulary = Profiles

- Look at the changes in the application:
Validation

- Client-side and Server-side validation handled by Dataface.

- Fields designated “NOT NULL” in SQL table definition are automatically required fields.

- Making 'Email' a required field using fields.ini file:

  [Email]
  widget:description = "e.g., john_doe@foo.com"
  validators:required = true
  validators:required:message = "You have to enter an email address"

- Email field now is required (note the red dot):
Now, if we try to save form with 'Email' field empty, we get error message:
More Validation Rules

- We can validate using other rules too:
  - required
  - maxlength
  - rangelength
  - regex
  - email
  - emailorblank
  - lettersonly
  - alphanumeric
  - numeric
  - nopunctuation
  - nonzero
Validation Examples

- **Regex:**

  ![Username]
  widget:description = "Unique user name to log into the system."
  validators:regex = "/^shannah$/"

  The above example is unrealistic but it accepts only “shannah” as input for the Username field.

- **Require a valid email address for the Email field:**

  ![Email]
  widget:description = "e.g., john_doe@foo.com"
  validators:email = true
Relationships

- We previously added a 'SupervisorID' field to the Profile table to store the supervisor of a record.

- What if we want to see the profiles *supervised* by the current profile.

- We will add a relationship to the 'Profile' table called 'Supervised'
Defining a Relationship

1) Create a file named 'relationships.ini' in the tables/Profile folder. The directory structure of the application should now look like:
1) Add the following to the relationships.ini file:

```
[Supervised]
__sql__ = "SELECT * FROM Profile WHERE SupervisorID = '$ProfileID'"
```

This says that Profiles whose 'SupervisorID' field match the ProfileID of the 'current' record are part of the 'current' record's 'Supervised' relationship.
1) Check the changes in web browser:

Notice the “Supervised” tab. This will show a list of all “Supervised” profiles of the current profile.
What can we do with relationships?

- Add new records to relationship
- Add existing records to relationship (for many-to-many relationships only).
- Remove records from relationship
Want more customization?

- What if we want to customize the behavior of our application?
  - Use custom templates and style sheets to customize Look & Feel
  - Add configuration directives in conf.ini / fields.ini files to enable/disable features
  - Use Delegate Classes to add permissions, display, import/export, custom serialization, calculated fields, and more.
More Information

- Dataface Documentation: http://www.fas.sfu.ca/dataface/documentation
- Sign up for Dataface mailing list: http://lists.sourceforge.net/lists/listinfo/dataface-users