

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

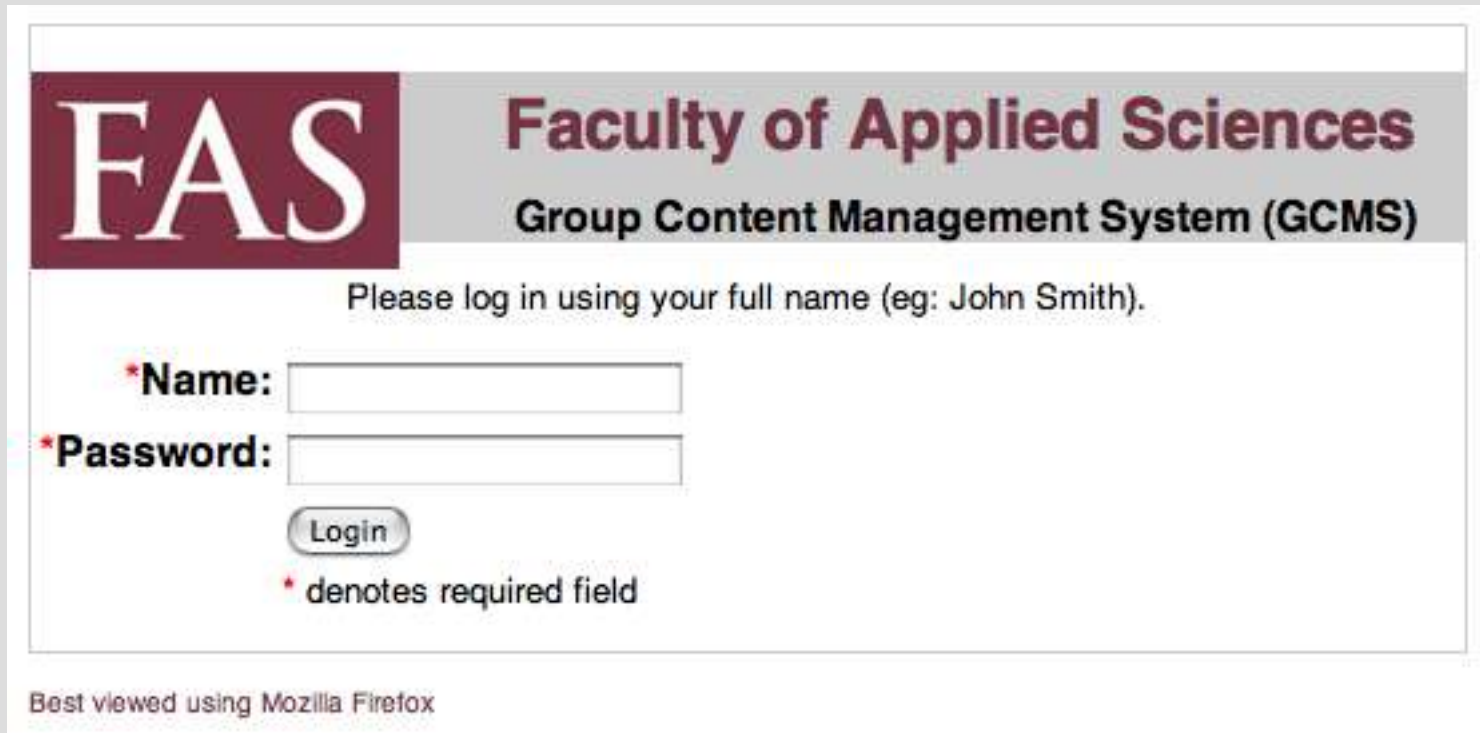
- **Step 1:** Design database
- **Step 2:** Decorate database application
- **Step 3:** Design web pages which draw content from database.

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



The screenshot shows a login page for the Faculty of Applied Sciences (FAS) Group Content Management System (GCMS). The page features a maroon header with the FAS logo and the system name. Below the header, there is a instruction to log in with a full name. Two input fields are provided for 'Name' and 'Password', both marked as required. A 'Login' button is positioned below the password field. A legend indicates that an asterisk denotes a required field. At the bottom, a note suggests the page is best viewed using Mozilla Firefox.

FAS Faculty of Applied Sciences
Group Content Management System (GCMS)

Please log in using your full name (eg: John Smith).

*Name:

*Password:

Login

* denotes required field

Best viewed using Mozilla Firefox

Example Application: Finished Product (2)

- Super-user's Homepage

The screenshot displays the Super-user's Homepage for the GCMS (Group Content Management System). At the top left is the FAS logo. The main header area shows "Found 256 of 256 Records in table Groups." and a "Jump: 1 to 30" dropdown menu. A search bar with a "Submit" button is located on the right. Below the header, there are tabs for "details", "list", and "find". A green bar indicates "actions to be performed". The main content area is titled "Welcome to GCMS (Group Content Management System)" and lists "Available Actions" categorized into "Personal", "Group", and "Web pages".

Navigation:

- Groups
- News
- Personal
- Publications
- Web Pages
- Web Sites

gcms menu:

- Home
- Edit My Profile
- View My Publications
- View My Groups

Available Actions:

- Personal**
 1. [Edit your profile](#)
 2. [Add Personal Publication](#)
- Group**
 1. Edit Group Profile:
 2. Add New Publication to group:
 3. Add Existing Publication to group:
 4. Add New Member to group:
 5. Add Existing Member to group:
 6. Add New News Item to group:
- Web pages**
 1. Edit Web page:

Example Application: Finished Product (3)

- Regular user's homepage

The screenshot displays the GCMS (Group Content Management System) homepage for a regular user. At the top left is the FAS logo. In the top right corner, there is a search bar with the text "Search:" and a "Submit" button. Below the logo, the text "Welcome to GCMS (Group Content Management System)" is displayed. The main content area is titled "Available Actions" and is divided into three sections: "Personal", "Group", and "Web pages".

gcms menu

- Home
- Edit My Profile
- View My Publications
- View My Groups
- Log out

Available Actions

Personal

1. [Edit your profile](#)
2. [Add Personal Publication](#)

Group

1. Edit Group Profile:
2. Add New Publication to group:
3. Add Existing Publication to group:
4. Add New Member to group:
5. Add Existing Member to group:
6. Add New News Item to group:

Web pages

1. Edit Web page:

Example Application: Finished Product (4)

- Edit Personal Profile

FAS << Go Back Search: Submit

Recent Records: Brian Funt

Brian Funt

gcms menu

- Home
- Edit My Profile
- View My Publications
- View My Groups
- Log out

main publications groups

Personal

First name

Last name

Contact numbers

	Name	Number
Delete	Office	555-555-5555

Add Row

Email address
(eg: john_doe@sfu.ca)

Degree(s)
One degree per line. (eg: Ph.D. University of Simon Fraser, 2003)

B.Sc. University of British Columbia, 1971
M.Sc. University of British Columbia, 1973
Ph.D. University of British Columbia, 1976

Description of Research
A short description of the research that this person is involved with.

Source

Example Application: Finished Product (5)

- Manage Publications

FAS << Go Back Search: Submit

Recent Records: Brian Funt

Brian Funt

gcms menu

- Home
- Edit My Profile
- View My Publications
- View My Groups
- Log out

main publications groups

Found 109 Records in relationship *publications*

Now Showing 1 to 30 (Display Records per page) Next: 31 to 60 >>

	pubid	publication type	publication date	bibliographic citation
<input type="checkbox"/>	1293	Refereed Article	2005-12-07 14:00:10	Funt, B., and Finlayson, G. 'The State of Computational Color Constancy,' Proceedings First Intersociety Color Council Panchromatic Conference, 38-39, Feb. 1995.
<input type="checkbox"/>	1294	Refereed Article	2005-12-07 14:00:10	Finlayson, G., and Funt, B., 'Optimal Spectral Sharpening,' Proceedings First Intersociety Color Council Panchromatic Conference, page 41, Feb. 1995.
<input type="checkbox"/>	1295	Refereed Article	2005-12-07 14:00:10	Funt, B.V., 'Experiential Reasoning (Abstract),' 1992 AAAI Spring Symposium on Reasoning with Diagrammatic Representations, AAAI, Mar. 1992.
<input type="checkbox"/>	1296	Refereed Article	2005-12-07 14:00:10	B. Funt, K. Barnard, M. Brockington and V. Cardei, 'Luminance-Based Multi-Scale Retinex,' Proc. AIC Color 97, Vol.I, 330-333, Kyoto, Japan, May 1997.
<input type="checkbox"/>	1297	Refereed Article	2005-12-07 14:00:10	B. Funt, V. Cardei and K. Barnard, 'Neural Network Color Constancy and Specularly Reflecting Surfaces,' Proc. AIC Color 97, Vol.II, 523-526, Kyoto, Japan, May 1997.
<input type="checkbox"/>	1298	Refereed Article	2005-12-07 14:00:10	V. Cardei, B. Funt and K. Barnard, 'Modeling Color Constancy with Neural Networks,' Proc. Int. Conf. on Vision, Recognition, and Action: Neural Models of Mind and Machine, Boston, May 29-31, 1997.
<input type="checkbox"/>	1299	Refereed Article	2005-12-07 14:00:10	Barnard, K. and Funt, B., 'Analysis and Improvement of Multi-Scale Retinex,' Proc. Fifth IS&T Color Imaging Conference, Scottsdale 1997.
<input type="checkbox"/>	1300	Refereed Article	2005-12-07 14:00:10	Finlayson, G. D., Dueck, J., Funt, B.V., Drew, M.S., 'Colour Eigenfaces,' Proc. Third International Workshop on Image and Signal Processing Advances in Computational Intelligence, November 4-7, 1996, Manchester, UK.
<input type="checkbox"/>	1301	Refereed Article	2005-12-07 14:00:10	Finlayson, G.D., Chatterjee, S.S., and Funt, B.V., 'Colour-Texture Indexing,' Proc. IEE Colloquium on Image Databases, 1996.
<input type="checkbox"/>	1302	Refereed Article	2005-12-07 14:00:10	Barnard, J., Finlayson, G., Funt, B., 'Colour Constancy for Scenes with Spectrally Varying Illumination,' ECCV96 Fourth European Conference on Computer Vision, Vol. II, pages 3-15, April 1996.
<input type="checkbox"/>	1303	Refereed Article	2005-12-07 14:00:10	Finlayson, G., Chatterjee, S., and Funt, B., 'Colour Angular Indexing,' ECCV96 Fourth European Conference on Computer Vision, Vol. II, pages 16-27, April 1996.
<input type="checkbox"/>	1304	Refereed Article	2005-12-07 14:00:10	B. Funt, V. Cardei and K. Barnard, 'Learning Color Constancy,' Proc. IS&T/SID Fourth Color Imaging Conference: Color Science, Systems and Applications, pp. 58-60, Scottsdale, Arizona, November 1996.
<input type="checkbox"/>	---	Refereed	2005-12-07	Finlayson, G., and Funt, B., 'Color Constancy Under a Varying Illumination,' Proceedings Fifth

A Simple Dataface Application

```
<?
```

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

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

```
?>
```

Line-by-line Annotation

```
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
```

Configuration Info

- 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:

Field	Type ?	Length/Values*	Collation	Attributes	Null	Default**	Extra				
ProfileID	INT	11			not nul		auto_increment	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Username	VARCHAR	64			not nul			<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Password	VARCHAR	64			not nul			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Email	VARCHAR	128			null			<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
BirthDate	DATE				null			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Blurb	TEXT				null			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
Gender	ENUM	'Male','Fem'			not nul			<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
LastModified	TIMESTAMP				not nul	<input checked="" type="checkbox"/> CURRENT_TIMESTAMP		<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

Step 1: Designing the Database (SQL)

- Resulting SQL for PHPMyAdmin table creation:

```
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`  
  ) ,  
  FULLTEXT (   
    `Email` ,  
    `Blurb`  
  )  
 ) 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.

```
<?
```

```
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:



The screenshot displays a web application interface. On the left is the Simon Fraser University crest with the motto 'VIVIS SOMMES PRIET'. Below it is a navigation menu with a folder icon and the text 'Profile'. The main content area shows a search result for 'Profile' with the message 'Found 0 of 0 Records in table Profile. Now showing 1 of 0'. A search bar with a 'Submit' button is visible. Below the search bar are tabs for 'details', 'list', and 'find'. A table header is visible with a tab labeled 'main' and a dropdown menu 'actions to be performed'. The table content area displays the message 'No records matched your request.'.

Found 0 of 0 Records in table *Profile*.
Now showing 1 of 0

Jump:

This Record:

Search:

details list find

actions to be performed ▾

main

No records matched your request.

Powered by Dataface
Dataface framework designed and developed by Steve Hannah, Faculty of Applied Sciences Web Services Developer (Simon Fraser University).
(c) 2005 All rights reserved GCMS (Group Content Management System) Developed by Faculty of Applied Sciences Web Team at Simon Fraser University
Stylesheets adapted from Plone 2.0.5 stylesheet

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)

SEARCH: SUBMIT

details list find

actions to be performed ▾

Profile

Username

Password

Email

BirthDate

Blurb

Gender

LastModified

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)

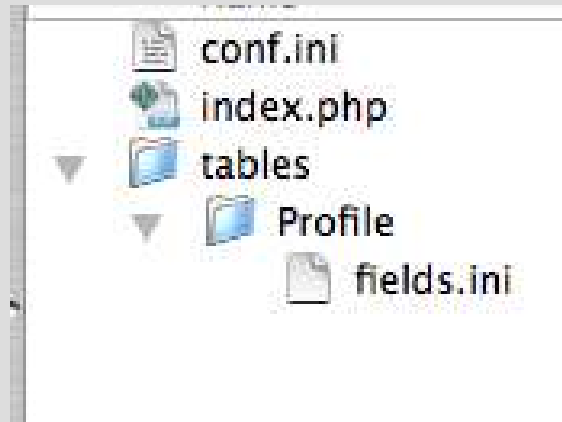
- Follow these steps (we'll explain and generalize later):
 - 1. Create a directory named *tables* inside the *simple_app* directory.
 - 2. Create a directory named *Profile* inside the *tables* directory.
 - 3. Create a file named *fields.ini*, and place it inside the *Profile* directory.

Step 3: Decorating (cont'd)

Application Directory Structure

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

* 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*.

Step 3: Decorating (cont'd) The *fields.ini* file

- 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

1978-12-27

- The BirthDate field after:

Birth Date

1978-12-27

- 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:

```
[BirthDate]
widget:label = Birth Date
widget:description = "YYYY-MM-DD"
```

- Now the BirthDate field looks like:



Birth Date
YYYY-MM-DD
1978-12-27

A screenshot of a web form field. The field is titled "Birth Date" in bold. Below the title is the format "YYYY-MM-DD". The field itself is a text input box containing the date "1978-12-27".

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:



The image shows a graphical user interface element for a date field. It is titled "Birth Date" in a bold, black font. Below the title are three separate pull-down menu boxes. The first box contains the number "27", the second box contains the text "Dec", and the third box contains the year "2001". Each box has a small downward-pointing arrow on its right side, indicating that the content can be selected from a list.

Available (Simple) Widget Types

- ***Text*** – text field
- ***Textarea*** – Text Area
- ***Htmllarea*** – 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

fields.ini file Changes...

[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

Results...

Profile


Username ■
Unique user name to log into the system.

Password
Minimum 6 characters

Email
e.g., john_doe@foo.com

Birth Date

Blurb



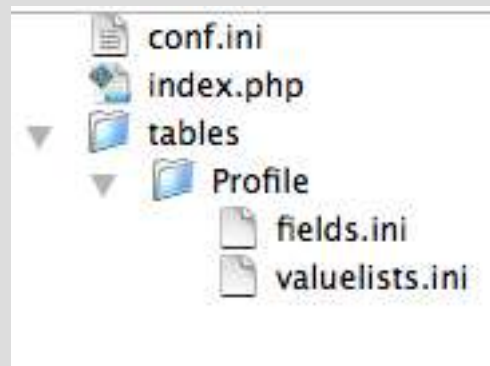
Blah blah blah

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.

Adding Valuelists to Table (Example)

- 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:

```
[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 PHPMysqlAdmin (or SQL).
`ALTER TABLE `Profile` ADD `FavouriteColour` VARCHAR(32) AFTER `Email``
- The FavouriteColor field now looks like:



A screenshot of a web form field. The label 'FavouriteColour' is positioned above a single-line text input box. The input box is empty and has a thin border.

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:

```
[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:

```
<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:

[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):

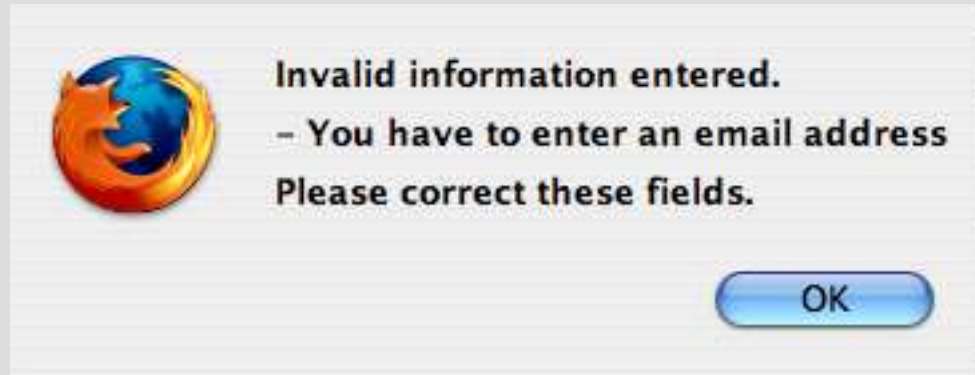
Email ■

e.g., john_doe@foo.com

shannah@sfu.ca

Validation (cont'd)

- 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
 - letteronly
 - 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"

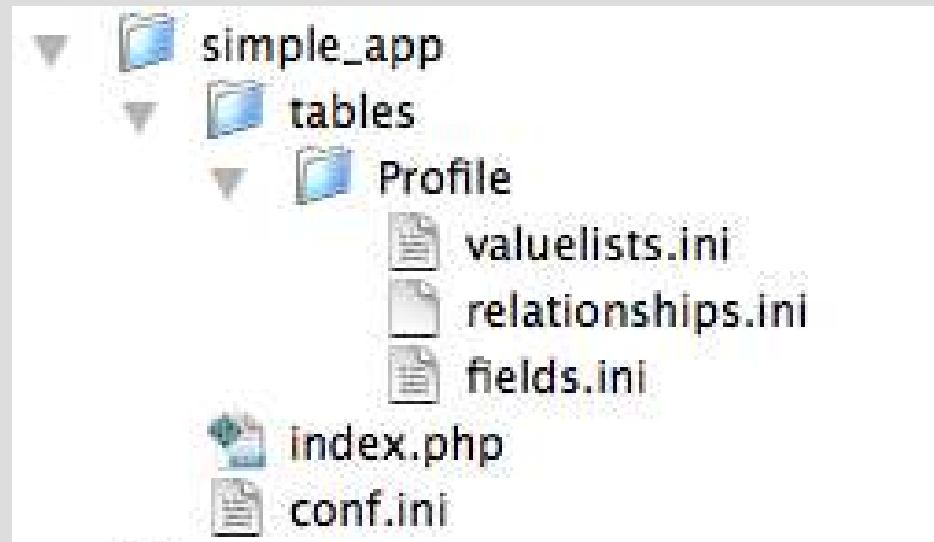
validators:email = true

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:



Defining a Relationship (2)

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.

Defining a Relationship (3)

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>