

MODELBAKER 2.1

The User's Manual

Last Updated: August 2, 2011



Widget Press, Inc.

Copyright © 2011 Widget Press, Inc. All rights reserved. Widget Press and the Widget Press logo are trademarks of Widget Press, Inc. ModelBaker is a trademark of Widget Press, Inc.

Acknowledgements

Portions of this Widget Press Software may utilize the following copyrighted material, the use of which is hereby acknowledged:

Cake Software Foundation - (CakePHP™) Rapid Development Framework
www.cakephp.org Copyright © 2005-2011, Cake Software Foundation, Inc.

CONTENTS

<i>Welcome to ModelBaker</i>	7
<i>ModelBaker Features at a Glance</i>	7
<i>Resources for Learning More</i>	8
<i>ModelBaker Setup</i>	9
<i>Installing ModelBaker</i>	9
<i>Installing Apache, MySQL and PHP</i>	9
<i>Beginning With ModelBaker</i>	11
<i>Configuration</i>	11
<i>Web Frameworks</i>	11
<i>Extending ModelBaker</i>	12
<i>Using the Extensions Manager</i>	13
<i>An Overview of ModelBaker</i>	14
<i>The Models Tab (⌘1)</i>	15
<i>The Controllers Tab (⌘2)</i>	17
<i>The Views Tab (⌘3)</i>	20
<i>The Settings Tab (⌘4)</i>	21
<i>Creating a Web Application</i>	22
<i>Step 1: Starting with New Project or Template</i>	22
<i>ModelBaker - The User's Manual</i>	3

<i>Step 2: Creating an Entity Model</i>	23
<i>Step 3: Adding Attributes</i>	24
<i>Step 4: Adding Validation Rules</i>	25
<i>Step 5: Saving Your Project</i>	26
<i>Step 7: Adding Database Credentials</i>	27
<i>Step 8: Dropping & Inserting Tables</i>	28
<i>Step 9: Building Your Web App</i>	29
<i>Entity Model Basics</i>	30
<i>Adding an Entity Model</i>	30
<i>Adding an Attribute with Type</i>	31
<i>Adding Custom Database Attribute Types</i>	34
<i>Adding Validation Rules</i>	37
<i>Creating File Uploads</i>	43
<i>Step 1. Create an Entity Model and Attribute 'filename'</i>	43
<i>Step 2. Adding Validation Rules for 'filename'</i>	44
<i>Basic HTTP Authentication</i>	46
<i>Step 1: Setting up the Controller</i>	46
<i>Step 2: Setting the Action for Authentication</i>	47
<i>Admin Routing Actions</i>	48
<i>Step 1: Setting up the Admin Routing Name</i>	48
<i>Step 2: Setting Action to the Routing Name</i>	49

<i>Internationalization & Localization</i>	<i>50</i>
<i>Creating Custom Templates</i>	<i>51</i>
<i>Step 1: Creating a new ModelBaker project</i>	<i>51</i>
<i>Step 2: Adding project files to a new folder</i>	<i>52</i>
<i>Step 3: Creating an XML .plist file</i>	<i>52</i>
<i>Step 4: Creating a PNG graphic file</i>	<i>53</i>
<i>Step 5: Renaming a folder to a ModelBaker template file</i>	<i>54</i>
<i>Step 6: Moving the template file to “Application Support”</i>	<i>55</i>
<i>Step 7: Testing the template</i>	<i>56</i>
<i>Creating a Custom Theme</i>	<i>58</i>
<i>Step 1: Collecting your graphical assets</i>	<i>58</i>
<i>Step 2: Creating a new folder “RedRuby”</i>	<i>61</i>
<i>Step 3: Creating another new folder “RedRuby”</i>	<i>61</i>
<i>Step 4: Creating the theme directory structure</i>	<i>62</i>
<i>Step 5: Adding your assets</i>	<i>63</i>
<i>Step 6: Renaming the file with ModelBaker theme extension</i>	<i>65</i>
<i>Step 7: Adding the file to “Application Support”</i>	<i>66</i>
<i>Step 8: Testing the new theme</i>	<i>69</i>
<i>Moving to a Hosting Server</i>	<i>70</i>
<i>Step 1: Changing your Database Settings</i>	<i>70</i>
<i>Step 2: Uploading to your Hosting Site</i>	<i>71</i>
<i>Step 3: Installing your Database Schema</i>	<i>71</i>
<i>ModelBaker - The User’s Manual</i>	<i>5</i>

WELCOME TO MODELBAKER

With ModelBaker, impressive web applications are just the beginning. This section will provide an overview of ModelBaker's features and list resources for learning more.

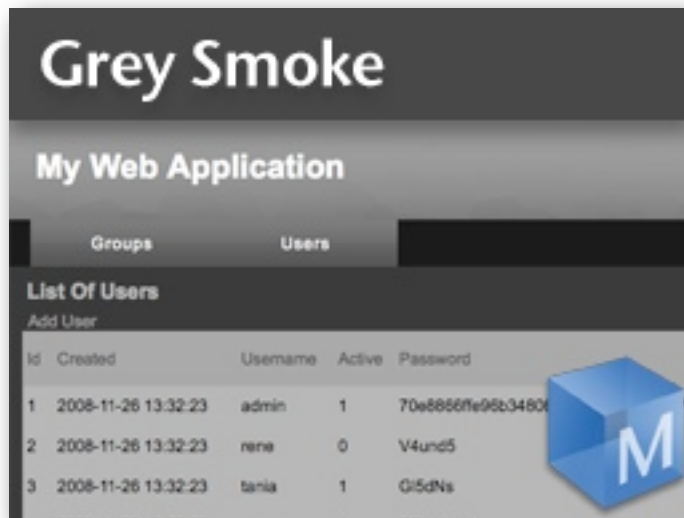
ModelBaker is a rapid development tool for create professional-quality web applications. ModelBaker delivers powerful applications backed by a database that persists your information.

ModelBaker Features at a Glance

The next few pages provide a high-level overview of the key ModelBaker features. The remainder of the manual provides the details on creating and building your web application.

PROFESSIONAL-QUALITY THEMES

ModelBaker provides seven high quality, professionally designed themes to help you get started working on your web projects. Every theme includes images, style sheets and the necessary javascript files. You can easily change a theme and apply those changes to other web applications. You can even modify the generated theme code in a text editor to add additional polish to your web app.



Each of the default themes is a CSS based layout, meaning the presentation styles are separated from the layout structure. ModelBaker's default theme

sets are: Blue Sapphire, Burnt Amber, Cocoa Citrine, Corn Flower Blue, Green Emerald, Grey Smoke, Purple Amethyst and Red Ruby.

MODEL BAKER TEMPLATES

Templates are standard ModelBaker projects that you can create and edit. Once you are satisfied with a project, you can save that template down and include it in ModelBaker as a foundation for future projects.

You can easily save your projects as ModelBaker templates by adding your “.mbtemplate” files to the “~/Library/Application Support/ModelBaker/Templates” folder.

Note that the file extension “.mbtemplate” is actually a folder which has been renamed and contains your “.modelbaker” project file.

Resources for Learning More

To get the most out of ModelBaker, please consult the following resources:

MODEL BAKER USER’S MANUAL

This full-color PDF document provides detailed information about ModelBaker features and instructions for working with your project.

ONSCREEN TOOLTIP HELP

To see the help, open ModelBaker and create a new project. You can hover over a sections and labels to receive information in a tooltip format.

WEB RESOURCES

Go to www.widgetpress.com/support to get the latest information about updates, articles, forums and how to guides. You can also purchase a ModelBaker license key on the web.

MODELBAKER SETUP

To make your development environment fully operational with ModelBaker, you will need to have the Apache web server, MySQL database engine, and PHP run time language module installed.

Installing ModelBaker

ModelBaker installation is very straight forward. Download ModelBaker to your Mac from the Mac App Store.

Installing Apache, MySQL and PHP

ModelBaker uses three open source software components to build your web applications: Apache, MySQL, and PHP. Since ModelBaker builds web applications, you will need to install and configure a web server (Apache), a database (MySQL) and a runtime language engine (PHP). There are different ways to install and setup these three components, but they each achieve the same end goal: **A web server that can dynamically generate content that is connected to a backend database.**

CUSTOM INSTALLATION

Providing installation instructions for Apache, MySQL, and PHP is beyond the scope of this manual. However, Mac OS X 10.5 and above does come with Apache and PHP already installed. If you are an experienced web developer, you will probably want to refer the documentation at <http://developer.apple.com> for additional tips.

INSTALLING USING MAMP AND MAMP PRO

A simple solution to install and configure Apache, MySQL and PHP is to use MAMP from Living-e. For detailed download and installation instructions, please visit <http://www.mamp.info/en/index.html>. When you download and install MAMP from Living-e, you get both versions: MAMP (free) and MAMP PRO (paid). MAMP PRO requires a paid license to operate. Please consult the MAMP PRO website for details on obtaining a license.

1. Download MAMP to your Mac.
2. Double-click the MAMP installation icon and follow the instructions.

3. After installation, note the location of the “/Applications/MAMP/htdocs” folder. For the free MAMP version, this will be your ‘Build Path’ for your ModelBaker projects. With the MAMP PRO version, you can serve ModelBaker projects from any folder.

INSTALLING USING XAMPP

Another simple solution to have all three components installed on your Mac is from Apache Friends called XAMPP. For detailed download and installation instructions, please visit the www.apachefriends.org website.

1. Download XAMPP to your Mac.
2. Double-click the XAMP installation package icon and follow the instructions.
3. After installation, note the location of the “/Applications/XAMPP/htdocs” folder. This folder will be your ‘Build Path’ for your ModelBaker projects. Note that this folder is actually an alias that points to the “/Applications/XAMPP/xamppfiles/htdocs” folder.

BEGINNING WITH MODELBAKER

In this chapter, we discuss the basic overall principles and ideas to keep in mind while developing projects with ModelBaker

When developing and testing your ModelBaker projects, your local requirements are to have a web server, a MySQL database, and the PHP runtime. ModelBaker generates code designed to run on top of the open source LAMP “Linux (our case the Mac), Apache, MySQL and PHP” web stack. This means you can deploy your applications to thousands of web hosting providers or deploy your application internally within your organization.

Configuration

ModelBaker assumes you have Apache, MySQL, and PHP configured and working together on your Mac. Apple offers Apache and PHP on the Mac OSX 10.5 (and higher) operating system, however you still need to set up and integrate MySQL.

There are downloadable packages, such as MAMP and XAMPP that offer Apache, MySQL, and PHP together that can be easily installed on your Mac. These packages simplify the process of trying to configure these three components.

ModelBaker looks for your local MySQL database and injects the schema based on the entity models you create. If your MySQL database server is not running or is not configured properly when you build your projects, then your database tables will not probably be created.

Web Frameworks

ModelBaker uses the CakePHP framework as the web deployment framework for your projects. CakePHP (<http://cakephp.org>) is an open source project and tens of thousands of developers have chosen CakePHP as their MVC web framework. All generated code from ModelBaker is open code, meaning you can edit and customize it within any text editor.

Extending ModelBaker

ModelBaker can be extended through a number of methods including style sheets and Javascript. Inside your “~/Library/Application Support/ModelBaker” folder are the directories where you add files to expand ModelBaker’s functionality.

Within the “~/Library/Application Support/ModelBaker/...”, you will see the following directories (not if you are using Mac OS X 10.7, the Library folder is hidden):

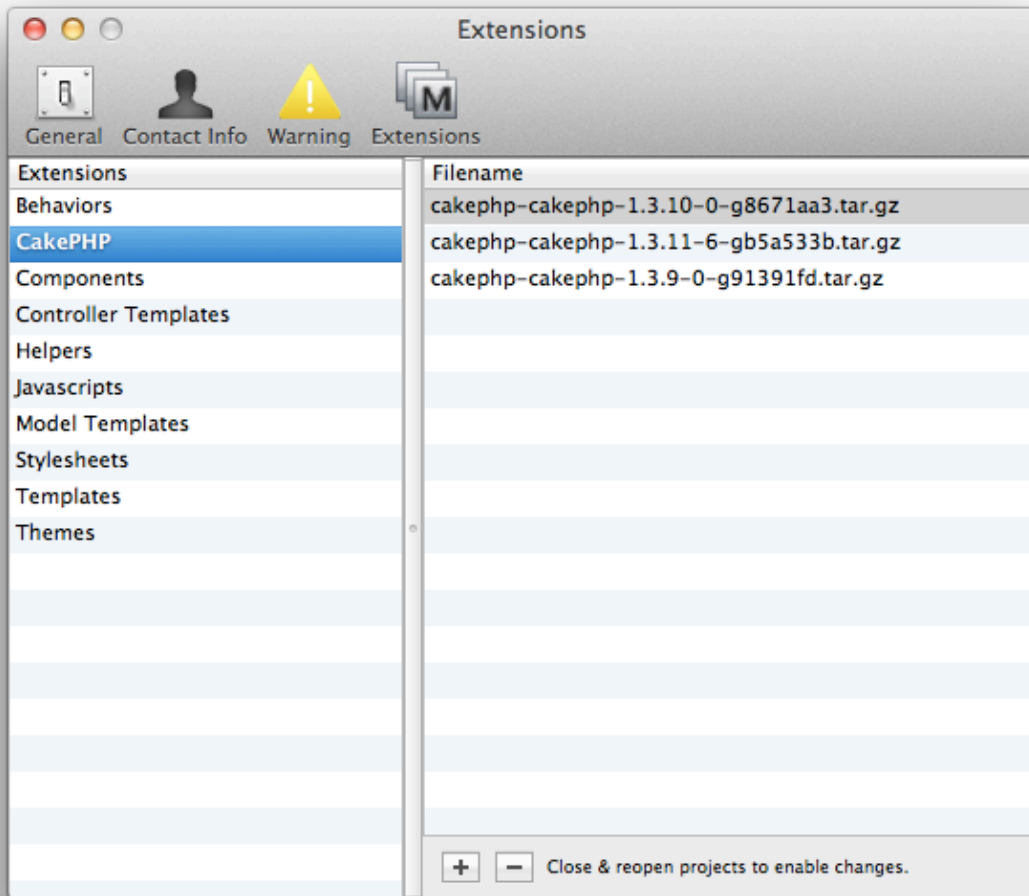
- Behaviors - Adds additional functionality to models. This folder holds the “.php” behavior files that will show up in the *Behaviors*’ tab, above the model browser, under *Models*’ (§1) in the main window. Must use “.php” file extensions”.
- CakePHP - Give you the ability to use a different CakePHP version build. Must use “.tar.gz” file extensions”. You can download a number of CakePHP versions from www.cakephp.org
- Components - Give your controllers additional functionality and behaviors. This folder holds the “.php” component files that you can use in multiple projects. These appear in the *Components*’ tab inside *Controllers*’ (§2) in the main window. If a component is selected, this component will be added to the build path’s “/app/controllers/components” directory. Must use “.php” file extensions”.
- Controller Templates - Offers the ability to use a custom controller that will replace the ModelBaker generated controller. If your controller is named “Autos” and there is a “autos_controller.php” located within this “Controller Templates” directory, then the checkbox *User Controller Template*’ will become selectable below the *Name*’ and *URL Path*’ textfields inside *Controllers*’ (§2) in the main window. During build, the “autos_controller.php” from the “Controller Templates” directory will be copied to your “/app/controllers/” directory. Must use “.php” file extensions”.
- Helpers - Gives the ability to extend and add behaviors to views. Any “.php” file in this directory will be listed under the *Helpers*’ tab in *Controllers*’ (§2) in the main window. During Build, any helper selected will be copied to the “/app/views/helpers” directory. Must use “.php” file extensions”.
- Javascripts - Gives you the ability to add custom Javascript scripts to views or layouts. Added scripts appear under the *Javascripts*’ tab in *Views*’ (§3) in the main window. By

adding Javascript files to this directory, you can select to have these files copied to “/app/webroot/js” directory during the Build process. Must use “.js” file extensions”.

- Model Templates - Offers the ability to use custom models that will replace the ModelBaker generated models in the “/app/models” directory. If an Entity model is named “Big Tree” and there is corresponding “big_tree.php” file located in the “Model Templates” folder, the ‘*Use Model Template*’ checkbox in ‘*Models*’ (⌘2) will be available. Must use “.php” file extensions”.
- Stylesheets - This directory can hold any custom Cascading Style Sheets for your web projects. Must use “.css” file extensions”.
- Templates - Gives you the ability to build new projects from custom ModelBaker projects. Templates are a way to take existing projects and use them over and over again. They follow a similar naming pattern to ‘Plugins’ but use the “.mbtemplate” extension. You can read more about templates at the [‘Creating Custom Template’](#) chapter. Must use “.mbtemplate” file extensions”.
- Themes - Simplifies changing of the look and feel of your projects. Adding themes follows the CakePHP design pattern for theming your project. A ModelBaker theme is actually a directory that has been renamed with the extension “.mbtheme”. Placing this file in the “Application Support/ModelBaker/Themes” folder makes the theme available to other ModelBaker projects. Must use “.mbtheme” file extensions”.

Using the Extensions Manager

From the Preferences (⌘,) Window, you can manage the Extensions by selecting the Extensions tab. You have the ability to add and remove extensions that will all ModelBaker projects. If you have any open ModelBaker projects during the adding or removal of Extensions, you will need to close and re-open the project to implement the changes.



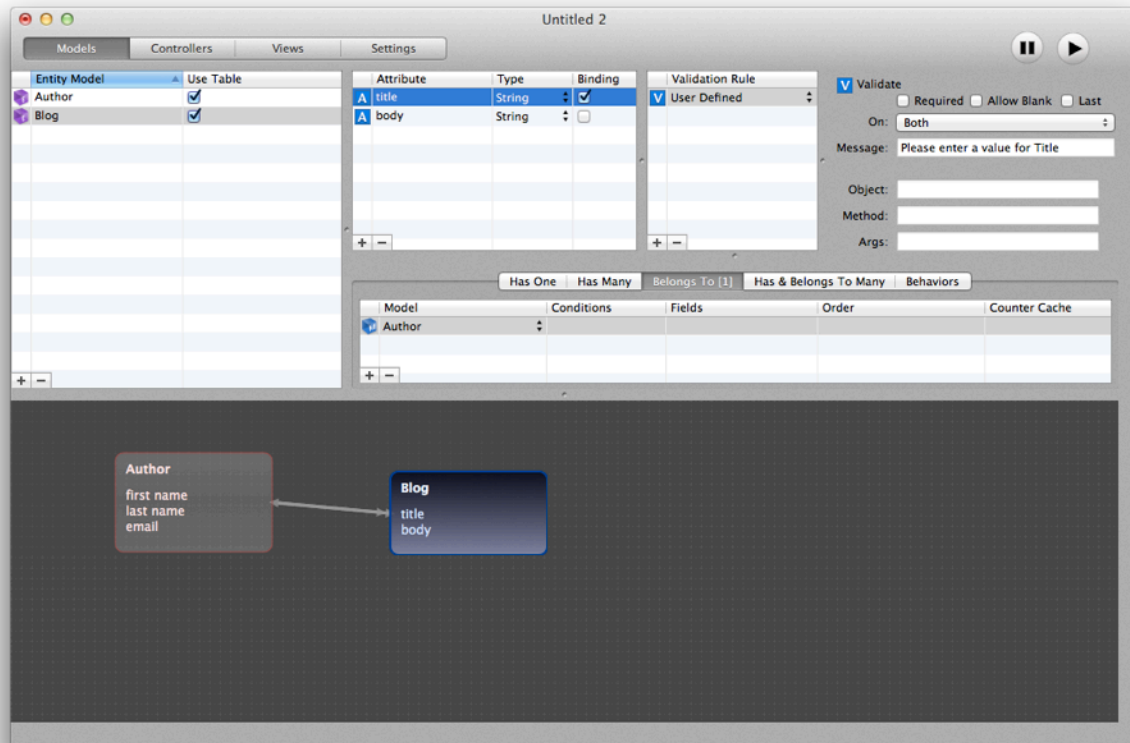
AN OVERVIEW OF MODELBAKER

*This chapter highlights the four main sections in ModelBaker:
Models, Controllers, Views and Settings.*

The Models Tab (§1)

MODELS

Models are the building blocks for your web application. They represent the data and business rules you will need for your application. ModelBaker makes creating models a one click process.



ModelBaker takes this a step further and abstracts your model into Entity Models. You can add attributes to your high level entity model, which will be used as database columns. When you define the model's attributes, all of the model classes and SQL code will be automatically generated.

Adding custom validations to your attributes is only a click away. Generated web apps will contain all the validations for your users.

Naming your entity model is very important. Entity models names should be in the singular format. 'Automobile', 'Person', and 'Jet Airplane' are valid entity model names, 'Automobiles', 'Persons', and 'Jet Airplanes' are not. If the 'Use Table' checkbox is selected, an entity

model named 'Jet Airplane' will generate (after a build) the database table 'jet_airplanes' with the any defined attributes as column names.

The 'Jet Airplane' entity model will also create a PHP class file titled "jet_airplane.php" located at "/app/models/jet_airplane.php". ModelBaker will generate the entity model's corresponding controller with all required actions views.

A T T R I B U T E S

Adding attributes to your entity model is, in affect, adding MySQL table columns. When users save information in your web application, this information will be saved in the columns of the database table.

A S S O C I A T I O N S

One of the powerful features of ModelBaker is the ability to link different models together through associations. Mapping your models through associations will automatically generate the necessary SQL and class model relationships.

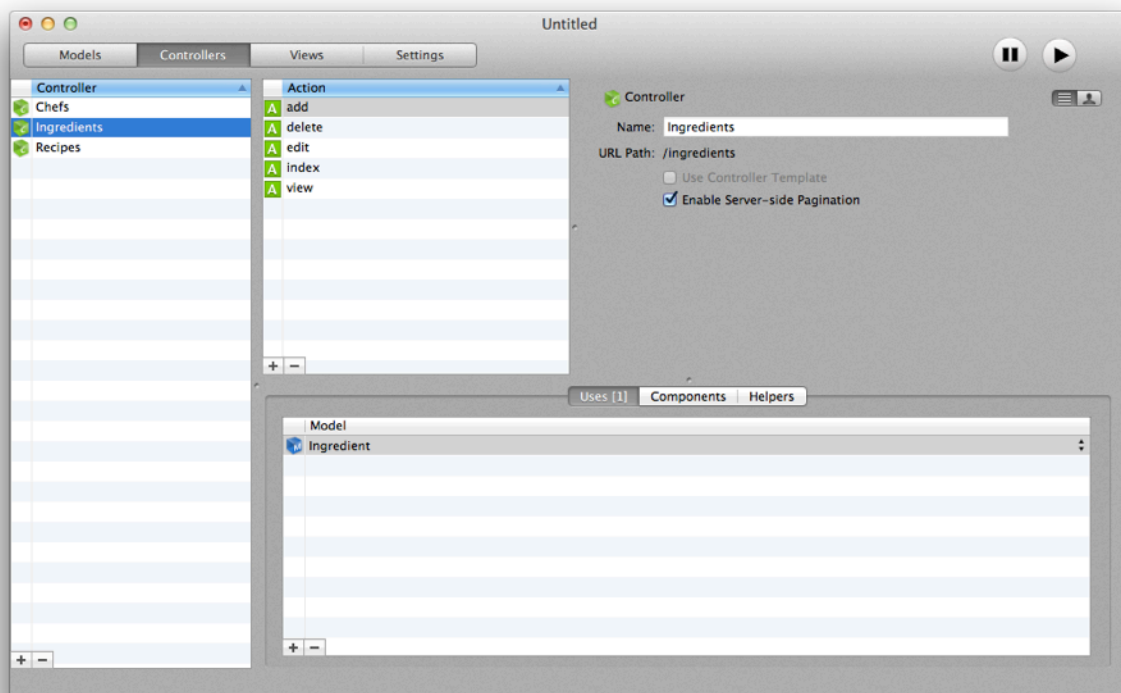
B E H A V I O R S

Behaviors are a way to organize "non-business" functionality and attach them to your models. ModelBaker provides access to CakePHP's default behaviors: ACL, Containable, Translate and Tree.

The Controllers Tab (⌘2)

CONTROLLERS

A controller is used to manage the logic for a part of your application. Most commonly, controllers are used to manage the logic for a single model. For example, if you were building a site for an online bakery, you might have a `RecipesController` and an `IngredientsController` managing your recipes and their ingredients, respectively. Since ModelBaker uses CakePHP, controllers are named after the model they handle, in plural form.



The Recipe model is handled by the 'RecipesController', the Ingredients model is handled by the 'IngredientsController', and so on.

Your application's controllers are classes that extend the CakePHP 'AppController' class, which in turn extends a core 'Controller' class. The 'AppController' class can be defined in `/app/app_controller.php` and should contain methods that are shared between all of your application's controllers. It extends the 'Controller' class which is a part of the standard CakePHP library.

Controllers can include any number of methods which are usually referred to as actions. Actions are controller methods used to display views. An action is a single method of a control-

ler. CakePHP's dispatcher calls actions when an incoming request matches the URL of a controller's action. Returning to our online bakery example, our 'RecipesController' might contain the 'view()', 'share()', and 'search()' actions.

COMPONENTS

Components are special packaged bundles of code that add additional functionality to controllers. These bundles can be shared across all controllers and can even be added to the base 'AppController' class.

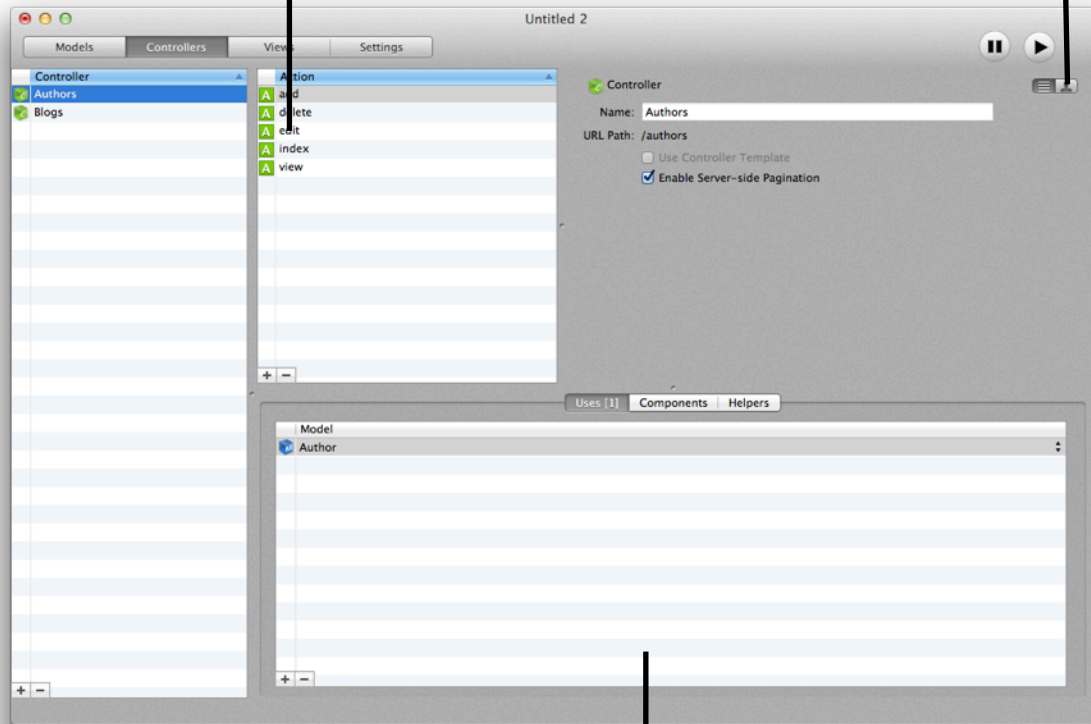
Using the foundation framework of CakePHP, ModelBaker can add the default components or your custom components to a project with a simple checkbox. The following built-in components are included to aid in the creation of your web app: Security, Sessions, Access Control Lists, Emails, Cookies, Authentication, and Request Handling.

HELPERS

Helpers are utility objects that assist with the presentation logic of the view layer. They are set from within the Controller layer to be pushed to the view. They can be shared among layouts, views, and elements. You can use CakePHP's built in helpers, extend them or create your own. ModelBaker currently ships with the Cache, Form, HTML, Js, Number, Paginator, RSS, Session, Text, Time and XML helper objects.

Actions are mapped to url
paths:
www.examples.com/author

Basic HTTP Authentication

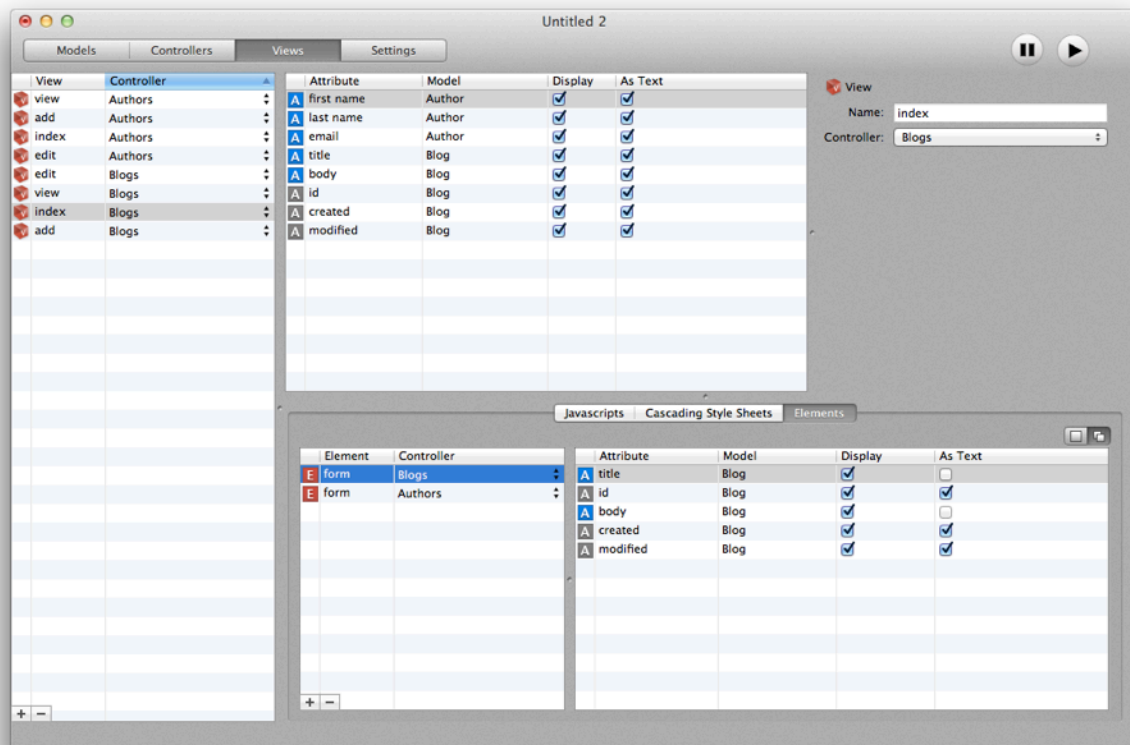


Uses, Components and Helpers

The Views Tab (⌘3)

VIEWS

The view layer of your web application is what users see and interact with. Most of the time, your application will be serving XHTML to your web and mobile users, but it can also deliver “.CSV” or even straight XML like an RSS feed.

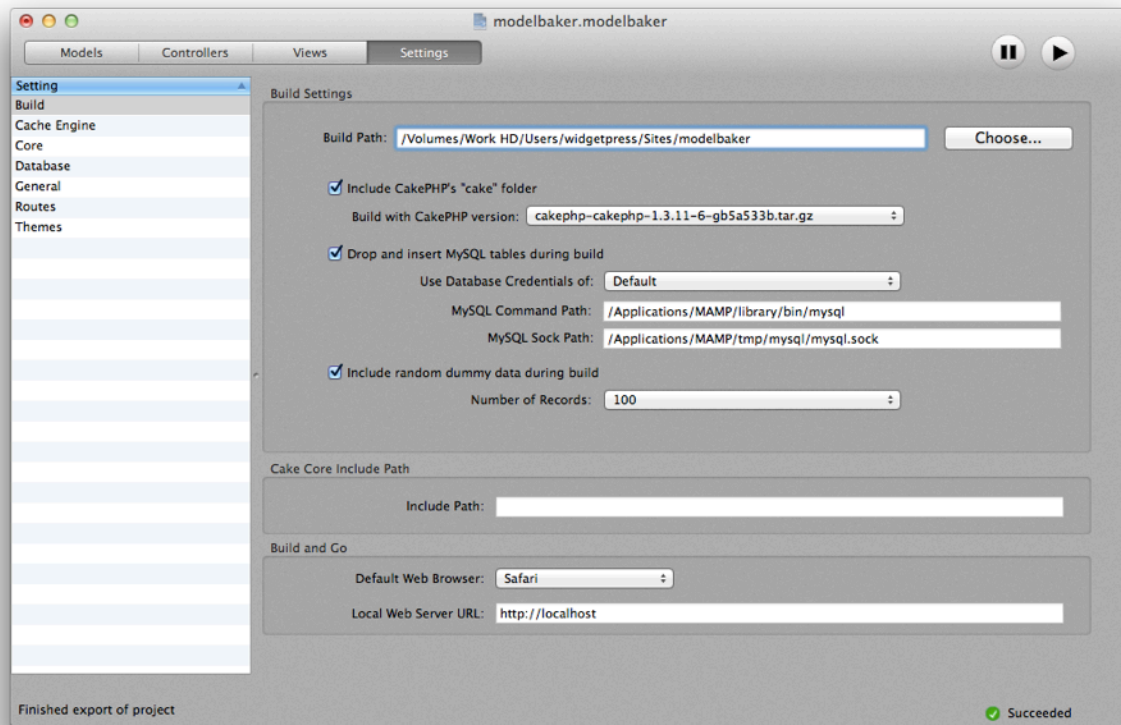


ModelBaker generates all of your views as template files. These files can be edited in any text editor. Template files have the “.CTP” (CakePHP Template) extension and contain all the presentation logic needed to render an XHTML view to your audience.

View files are stored in the folder “/app/views/”, inside subfolders named after the controller that uses the files, and named after the view it corresponds to. For example, The ‘view()’ action of the ‘Products’ controller would be found in “/app/views/products/view.ctp”.

The Settings Tab (⌘4)

The Settings Tab contains the configurations for your project. There are twelve main categories that provide access to an application's functionality and configurations. The two settings required in order for you to build a successful web app are 'Build' and 'Database'.

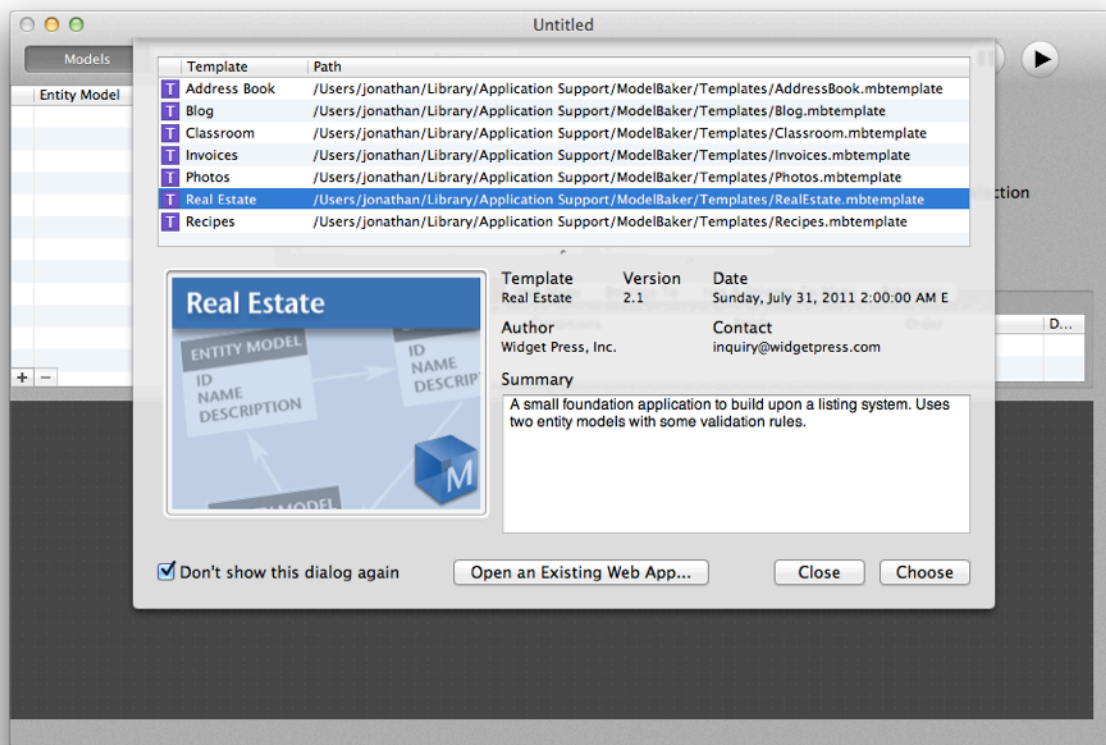


CREATING A WEB APPLICATION

This chapter outlines the basic steps for completing a professional-quality web application.

Step 1: Starting with New Project or Template

To create a new ModelBaker project simply select File - > New Project (⌘N) and an empty project window will appear.



ModelBaker provides templates to help quickly build a foundation for your web application. Each templates includes a simple skeleton of entity models with predefined attributes and associations. ModelBaker comes with a variety of templates to suit different kinds of business needs.

STARTING A NEW PROJECT

Whenever you create a new ModelBaker project, you can select a template in the ‘Template Chooser’. You can easily create new ModelBaker templates based upon ModelBaker projects you have created.

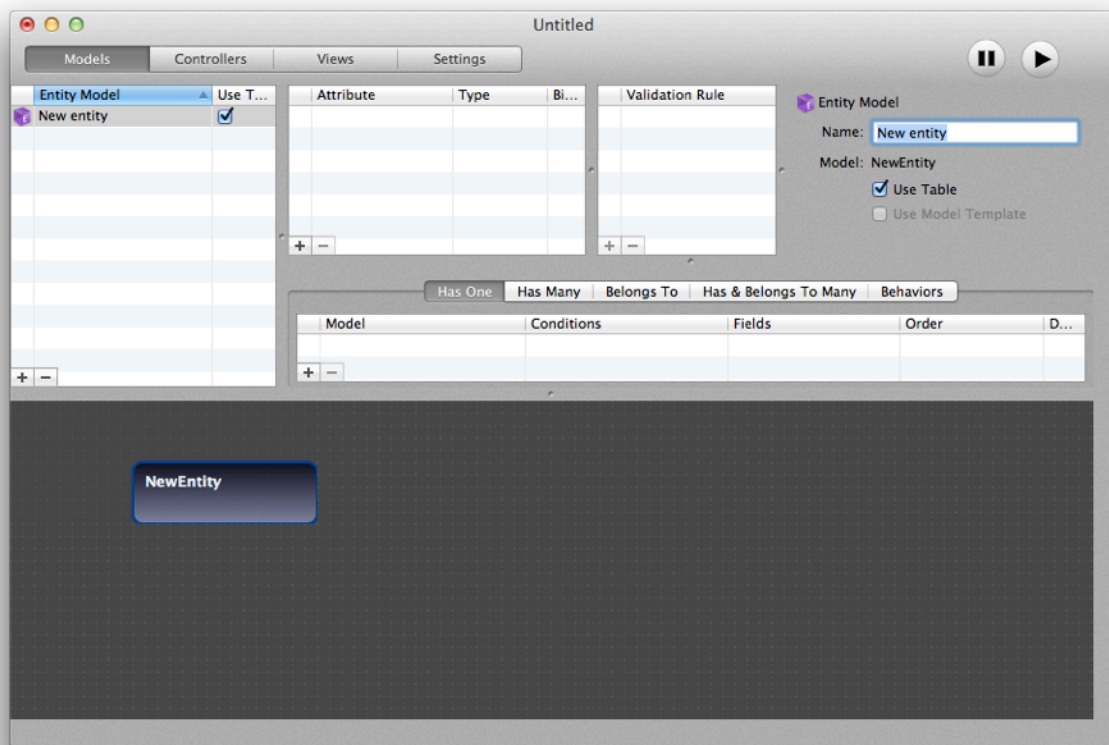
TO START A NEW PROJECT:

1. Double-click the ModelBaker icon and select a template.
2. Click ‘Choose’.

Whichever template you choose, you will probably want to add additional entity models, attributes, and validation rules for those attributes. You do not have to select a template, you can simply click ‘Close’ to ignore the Template Chooser and a new empty project window will be available.

Step 2: Creating an Entity Model

You can add new entity models to your project as needed from the Models Tab (⌘1).



To add an entity model, do the following:

- Click the 'New' (+) button at the lower left of the entity model table. You can also choose 'Design ⇌ Entity Model ⇌ Add Entity Model' (^M) to insert a new entity model.
- Rename 'New Entity' to a single noun name. For example, if you want to have a customers interact with your web application, a good choice for a new entity model name will be 'Customer'. Or if you want to have a web application that will store different types of perennials, an entity model name might be 'Flowering Perennial'.

Remember not to create plural model names, like 'Customers' or 'Perennials', this will create complications when you build your project. ModelBaker will automatically create a Controller for you with Actions. It will also create the corresponding Views as well. Remember that ModelBaker uses the CakePHP framework which uses convention over configuration.

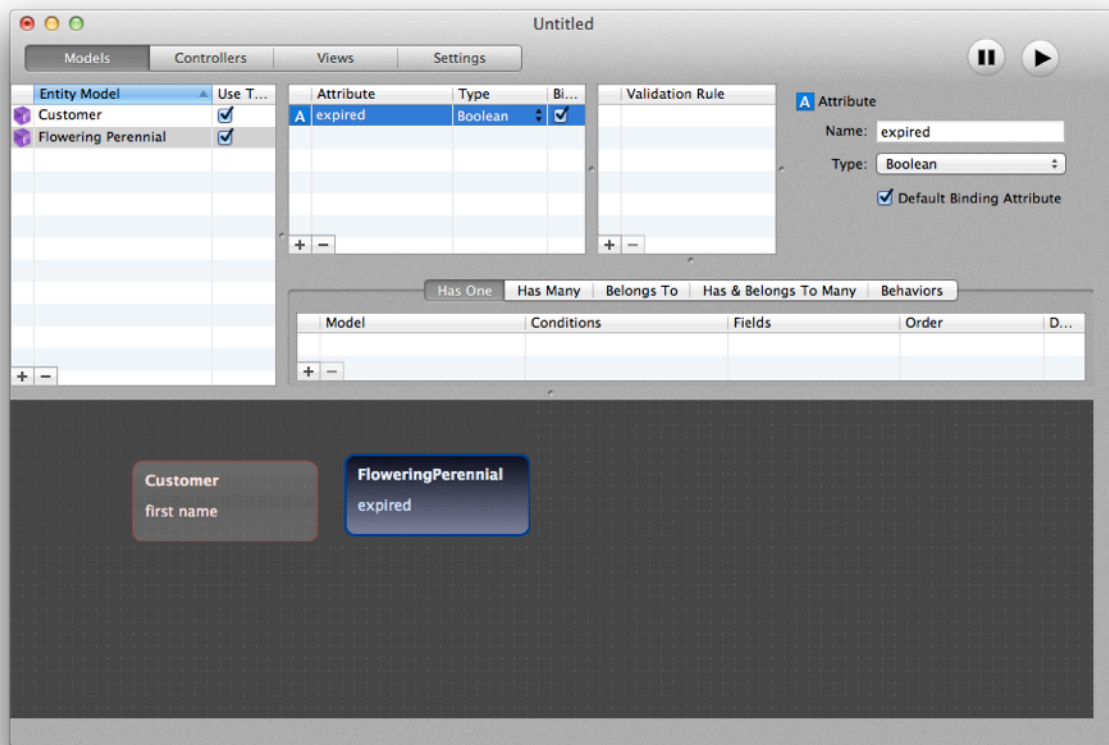
Important: Save your work often by choosing File > Save. For more details about saving ModelBaker projects, see ["Step 5: Saving Your Project"](#).

Step 3: Adding Attributes

Add attributes to your entity model as you need them to store different types of data.

To add an attribute to an entity model, do the following:

- Select the entity model needing an attribute from the Entity Model table.
- Click the 'New' (+) button at the lower left of the attribute table. You can also choose 'Design ⇌ Entity Model ⇌ Add Attribute' (^T) to insert a new attribute.
- Rename 'New Attribute' to a single noun name. For example, if you have a 'Customer' model, a very likely attribute you might want to store is a 'First Name'. If you are having your '**Great Perennial Application!**' online, a good attribute might be 'Expired' for the 'Flowering Perennial' model to indicate whether this type of perennial has expired or not.



- Change the attribute type if needed. For both the above examples, the default ‘String’ would work just fine for the ‘Customer/First Name’ attribute. A ‘Boolean’ would be the choice for the ‘Flowering Perennial/Expired’ attribute.

Step 4: Adding Validation Rules

Add validation to your attributes when you need constraints around that saving models.

To add a validation rule to an attribute, do the following:

- Select the attribute that is needing a validation rule from the Attribute table.
- Click the ‘New’ (+) button at the lower left of the validation table. You can also choose ‘Design ⇄ Entity Model ⇄ Add Entity Model’ ⇄ Add Validation Rule’ (^R) to insert a new validation rule.
- By default, a ‘User Defined’ rule is selected for your attribute. ModelBaker included several redefined validation rules including: Alphanumeric, Date, Credit Card, Phone, and

SSN (Social Security Number). If want an attribute to be required, click the *'Required'* checkbox. Choose if you want to apply this rule to *'New Entries'*, *'Existing Entries'*, or *'Both'*. Change the *'Message'* if you want a custom message for your users if the validation fails.

Step 5: Saving Your Project

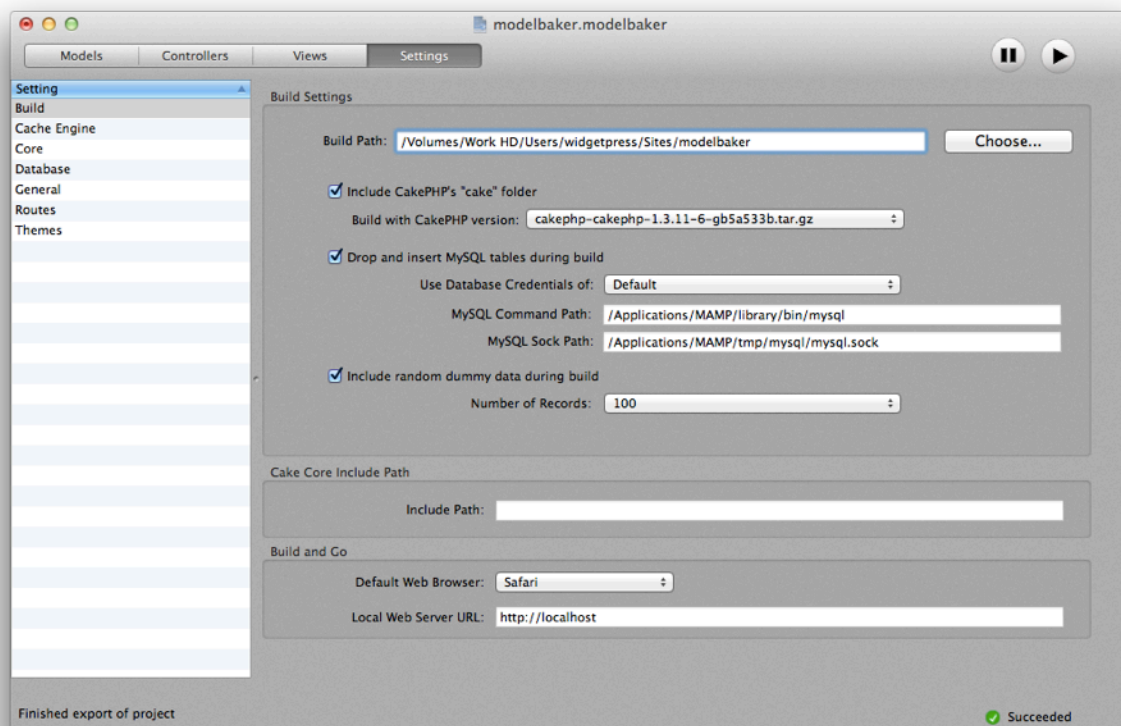
Saving your project frequently is typically the best practice when developing a web application.

To save your project, do the following:

- Select *'File ⇨ Save'* (⌘S), and choose the save path and file name of your ModelBaker project file.

Step 6: Adding a Build Path

Choose the location where ModelBaker will build your web application. This path should also be the web serving directory on your Mac.



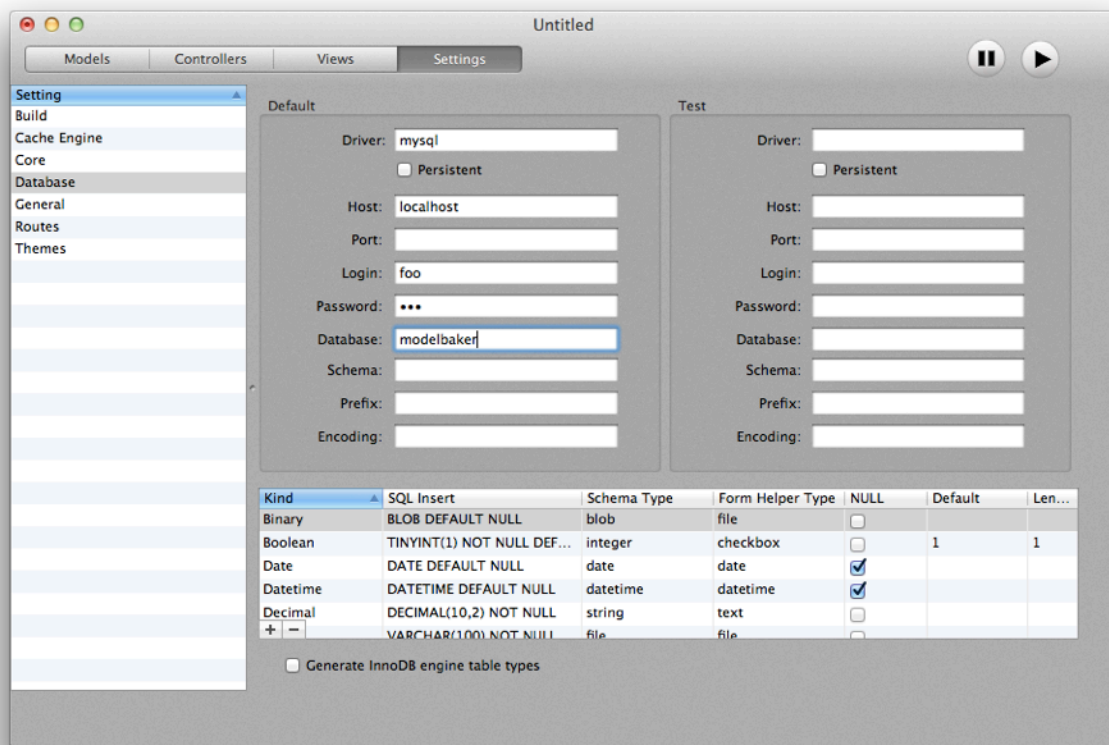
To add your Build Path, do the following:

- Select ‘Settings’ (⌘4) and choose the ‘Build’ setting from the list under ‘Setting’.
- Press the ‘Choose’ button next to the ‘Build Path:’ textfield and select the folder where you want ModelBaker to build your project.

Important: If you are using the free version of MAMP, this directory is “~/Applications/MAMP/btdocs”. If you are using XAMPP, this directory is “~/Applications/XAMPP/btdocs”.

Step 7: Adding Database Credentials

Your web application is backed by a MySQL database. To perform any testing locally, ModelBaker needs to talk to the same database when it builds and configures your web application.



To add the database credentials, do the following:

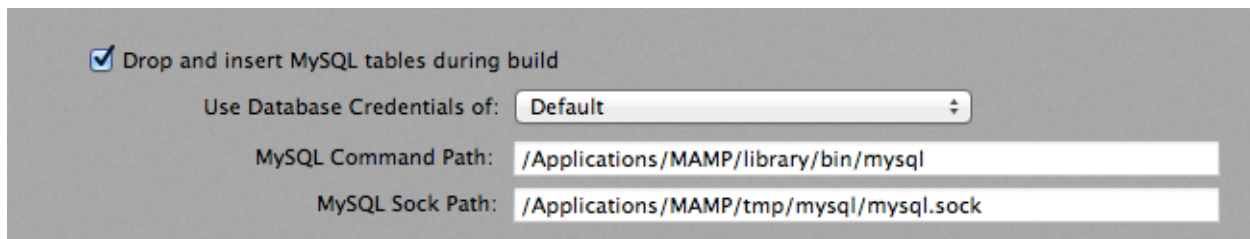
- Select ‘Settings’ (⌘4) and choose the ‘Database’ setting from the list under ‘Setting’.
- Enter the login user name of the MySQL database in the ‘Login:’ field.
- Enter the password of the MySQL database in the ‘Password:’ field.

- Enter the name of the MySQL database in the ‘Database:’.

Important: *ModelBaker will not create the database for you. You will need to create the MySQL database with the user name, password, and database name before building your project.*

Step 8: Dropping & Inserting Tables

ModelBaker needs to insert your project’s schema into the database during each build. You will need to know two very important path locations: one for the MySQL command and one for the active MySQL socket.



The screenshot shows a settings window with the following configuration:

- ☒ Drop and insert MySQL tables during build
- Use Database Credentials of: Default
- MySQL Command Path: /Applications/MAMP/library/bin/mysql
- MySQL Socket Path: /Applications/MAMP/tmp/mysql/mysql.sock

To allow ModelBaker to insert new tables and columns, do the following:

- Select ‘Settings’ (⌘4) and choose the ‘Build’ setting from the list under ‘Setting’.
- Enable the checkbox for ‘Drop and insert MySQL tables during build’.
- Enter the full MySQL command path location in the ‘MySQL Command Path:’ textfield. For MAMP, the path is “-/Applications/MAMP/Library/bin/mysql” and for XAMPP it is “-/Applications/XAMPP/xamppfiles/bin/mysql”. For any other configuration, please refer to the included documentation for the full path.
- Enter the full MySQL socket path location in the ‘MySQL Socket Path:’ textfield. For MAMP it is “-/Applications/MAMP/tmp/mysql/mysql.sock” and for XAMPP it is “-/Applications/XAMPP/xamppfiles/temp/mysql.sock”.

Important: *To successfully build your web application with ModelBaker you need to have the MySQL server running and a database created with the name matching the ‘Database:’ textfield under the ‘Database’ setting. This database needs to have the ‘Login:’ user name with the ‘Password:’ already set.*

Step 9: Building Your Web App

At this step, the last thing to do is save your project and build it. After building your project, you should be able to see the result in a web browser and view the database with a database query tool.

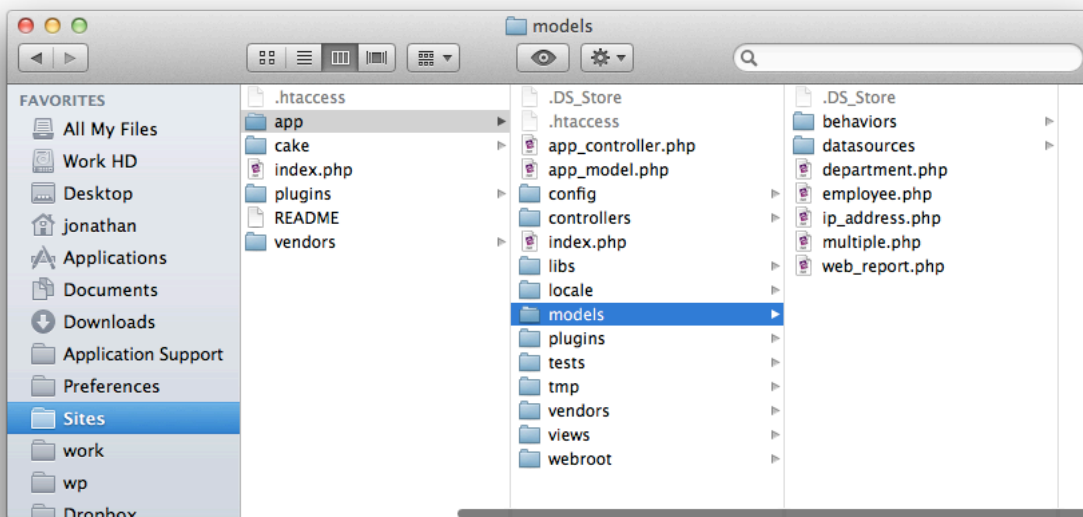
To build your web application, do one the following:

- Select *'File ⇨ Save'* (⌘S), and choose the save path and file name of your ModelBaker project file (if you have not already done so).
- Select the *'Build'* (⌘B) button from the toolbar.

or

- *'Save'* your project (⌘S) and select *'Build and Go'* (⌘⇧G) from the toolbar, if you have entered a URL in the *'Local Web Server URL:'* textfield under the *'Build and Go'* section in the *'Build'* setting.

After you've built your project, you should also see the directory and file structure at your build path:



ENTITY MODEL BASICS

This chapter will explore the foundation of ModelBaker and why we build from the ground up.

ModelBaker's takes a "ground up" approach to development; building models with agility and quick prototyping. Much like a home's foundation, ModelBaker is targeted at developing the foundation of your web project. You build entity models that will have attributes to hold our data, relationships to connect entities together, and validation rules to ensure data consistency, just like a house would have a floor foundation, walls, electrical wiring, and plumbing held together between the flooring and walls.

ModelBaker's 'Entity Model' foundation is not exactly the same as the database world's 'entity model'. Borrowing database terminology and applying common object oriented terms, ModelBaker provides a collage for constructing an entity model. For example, you do not need to worry about MySQL's 'id' or 'foreign_key_id' when you construct your project, simply think about the task at hand and add attributes, relationships, and validation rules to your entities.

ModelBaker takes care of building your MySQL database and more.

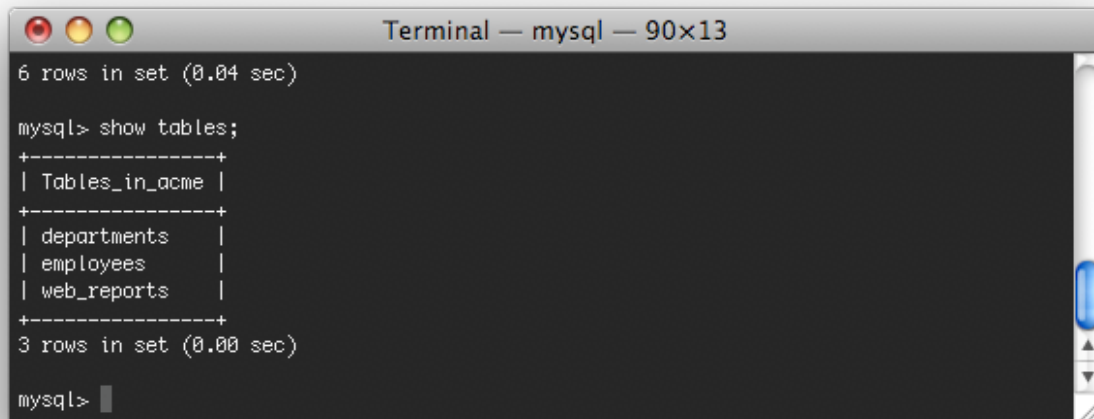
Adding an Entity Model

When you add an entity model within ModelBaker you are really creating a place holder object for the model definition. Only when you build your project are a MySQL database table and PHP model class file created.

An entity model name should be both a noun and singular. Two or more words in an entity model name are acceptable (i.e. 'Web Report' or 'Visitor Trend'). ModelBaker will alert you if the name you have chosen is plural or violates a PHP, CakePHP, or MySQL reserved word.

After an entity is created, ModelBaker creates a corresponding controller with CRUD actions and places it under 'Controllers' in the main window. Additionally, four views ('add', 'edit', 'index', and 'view') are created and under the main window's 'Views' tab.

For example, if you create an entity model named ‘Web Report’, ModelBaker will create a PHP model class called ‘WebReport’ and the file ‘web_report.php’ after you build your project.

A screenshot of a macOS Terminal window titled "Terminal — mysql — 90x13". The window has a dark background and shows the output of a MySQL query. The first line is "6 rows in set (0.04 sec)". The second line is the command "mysql> show tables;". The output is a table with one column and three rows: "Tables_in_acme", "departments", "employees", and "web_reports". The third line is "3 rows in set (0.00 sec)". The prompt "mysql>" is visible at the bottom.

```
6 rows in set (0.04 sec)

mysql> show tables;
+-----+
| Tables_in_acme |
+-----+
| departments    |
| employees      |
| web_reports    |
+-----+
3 rows in set (0.00 sec)

mysql>
```

The ‘Web Report’ model class will also be backed by the corresponding MySQL database table named ‘web_reports’, a controller class named ‘WebReports’, and will have the file ‘web_reports_controller.php’.

USE TABLE

By default, ModelBaker expects to create database tables for your entity model. If you uncheck the ‘*User Table*’ checkbox, ModelBaker updates your model class to not use a MySQL table.

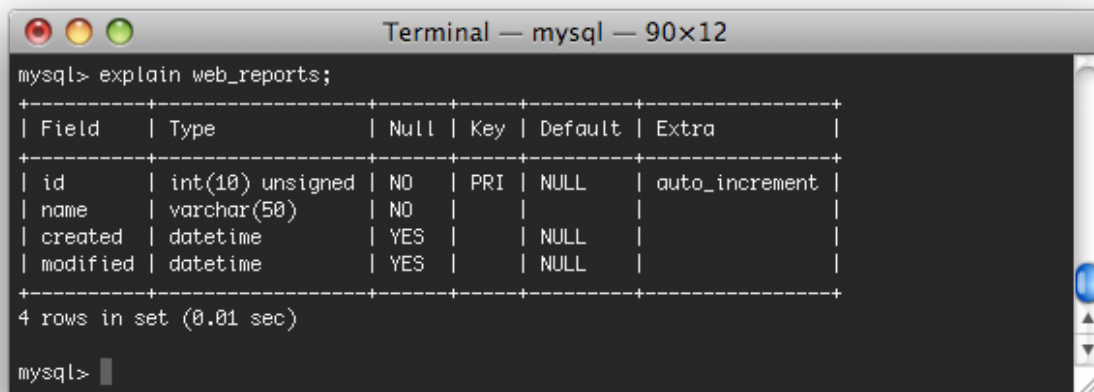
USE MODEL TEMPLATE

ModelBaker provides the ability to use an existing model class file to replace the generated model file during the build process. To replace the generated model with your own model, simply add your model class file into the “-/Application Support/ModelBaker/Model Templates/” folder. The ‘*Use Model Template*’ checkbox will become enabled and you can select the desired template.

Adding an Attribute with Type

Attributes are the properties of your entity model and map directly to the model’s MySQL table column. For example, the ‘WebReport’ PHP class (the entity model of ‘Web Report’), would have a MySQL table named ‘web_reports’. If you add a ‘name’ attribute with the type

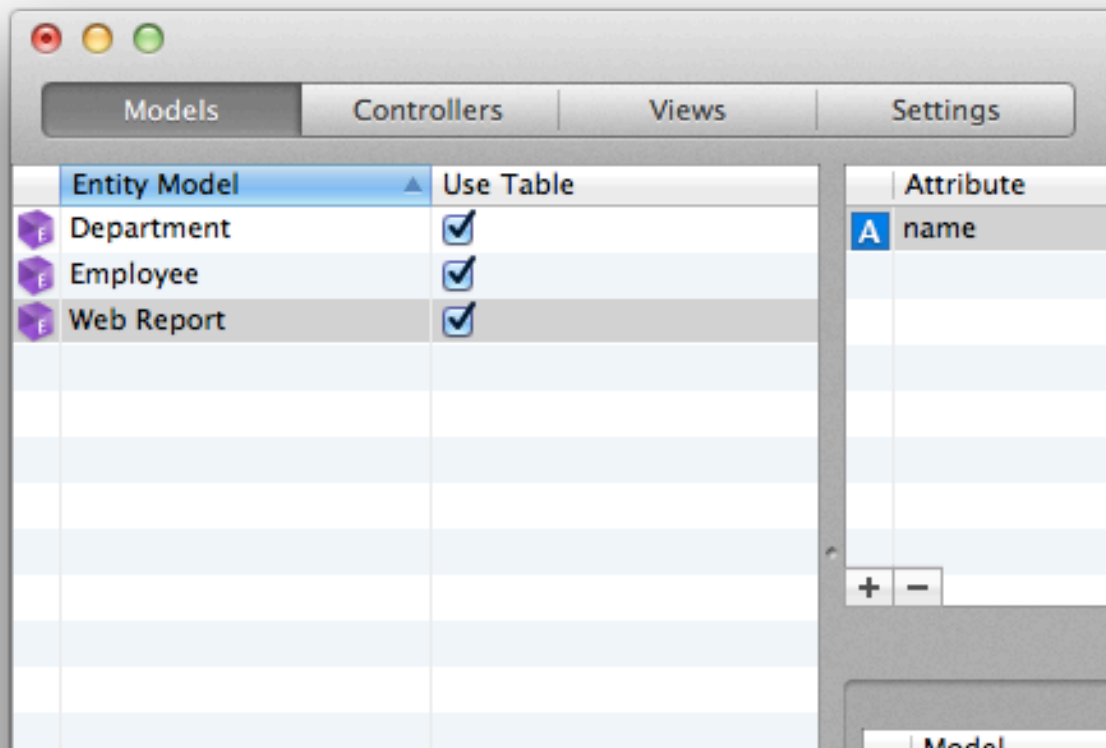
'String' to the 'Web Report' entity model, ModelBaker will generate an SQL statement to insert the appropriate column for that attribute.



```
Terminal — mysql — 90x12
mysql> explain web_reports;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id     | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name   | varchar(50)      | NO   |     |          |                |
| created | datetime         | YES  |     | NULL    |                |
| modified | datetime         | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)

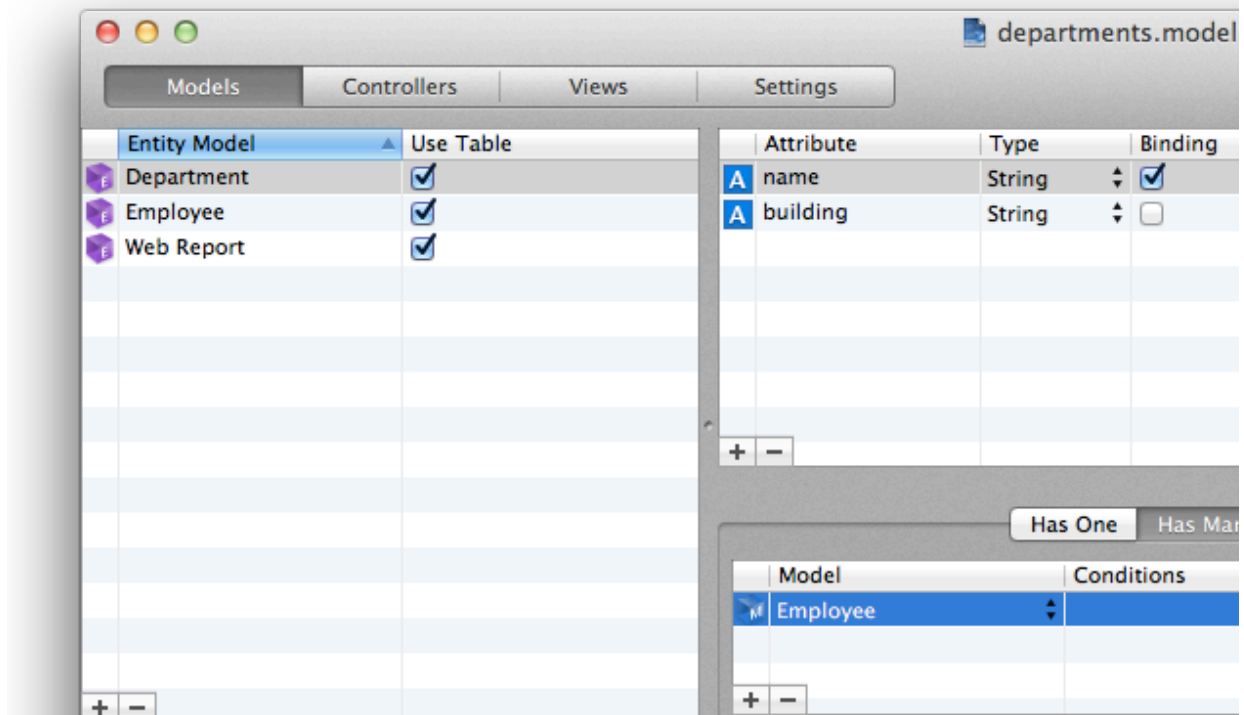
mysql>
```

Notice the Terminal screen above; the 'id', 'name', 'created', and 'modified' attributes are contained within the 'web_reports' table. ModelBaker automatically adds these attributes by default for each table it builds. There is no need to create either these attributes or any foreign key id. ModelBaker you focus on the design of the entity model while it builds the relationship keys and SQL for you.



DEFAULT BINDING ATTRIBUTE

ModelBaker uses the first attribute you add to an entity model as the default ‘Binding’ attribute for that model. Each model can have only one attribute as the ‘binding’ attribute for that model.



This means that a ‘*Belongs To*’ relationship will have a drop-down menu appear next to this attribute in the user interface. For example, take a ‘Department’ model with the attribute ‘name’; add a ‘*Belongs To*’ relationship from the ‘Employee’ model to the ‘Department’ model and when you build your project, the add/edit views would generate a drop-down menu bound to the ‘Department’ model’s ‘name’ attribute. See the screen shot below:

My Web Application

Departments

Employees

Web Reports

Actions

Add Department

List Of Departments

Add Employee

List Of Employees

Add Web Report

List Of Web Reports

Add Employee

Department

Human Resources

First Name

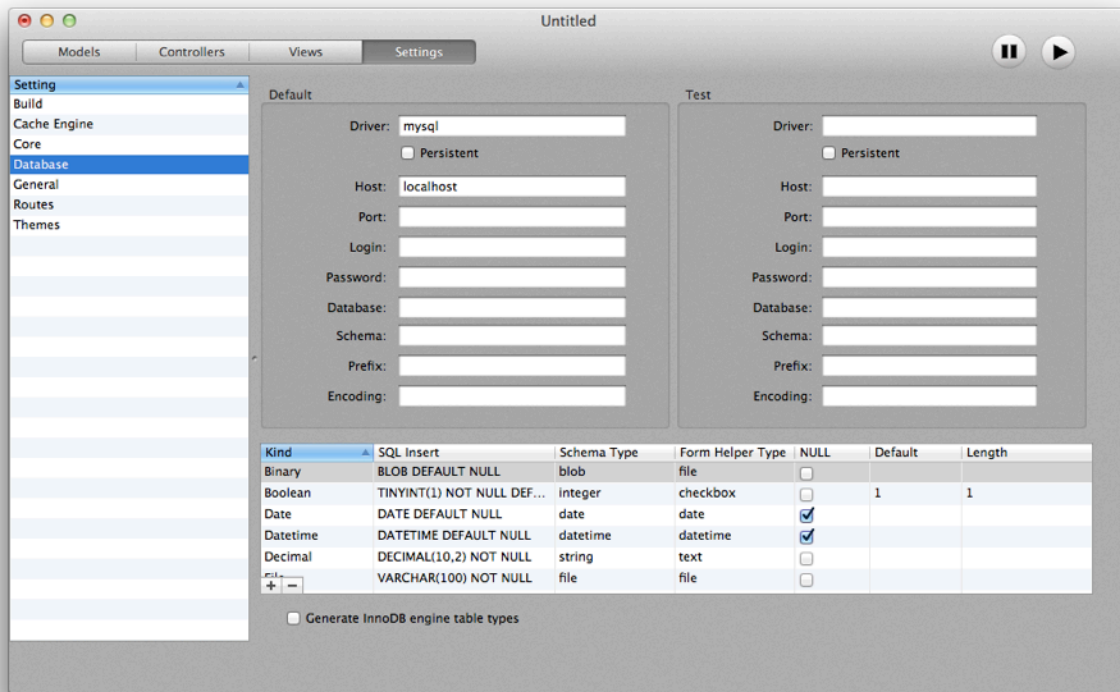
Last Name

Submit

The *'Human Resources'* value listed next to *'Department'* in the image above is one of many records from the column (attribute) *'name'* in the *'departments'* table. The *'name'* attribute is the default binding attribute.

Adding Custom Database Attribute Types

ModelBaker comes with a default list of database attribute types that can be applied to your entity model attributes. These default database types can only be modified and added to each project individually.



If you need to add a custom database attribute type to your project,

- Select 'Settings' (⌘4) and choose the 'Database' setting from the list under 'Setting'.
- Click the Add button (+) located near the bottom of the window.
- In the 'Kind' column, enter the name of the Database Type (i.e. 'String Medium'). This name will appear in the drop-down menu in the 'Attribute' table on the 'Models' screen.
- In the 'SQL Insert' column, enter the raw SQL type. For this example, the type is 'VARCHAR(25) NOT NULL'.
- In the 'Schema Type' column, enter the SQL schema type for this attribute type, in this case the type is 'string'.
- Next, change the 'Form Helper Type' to 'text'.

It is also possible to set whether the attribute can be NULL, its default setting, and a maximum length.

The new database attribute type 'String Medium' will now appear in the 'Type' drop-down menu under 'Models' (⌘1).

Settings						
	Attribute	Type	Binding			Validation
A	name	String Medium	<input checked="" type="checkbox"/>			

Adding Validation Rules

ModelBaker's validation rules are based on the CakePHP framework's validation rule mechanism. ModelBaker provides 23 default validation rules you can use on attributes. If validation rule fails, then no data will be saved to the database. In the event of a rule failure, it is possible to display a custom error message to the user explain what has occurred.

All validation rules have a minimum of four options that can be applied to the selected attribute. Each validation rule can use one or more of the following options: .

REQUIRED

When the *'Required'* checkbox is selected, ModelBaker generates the code necessary to make sure that when a user saves a form, this attribute must have a valid value. If validation fails, then the comment in the *'Message:'* textfield will be displayed to the user.

ALLOW BLANK

When *'Allow Blank'* is selected, ModelBaker will allow the user to submit the form with no data or empty data for the this field. When the checkbox is unselected, the user must submit real, nonempty data for this field. If validation fails, then the comment in the *'Message:'* textfield will be displayed to the user.

ON

This drop-down menu will determine when the validation rules for this attribute are applied. If *'Create'* is selected, this rule only applies when the user is adding a new record. If *'Update'* is selected, this rule only applies when a record is being edited by the user. Selecting *'Both'*, applies the rule when a record is created or updated.

MESSAGE

This is the text message that will be displayed when a validation rule fails.

The default validation rules are listed below:

ALPHANUMERIC

Checks a string to make sure it contains only letters or integers.

BETWEEN

Checks to see if the input is within a specified range. The '*Min:*' and '*Max:*' values should be integers.

BLANK

Checks to see if a field is left blank or contains only whitespace characters (Space, Tab, Carriage Return, or New Line). Keep in mind, the '*Allow Blank*' option (described in the previous section) must be selected if you want this rule to take into effort. Leaving the '*Allow Blank*' unselected for this rule will cause the rule to be ignored.

BOOLEAN

Boolean validation, determines if value passed is a boolean integer or true/false.

COMPARISON

Checks the value of the attribute and compares it to an integer value you provide. Comparisons include: '*Is Greater*', '*Greater or Equal*', '*Less or Equal*', '*Is Less*', '*Equal To*', and '*Not Equal*'.

CREDIT CARD

Checks the value in the textfield is a valid credit card number. Selecting '*All*' will check all 13 credit card types, while selecting '*Fast*' will check the major credit card numbering formats. If '*Luhn Algorithm*' is selected, it will use this algorithm for validation. For more information on this algorithm, see this [Wikipedia article](#).

You can choose to write a regular expression in the '*Or Regex:*' field and this expression will be used instead of the other fields.

CUSTOM

Checks if the attribute value passes the regular expression contained in the '*Regex:*' field.

For example, if you want to make sure only US state codes are only allowed, you can add the following regular expression in the '*Regex*' field:

```
/\\b(?:A[KLRZ]|C[AOT]|D[CE]|FL|GA|HI|I[ADLN]|K[SY]|LA|M[ADEINOST]|N[CDEHJMVY]|O[HKR]|PA|RI|S[CD]|T[NX]|UT|V[AT]|W[AIVY])\\b/
```

If you need to match a word where the first letter was ‘b’ and the last letter was ‘g’, the following regex would suffice:

```
/b[aoiu]g/
```

D A T E

Checks for a valid date string.

You can choose to write a regular expression in the ‘*Or Regex:*’ field and this expression will be used instead of the other fields.

For example, if you want to match the following date ranges with dates of 1/1/00 through 31/12/99 and 01/01/1900 through 31/12/2099, then the following regex would work:

```
%^\\b(0?[1-9]|12)[0-9]|3[01])(- /.)(0?[1-9]|1[012])(- /.)(19|20)?[0-9]{2}\\b$%
```

D E C I M A L

Checks if the attribute value contains the correct number of places from the decimal point based on the ‘*Places:*’ field. The ‘*Places:*’ field takes an integer value.

You can choose to write a regular expression in the ‘*Or Regex:*’ textfield and this expression will be used instead of the other fields.

For example, if you want to check the C-Style hex decimal (0xCAFE and 0X042 style) for a field, the following regex would suffice:

```
^\\b0[xX][0-9a-fA-F]+\\b^
```

E M A I L

Checks to see if the field contains a valid email address. If the checkbox for ‘*Check availability of host*’ is selected, ModelBaker will check if the email host is currently active and available.

You can choose to write a regular expression in the ‘*Regex:*’ field and this expression will be used instead of the other fields.

The following regex will check to make sure the attribute contains a valid URL:

`^[A-Z0-9._%-]+@[A-Z0-9.-]+\.(?:[A-Z]{2}|com|org|net|biz|info|name|aero|biz|in
fo|jobs|museum|name)$`

EXACTLY EQUAL

Checks if the value in the field exactly matches the value contained in the ‘*To:*’ field.

EXTENSION

Checks the field against all the extensions provided in the ‘*File Extension*’ table. When adding extensions to validate, use dot syntax. For example, “.jpg” or “.xls”.

FILE

Checks if the field validates against the maximum file upload size (‘*Max Size:*’ in MegaBytes) and the allowed upload file types listed in the ‘*Allowed File Upload Types:*’ table. If the file size is too large or does not match an upload file type, the user will receive the ‘*Message:*’ value.

The location specified in the ‘*Path:*’ field will place these uploaded files in the “/app/webroot/files/<PATH>” folder. When you build your project, ModelBaker will create the directory using the path “/app/webroot/files/” so your users can upload to this directory behind the scenes.

To add a file type to the ‘*Allowed File Upload Types:*’ list, you will need to add the known MIME type for the file. For example, for a Portable Network Graphics file, the correct MIME type is ‘image/png’. A complete listing can be found at [‘www.iana.org/assignments/media-types/’](http://www.iana.org/assignments/media-types/).

IPV4 ADDRESS

Checks the field for a valid IP input value provided by the user, such as ‘10.77.188.166’. You can also check for IPv6, such as ‘2001:0db8::1428:57ab’ and you can even check both.

IN LIST

Checks if a value is in a given list.

IS UNIQUE

Checks if a value is in a given list.

MAX LENGTH

Check to see if the input value stays below the length requirement shown in the '*Maximum:*' field.

MIN LENGTH

Conversely, this will check to see if the input value stays above the length requirement listed in the '*Minimum:*' field.

MONEY

Checks if field passes with a monetary amount. It also checks the position of the currency symbol whether it is on the left or the right in accordance with the '*Symbol:*' pulldown menu.

MULTIPLE

This rule is used to validate multiple selection with the regular expression listed in the '*Regex:*' field.

NOT EMPTY

Checks that a string contains something other than whitespace.

NUMERIC

Checks if the field contains a numeric value.

PHONE

Checks if the field contains a phone number that conforms to the standard of the country selected in the '*Country:*' pulldown menu.

You can choose to write a regular expression in the '*Or Regex:*' field and this expression will be used instead of the other fields.

POSTAL

Checks if the field contains a postal code that conforms to the standard of the country selected in the '*Country:*' pulldown menu.

You can choose to write a regular expression in the '*Or Regex:*' field and this expression will be used instead of the other fields.

R A N G E

Validates that a number is in a specified range. If the Lower and Upper values are not set, it will return true if the checking value is a legal finite on running platform.

S S N

Checks if the field contains a Social Security Number (SSN) that conforms to the standard of the country selected in the '*Country*' pulldown menu.

You can choose to write a regular expression in the '*Or Regex:*' field and this expression will be used instead the other fields.

U R L

Checks if the value of the field is a valid URL address.

U S E R D E F I N E D

This rule allows you to add custom validation for the data entered by the user.. The '*Object:*' field is the class name that holds the validation method. The '*Method:*' is the validation method to execute. The '*Args:*' field holds the arguments to send to the chosen method.

CREATING FILE UPLOADS

In this chapter we will discuss the steps needed to build a file upload system for your web application. We will collect the uploaded files on the web server side in a specified upload directory.

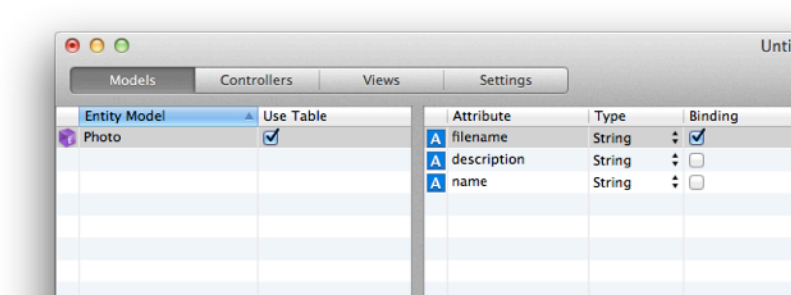
Uploading files from the web browser is a common activity these days for some web applications, such as Flickr or Facebook. ModelBaker provides the base methods for your project to have a working, fully functioning upload system.

Unfortunately, not all browsers will allow standard file uploads. Your generated uploading system works with all Class 'A' web browsers with a few exceptions: Mobile Safari on the iPhone and iPod Touch and the Google Android Browser on Google's Android platform. As of the writing of this manual, both Apple and Google have determined to not allow mobile browser based file uploads. Hopefully, this policy will change soon.

If you need further details about the 'File' validation rule, please refer to the ['File'](#) listing in the ['Adding Validation Rules'](#) chapter in this guide. The following steps illustrate the process for enabling a file upload system within your project.

Step 1. Create an Entity Model and Attribute 'filename'

You can add new entity models to your project as needed from the 'Models' tab (§ 1). The key for the upload system is having an attribute with the special name of 'filename'.



To add an entity model and attribute, do the following:

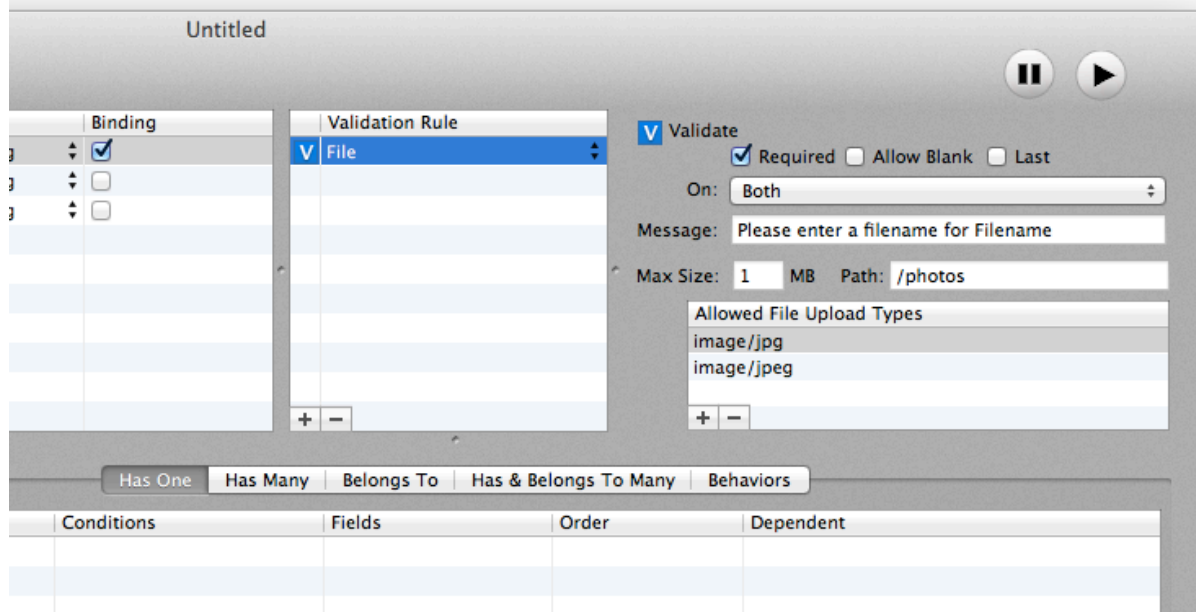
- Click the 'New' (+) button at the lower left of the 'Entity Model' table. You can also choose 'Design ⇄ Entity Model ⇄ Add Entity Model' (^M) to insert a new entity model.
- Rename this entity model to 'Photo'.

- With the *'Photo'* entity model selected, click the *'New'* (+) button at the lower left of the *'Attribute'* table. You can also choose *'Design ⇄ Entity Model ⇄ Add Attribute'* (^T) to insert a new attribute.
- Rename this attribute to *'filename'* and make sure the *'Type'* is set to *'file'*.

Important: Make sure the *'filename'* attribute's *'Type'* is set to *'file'*. If this is not set correctly, the upload system will not be created properly!

Step 2. Adding Validation Rules for *'filename'*

The following steps add the final parameters needed for the upload system.



To add the *'file'* validation rule with parameters, do the following:

- Select the *'filename'* attribute by clicking on its row in the *'Attribute'* table.
- Click the *'New'* (+) button at the lower left of the *'Attribute'* table. You can also choose *'Design ⇄ Entity Model ⇄ Add Attribute'* (^T) to insert a new attribute.
- Click the *'New'* (+) button at the lower left of the *'Validation Rule'* table. You can also choose *'Design ⇄ Entity Model ⇄ Add Validation Rule'* (^R) to insert a new validation rule. Change the validation rule type to *'file'*.

- In the 'Max Size:' input field, set the desired maximum size allowed for uploads in Mega-Bytes (MB).
- In the 'Path:' input field, type the directory path where the uploaded files will be stored.
- In the 'Allowed File Upload Types:' table, add the MIME file types to allow by clicking the 'New' (+) button at the lower left of the table.

A complete listing of the standard MIME file types can be found at:

www.iana.org/assignments/media-types/.

Note that if two users upload the same file with the exact same name, i.e. both upload "spreadsheet.xls", then the second spreadsheet file will be appended with a sequential number. This will prevent the second user from overwriting the first user's spreadsheet.

BASIC HTTP AUTHENTICATION

In this chapter we will discuss basic HTTP authentication using ModelBaker.

[Wikipedia](#) defines [basic access authentication](#) as “In the context of an HTTP transaction, the basic access authentication is a method designed to allow a web browser, or other client program, to provide credentials – in the form of a user name and password – when making a request.”

Step 1: Setting up the Controller

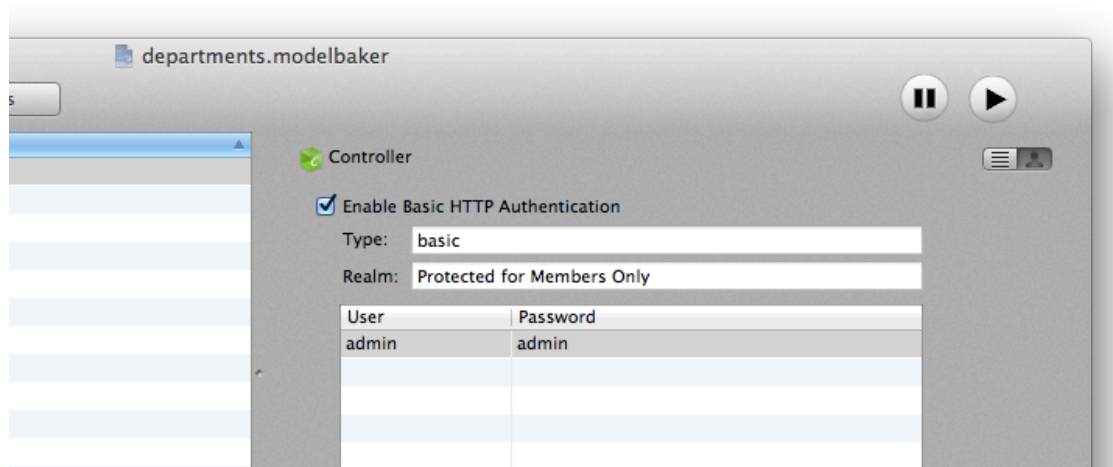
To have an action become protected, we need to tell the action's Controller.

To setup a Controller for authentication, do the following:

- In the 'Controllers' tab (⌘2), select the controller to which you would like to add Authentication.



- Select the 'Basic HTTP Authentication' user icon.
- Enable the checkbox for 'Basic HTTP Authentication'.
- Enter 'basic' in the 'Type' textfield.
- Enter a desired message (i.e. 'Protected for Members Only') in the 'Realm:' textfield.
- Click the 'New' (+) button to add a user name and password (i.e. 'foo' and 'bar').

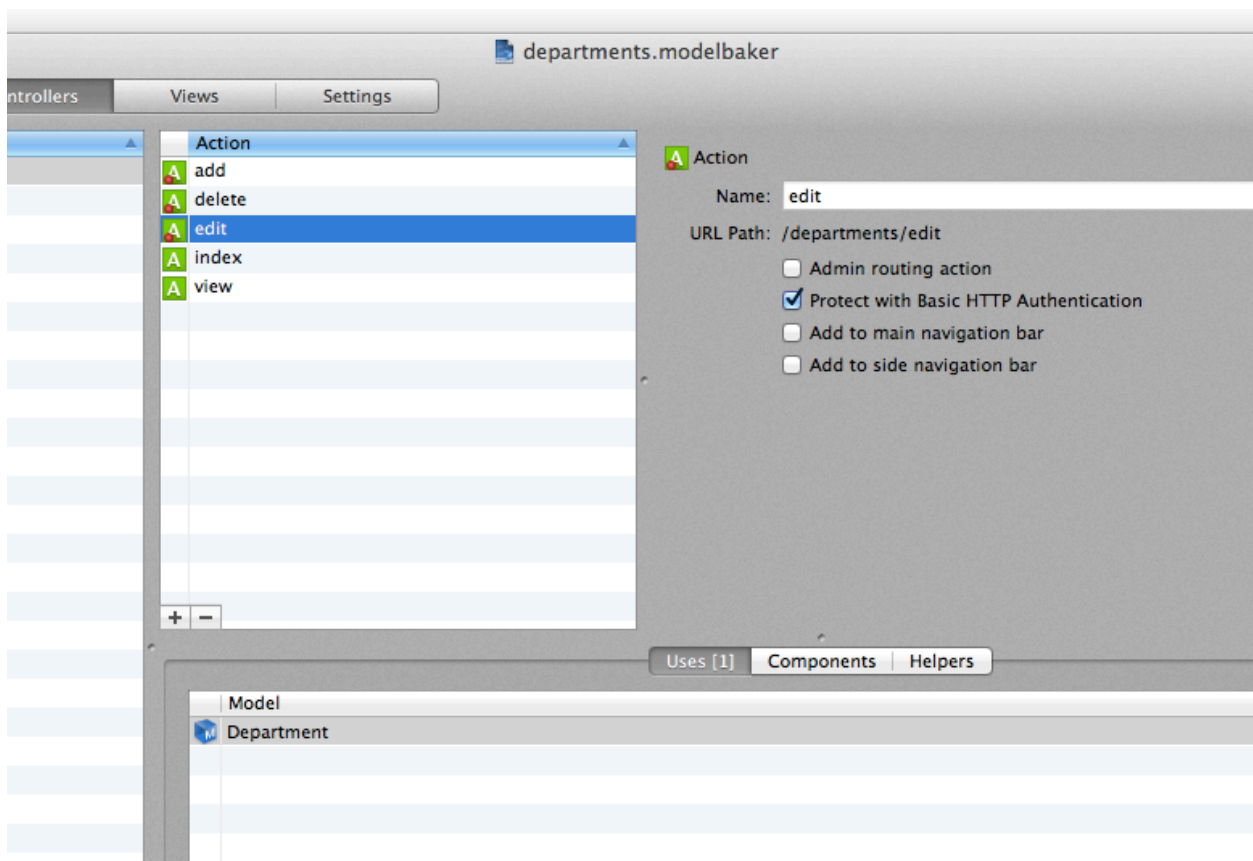


Step 2: Setting the Action for Authentication

Now that a controller is aware of HTTP authentication, we can tell it which actions will use the user names and passwords. Notice that when you select an Action to edit, the *'Protect with Basic HTTP Authentication'* is now available.

To add basic authentication for an action, do the following:

- Select the desired action to protection.
- Select the checkbox for *'Protect with Basic HTTP Authentication'* for each of the actions you wish to add HTTP Authentication to.



ADMIN ROUTING ACTIONS

This chapter describes how to create a simple admin routing action.

Sometimes we would like to move certain actions, perhaps the ‘Add’, ‘Edit’, or ‘Delete’ actions, behind an administration URL. For example, “www.example.com/authors/add” would become “www.example.com/admin/authors/add”.

Step 1: Setting up the Admin Routing Name

To have an action become protected behind an administration URL, we need to provide the Core Routing File of your generated web app with a Routing Admin name.

To setup the core routing file with a routing admin name, do the following:

- In the ‘Settings’ tab (⌘4), select ‘Core’ from the ‘Setting’ table located on the left hand side of the window.
- Locate the ‘General’ settings box.
- Enter ‘admin’ in the ‘Routing Admin:’ field.

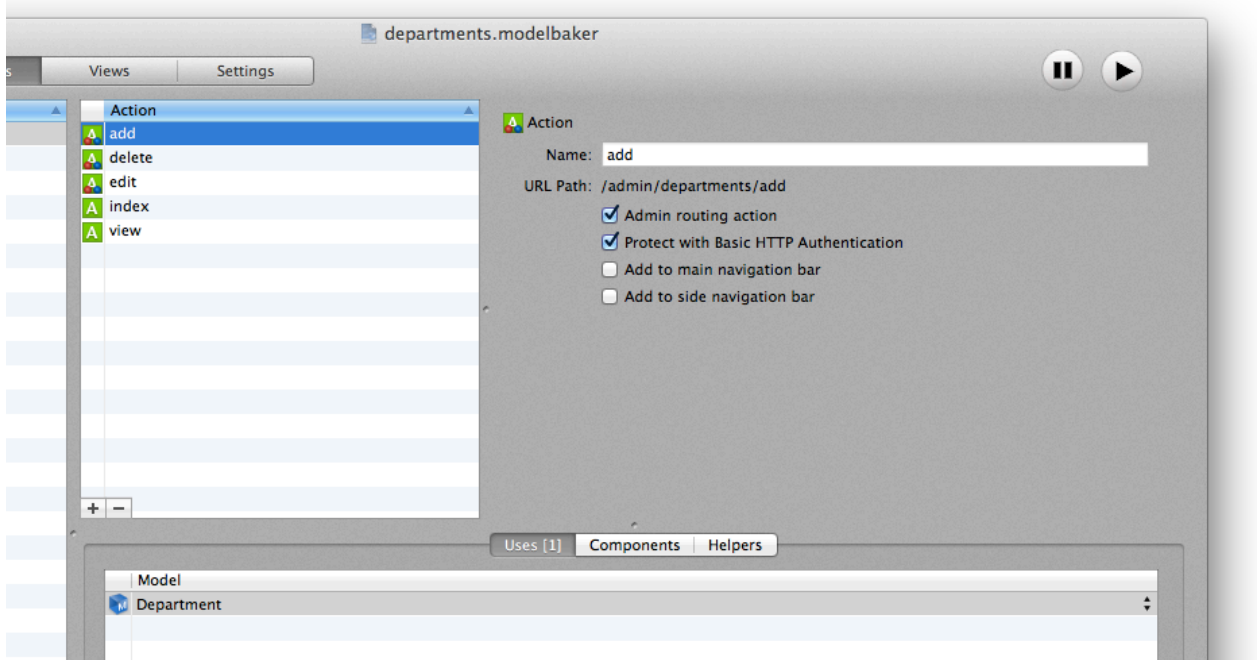
The screenshot shows the 'ModelBaker' application settings window. The 'Settings' tab is selected, and the 'General' section is active. The 'Routing Admin' field is highlighted and contains the text 'admin'. Other visible fields include 'Session Save' (php), 'Session Table', 'Session Database', 'Session Cookie' (CAKEPHP), 'Session Timeout' (120), 'Security Level' (High), 'Security Salt' (YhG93b0qyJfbfs2guVoU), 'Cipher Seed' (6859309657453542496), 'ACL Class Name' (DbAcl), 'ACL Database' (default), 'Application Encoding' (UTF-8), 'Base URL', 'Asset Filter CSS', 'Asset Filter Javascript', 'Apache mod_rewrite' (checked), 'Enable Application-wide Cache' (unchecked), 'Enable Cache Checking' (unchecked), and 'Debugging' options (0: No error messages, errors, or warnings shown. Flash messages redirect. 1: Errors and warnings shown, model caches refreshed, flash messages halted. 2: As in 1, but also with full debug messages and SQL output).

Step 2: Setting Action to the Routing Name

To have an action use the *'Routing Admin:'* URL you have just entered, you need to enable it for each action.

To setup the routing admin name to each action, do the following:

- In the *'Controllers'* tab (§2), select a controller from the *'Controller'* table located on the left hand side of the window.
- Next, select the action you would like to include the Routing Admin URL.
- The *'Admin routing action'* checkbox is now available to be selected. Select the checkbox and you will see the string shown next to the label *'URL Path:'* change to include the *'Routing Admin'* URL.



INTERNATIONALIZATION & LOCALIZATION

This chapter describes the steps you take to change your application into multiple languages.

Changing the strings in your application is simple and straight forward. ModelBaker generates a simple “.PO” file saved in the UTF-8 format that you can modify for your application. After building your project, this .PO file will be located at “/app/locale/eng/LC_MESSAGES/default.po”.

Within the “default.po” file are the string values you will change to include another language.

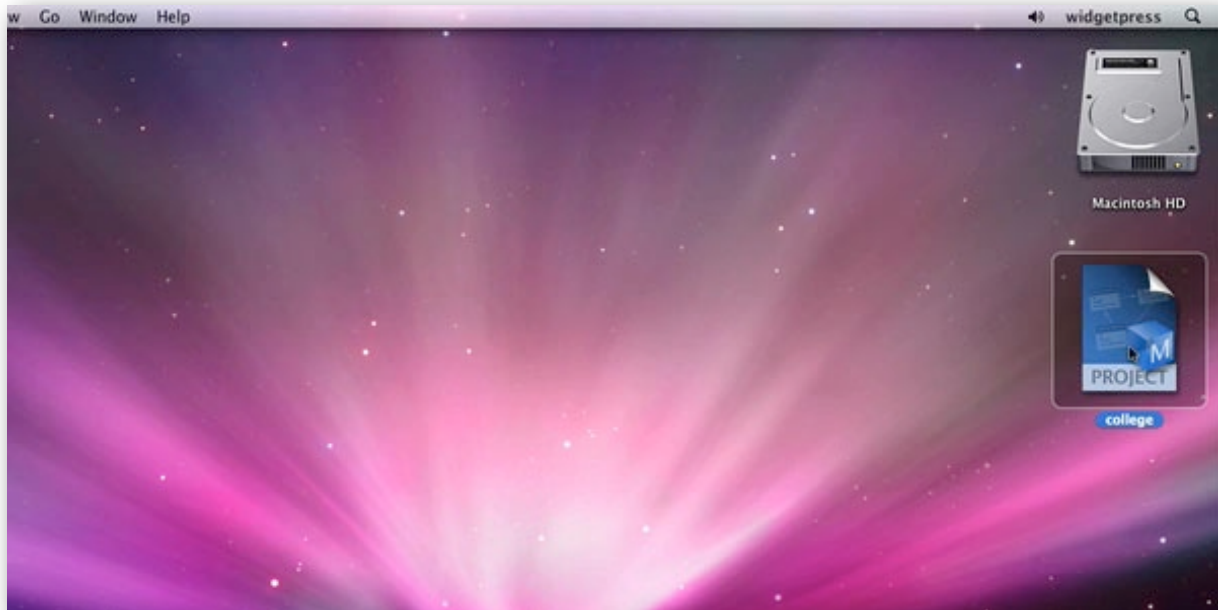
CREATING CUSTOM TEMPLATES

This chapter describes how to create a new ModelBaker template based on an existing ModelBaker project.

Note: This chapter assumes you already have created a ModelBaker project called 'college' populated with entities and attributes. Pay special attention to the naming conventions in this chapter because a template needs to adhere to specific naming conventions.

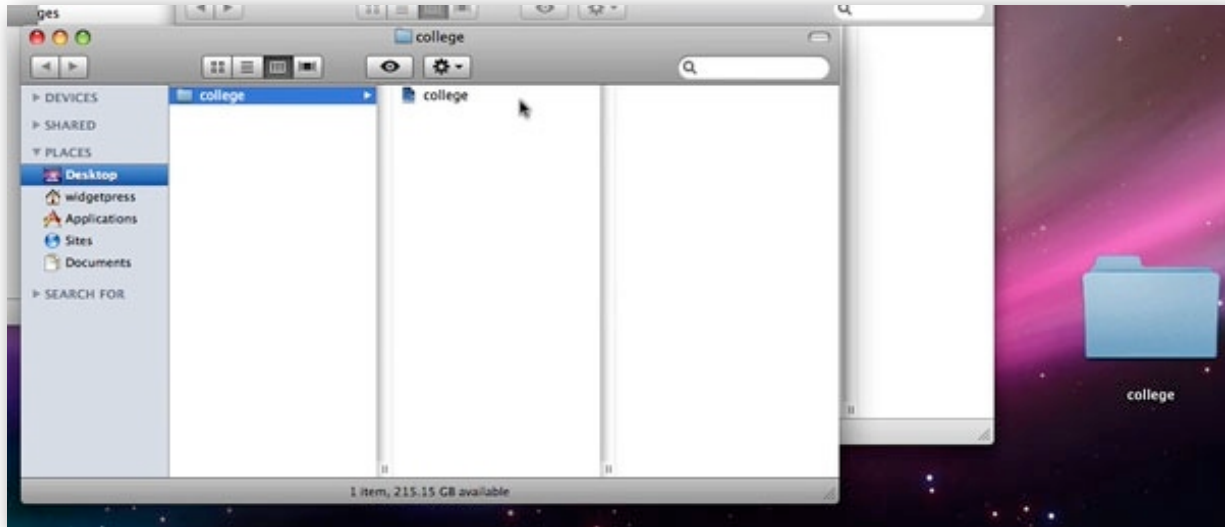
Step 1: Creating a new ModelBaker project

For this example, create a project, rename it "college.modelbaker", and save it to the desktop.



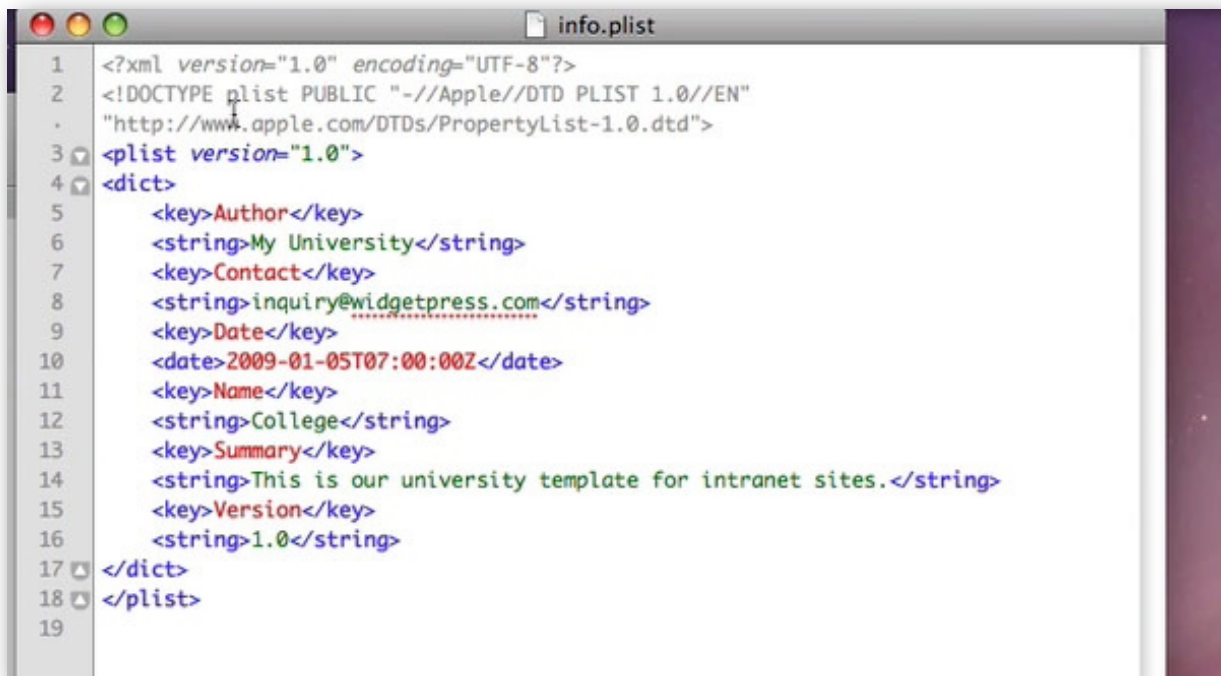
Step 2: Adding project files to a new folder

Create a new folder on your desktop and name it "college". After you create the "college" folder, drag the "college.modelbaker" file into the "college" folder.



Step 3: Creating an XML .plist file

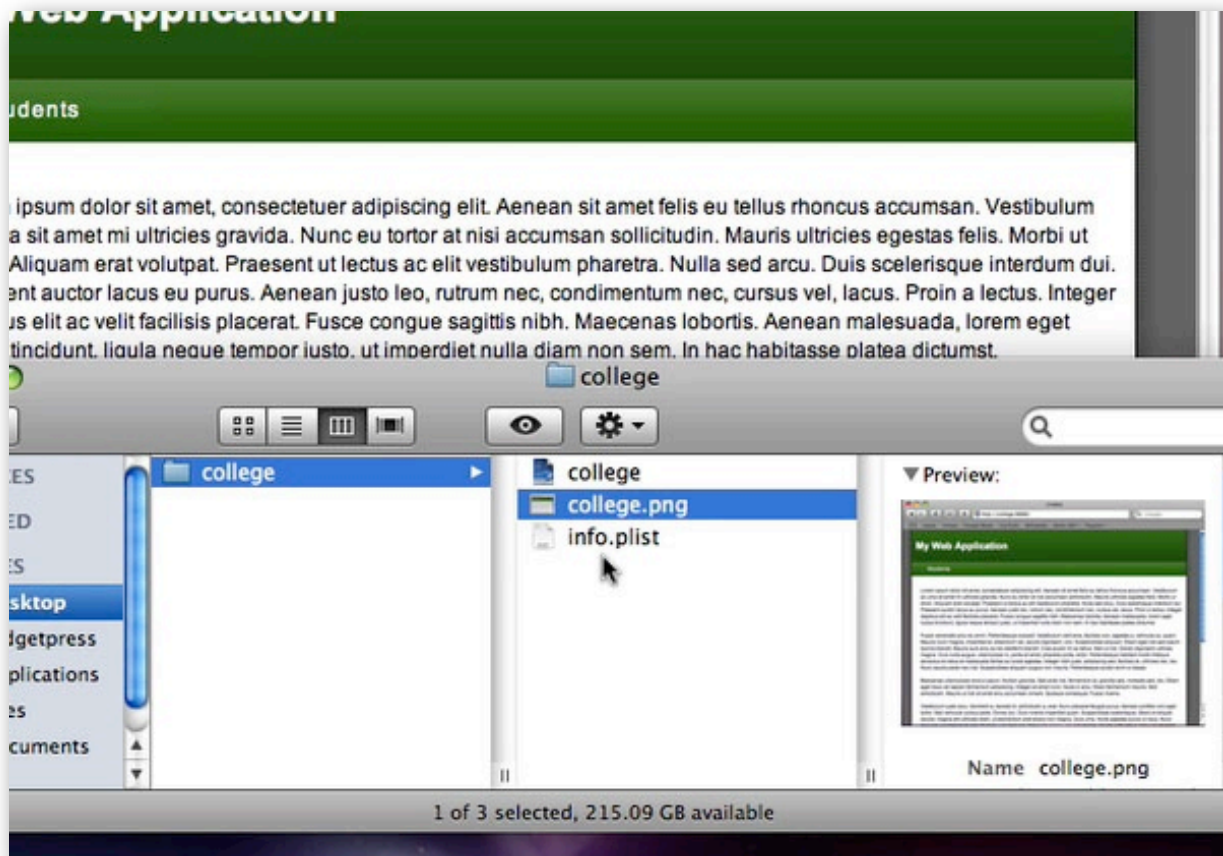
Create a new text file called "info.plist" and save it to your "college" folder. The "info.plist" file can be edited with TextEdit or any similar text editing application. If you have Apple's Xcode installed, you can use this to edit the .plist file in a nice GUI. Follow the XML elements and values examples below, and save your file.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
3 "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
4 <plist version="1.0">
5   <dict>
6     <key>Author</key>
7     <string>My University</string>
8     <key>Contact</key>
9     <string>inquiry@widgetpress.com</string>
10    <key>Date</key>
11    <date>2009-01-05T07:00:00Z</date>
12    <key>Name</key>
13    <string>College</string>
14    <key>Summary</key>
15    <string>This is our university template for intranet sites.</string>
16    <key>Version</key>
17    <string>1.0</string>
18  </dict>
19 </plist>
```

Step 4: Creating a PNG graphic file

Create a ModelBaker preview image by saving a "college.png" file into the "college" folder. This .PNG file gets loaded when you are reviewing the selection of Templates from the *Template Chooser*. These preview files are typically screenshots of your web application. You can use the *Preview* application to resize and crop your screenshot. ModelBaker will eventually scale this .PNG file to a resolution size of 252(w) x 192(h).



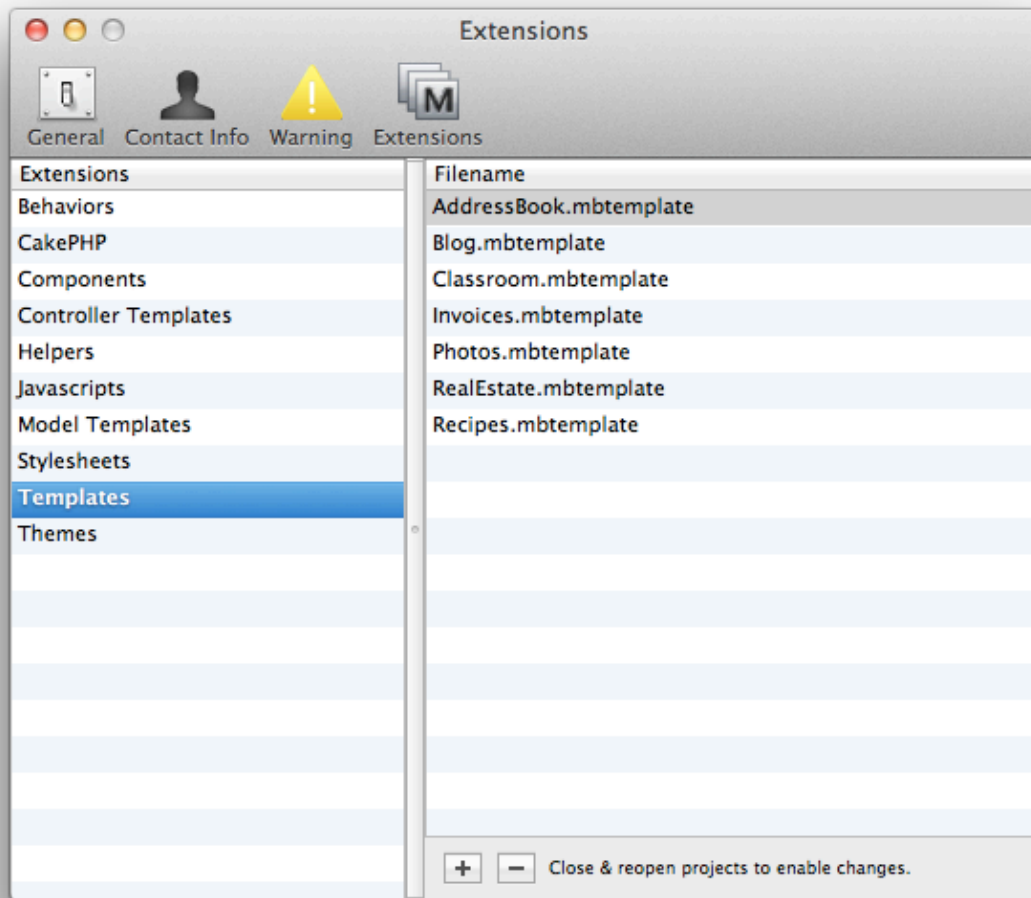
Step 5: Renaming a folder to a ModelBaker template file

Rename your "college" folder to "college.mbtemplate". By renaming the "college" folder, you are creating a plugin bundle which we will add to ModelBaker's "Application Support" folder.

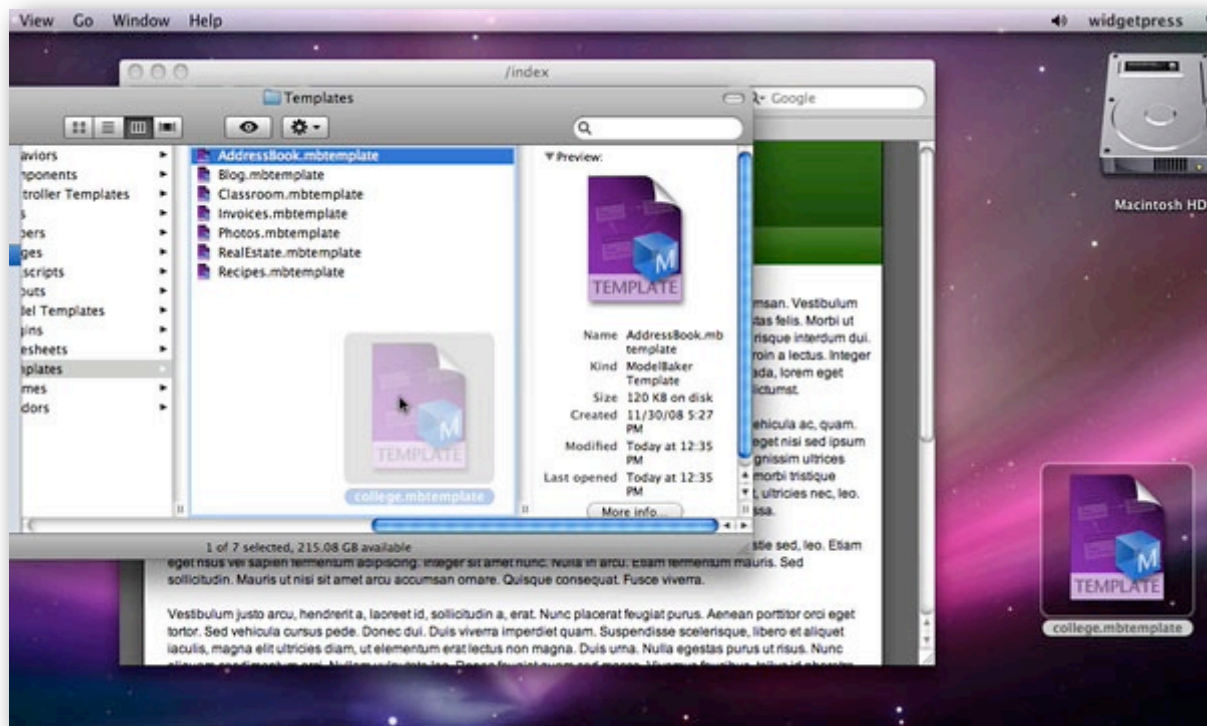


Step 6: Moving the template file to “*Application Support*”

You can add the “college.mbtemplate” to ModelBaker in two ways. One by adding it from the Extensions Manager from within the Preferences Window and the other by dragging and dropping the template file into the templates folder within Application Support.

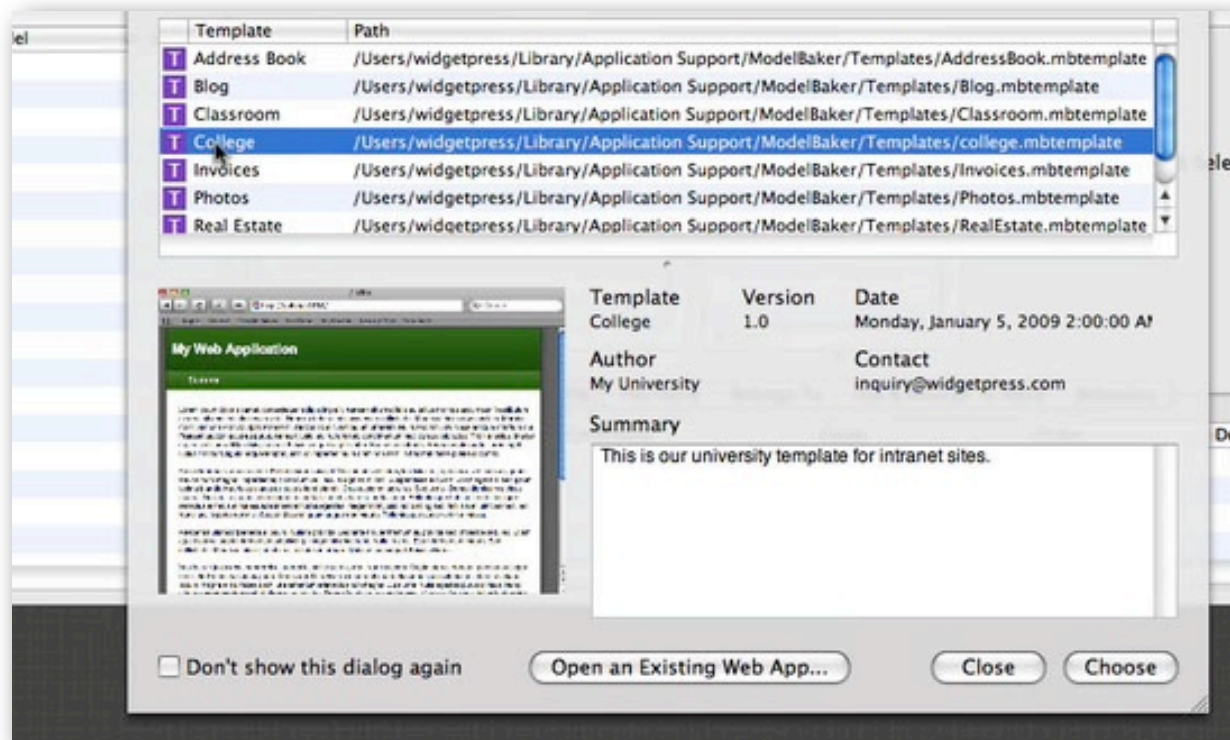


Drag the "college.mbtemplate" plugin bundle to your “~/Library/Application Support/ModelBaker/Templates” folder.



Step 7: Testing the template

Launch ModelBaker and select *'File ⇨ New from Template Chooser...'*. You should see your template listed in the window sheet that appears. By choosing your template you will create a new *"untitled"* project with all your entity models, attributes, relationships, validation rules and so on.



CREATING A CUSTOM THEME

In this chapter, we describes how to create a ModelBaker theme from scratch.

Note: This chapter assumes you have already created all of the required CSS, JS, and Layout files and that they are collected within a folder on your desktop. Pay special attention to the naming conventions in this chapter because a template needs to adhere to specific naming conventions.

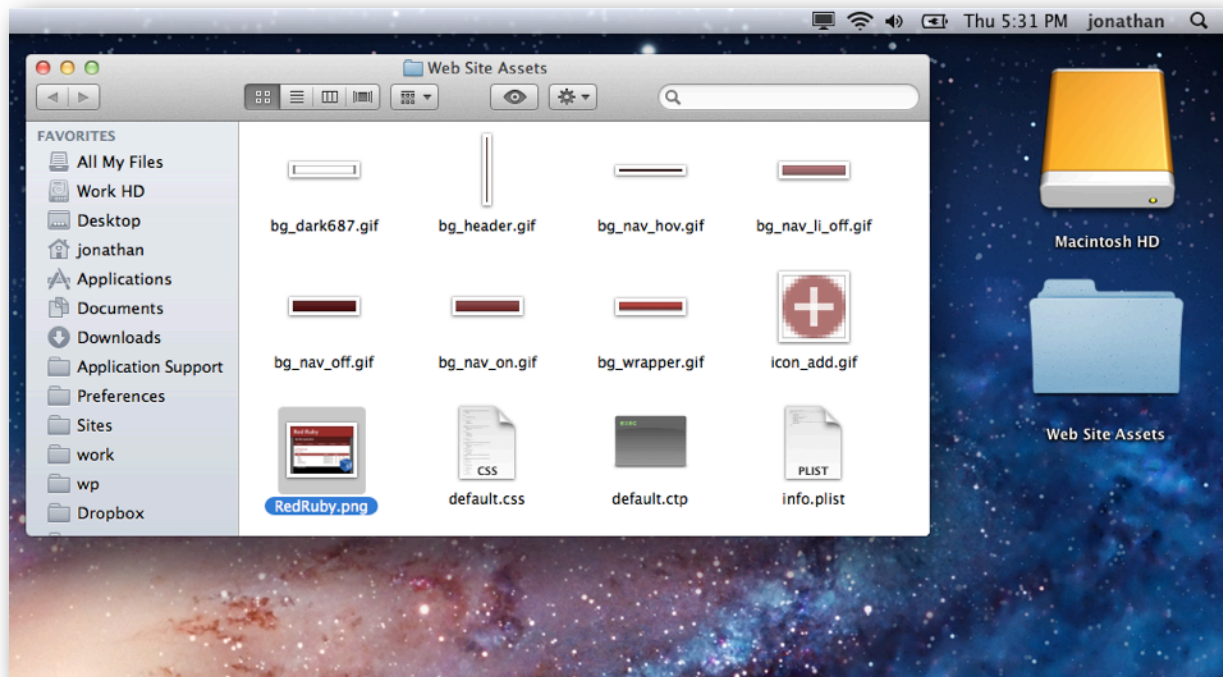
Step 1: Collecting your graphical assets

This step assumes you have already created your graphics assets.

- Collect these graphics assets together, including any “info.plist”, image, style sheet, javascript, and layout (“.ctp”) files. Place them in a folder on your desktop called “Web Site Assets”.



- Your collection of graphical assets should look something like the following:



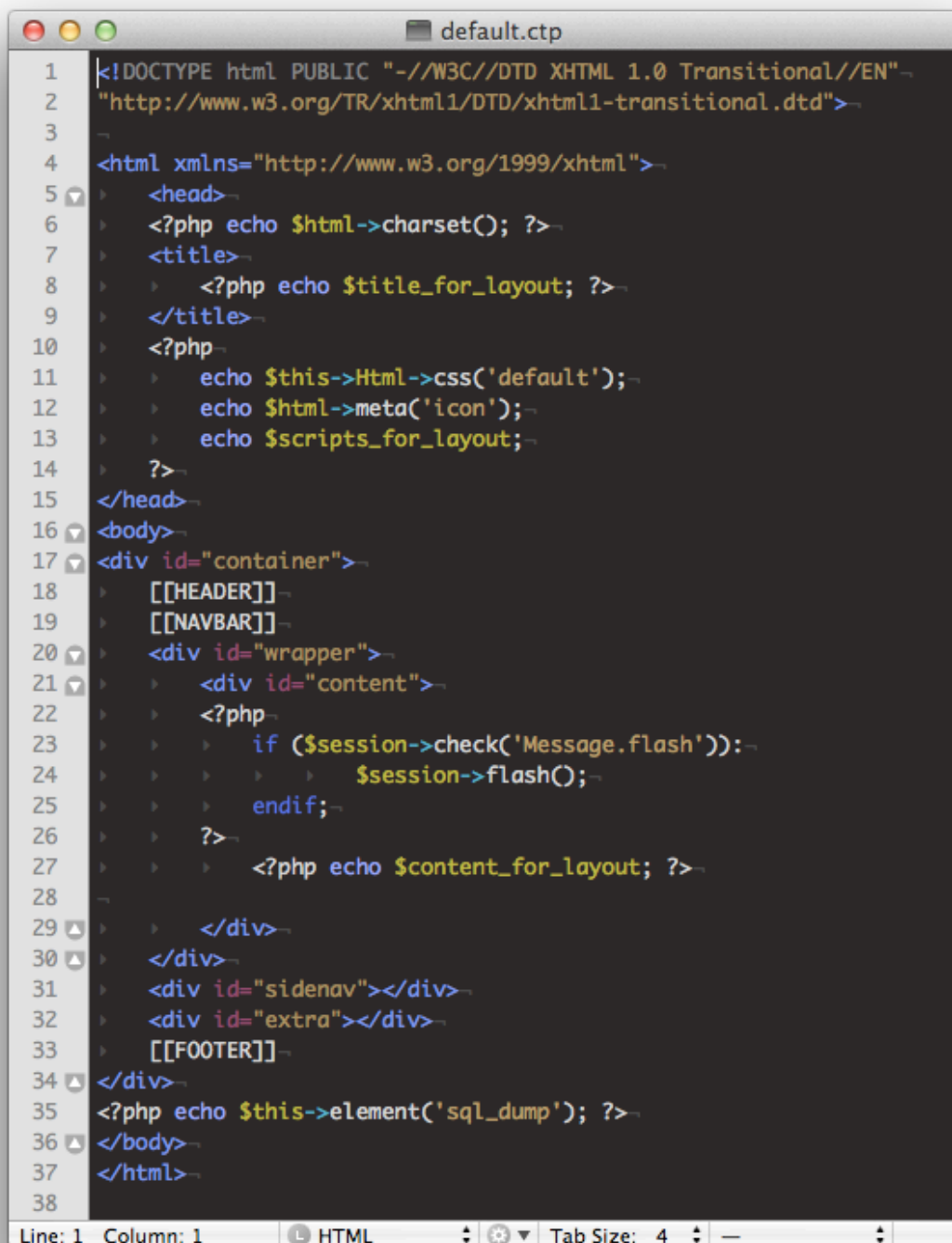
The “info.plist” is a simple XML document that has “.plist” file extension. The following screenshot shows the structure of the elements within the XML file:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
3  "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
4  <plist version="1.0">
5    <dict>
6      <key>Author</key>
7      <string>Widget Press, Inc.</string>
8      <key>Contact</key>
9      <string>inquiry@widgetpress.com</string>
10     <key>Date</key>
11     <date>2011-07-31T05:00:00Z</date>
12     <key>Name</key>
13     <string>Red Ruby</string>
14     <key>Summary</key>
15     <string>A simple single column, fixed format web application theme with rich red tones
16     throughout the application.</string>
17     <key>Version</key>
18     <string>2.1</string>
19   </dict>
20 </plist>

```

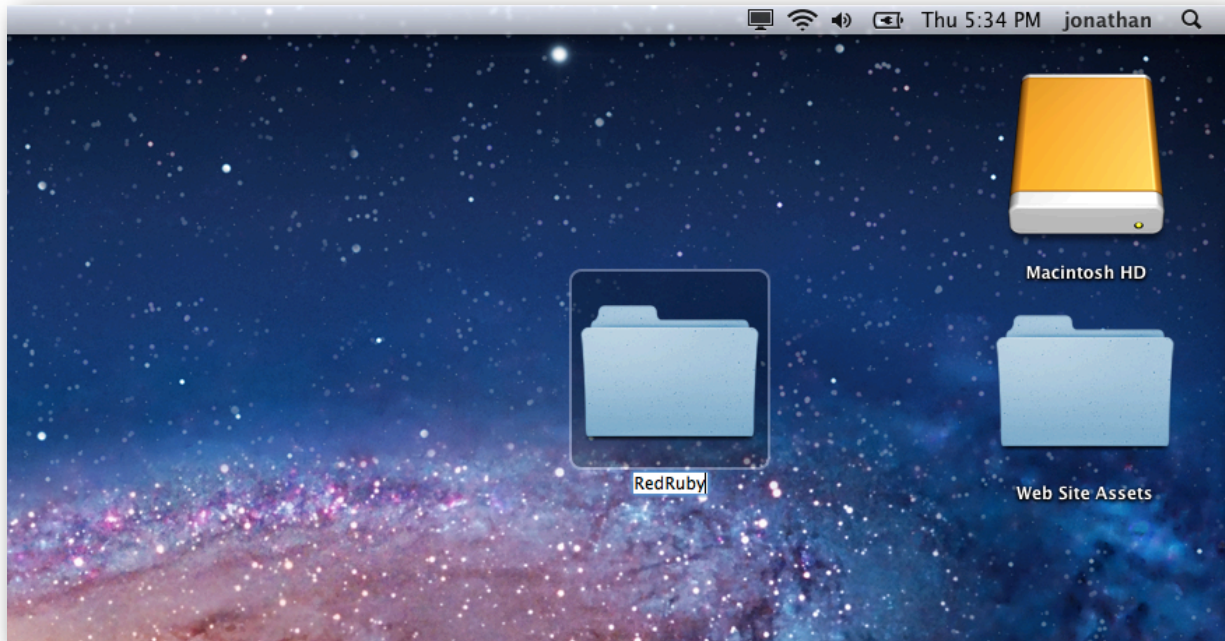
The “default.ctp” is a standard CakePHP template file that has a “.ctp” file extension. There are three places holders, “[[HEADER]]”, “[[NAVBAR]]” and “[[FOOTER]]” where ModelBaker will inject HTML/PHP content during its building.



```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml">
5 <head>
6 <?php echo $html->charset(); ?>
7 <title>
8 <?php echo $title_for_layout; ?>
9 </title>
10 <?php
11 > echo $this->Html->css('default');
12 > echo $html->meta('icon');
13 > echo $scripts_for_layout;
14 > ?>
15 </head>
16 <body>
17 <div id="container">
18 > [[HEADER]]
19 > [[NAVBAR]]
20 > <div id="wrapper">
21 > <div id="content">
22 > <?php
23 > > if ($session->check('Message.flash')):
24 > > > $session->flash();
25 > > > endif;
26 > > > ?>
27 > > > <?php echo $content_for_layout; ?>
28 > > >
29 > > </div>
30 > > </div>
31 > <div id="sidenav"></div>
32 > <div id="extra"></div>
33 > [[FOOTER]]
34 </div>
35 <?php echo $this->element('sql_dump'); ?>
36 </body>
37 </html>
38
```

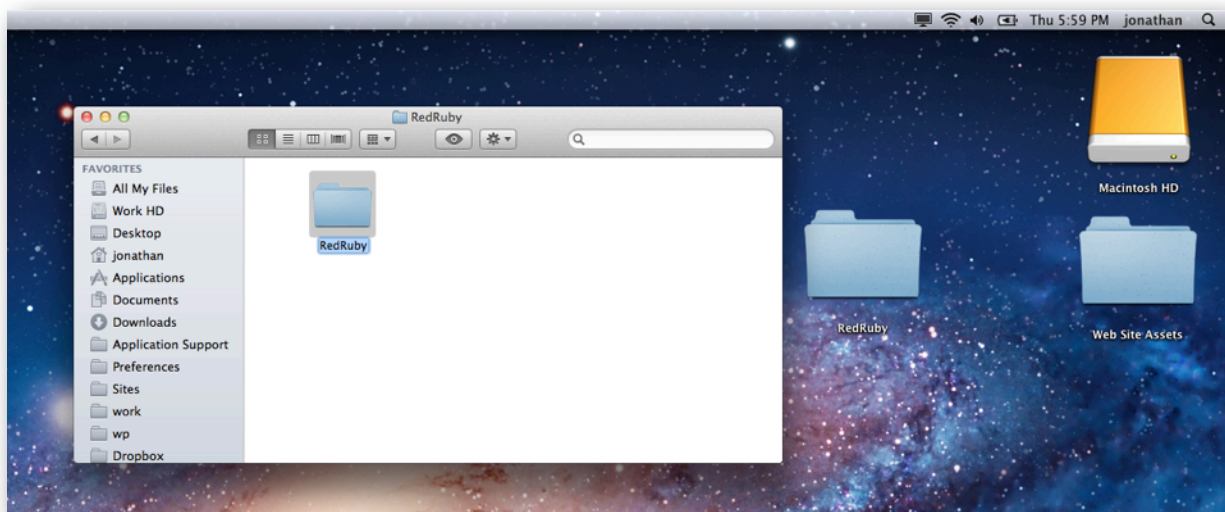

Step 2: Creating a new folder “RedRuby”

- Create a new folder on your desktop and name it "RedRuby". This folder will function as the foundation folder for our theme.



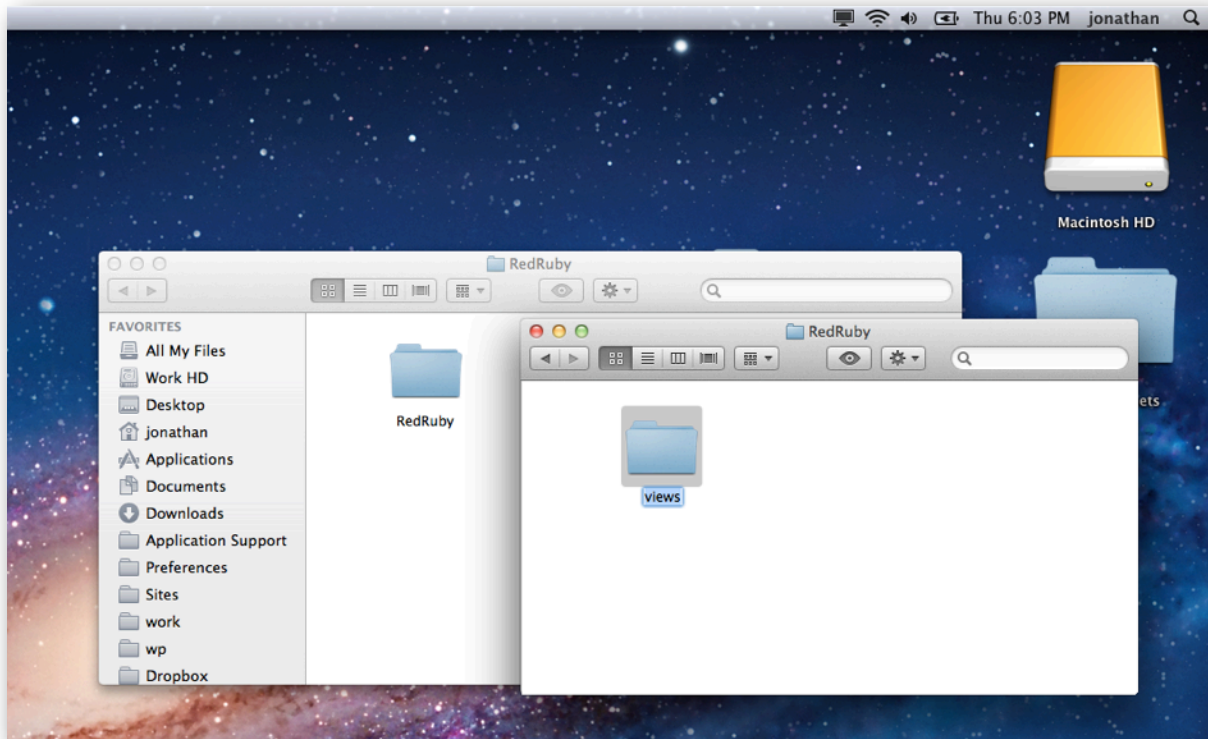
Step 3: Creating another new folder “RedRuby”

- Create another folder called "RedRuby" within the "RedRuby" folder.

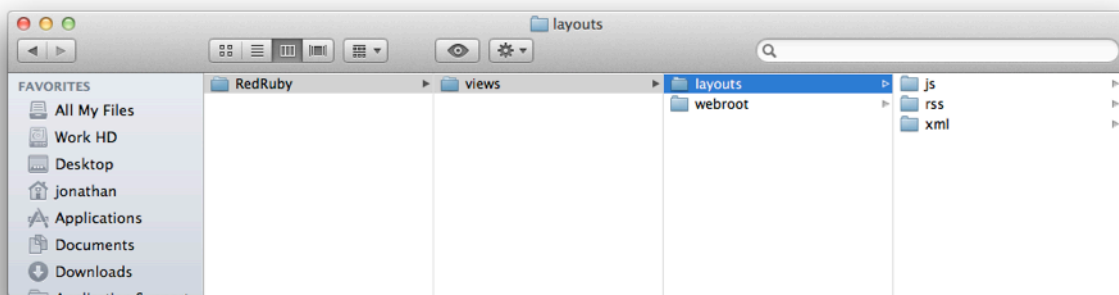


Step 4: Creating the theme directory structure

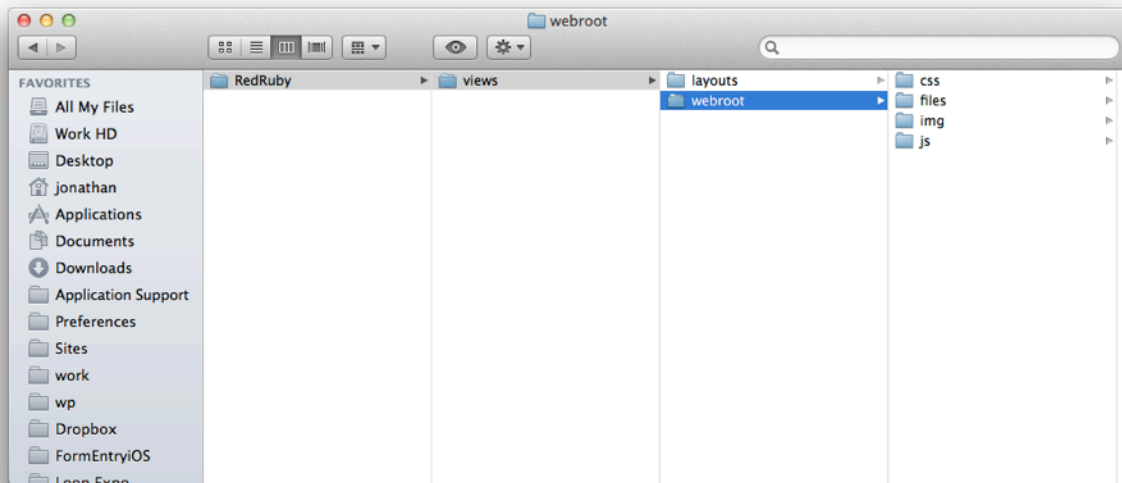
- Create a new folder within the innermost "RedRuby" folder called "views".



- In the “views” folder, create two folders, one called titled "layouts" and the other titled “webroot”.
- In the “layouts” folder, create three new folders: "js", "rss", and "xml".

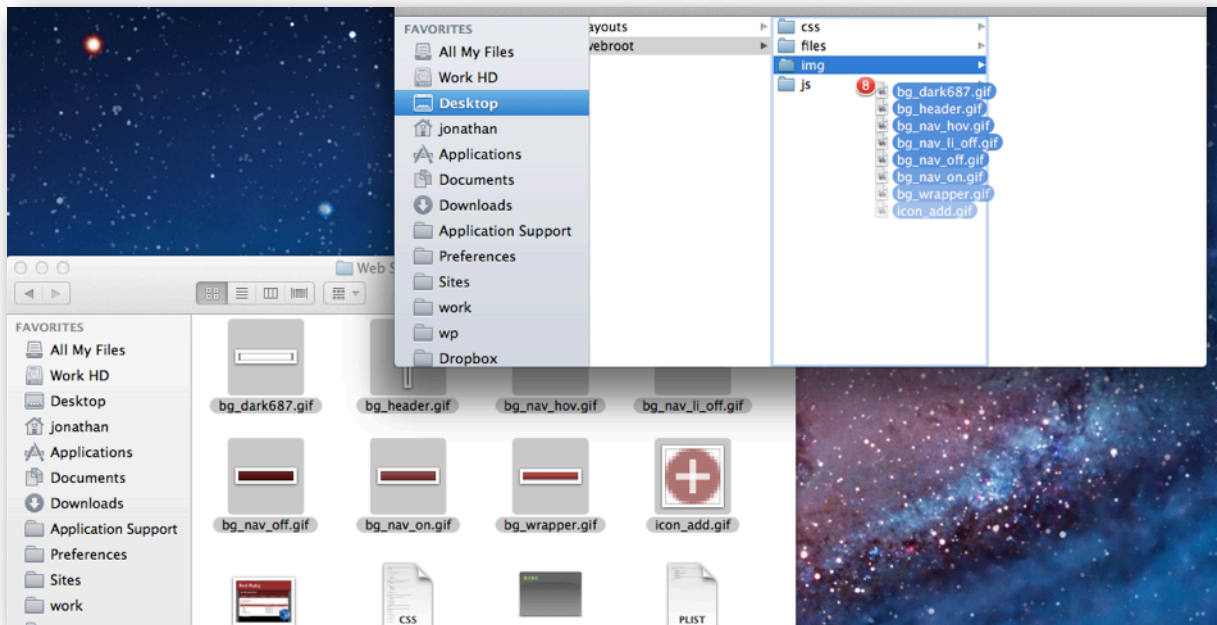


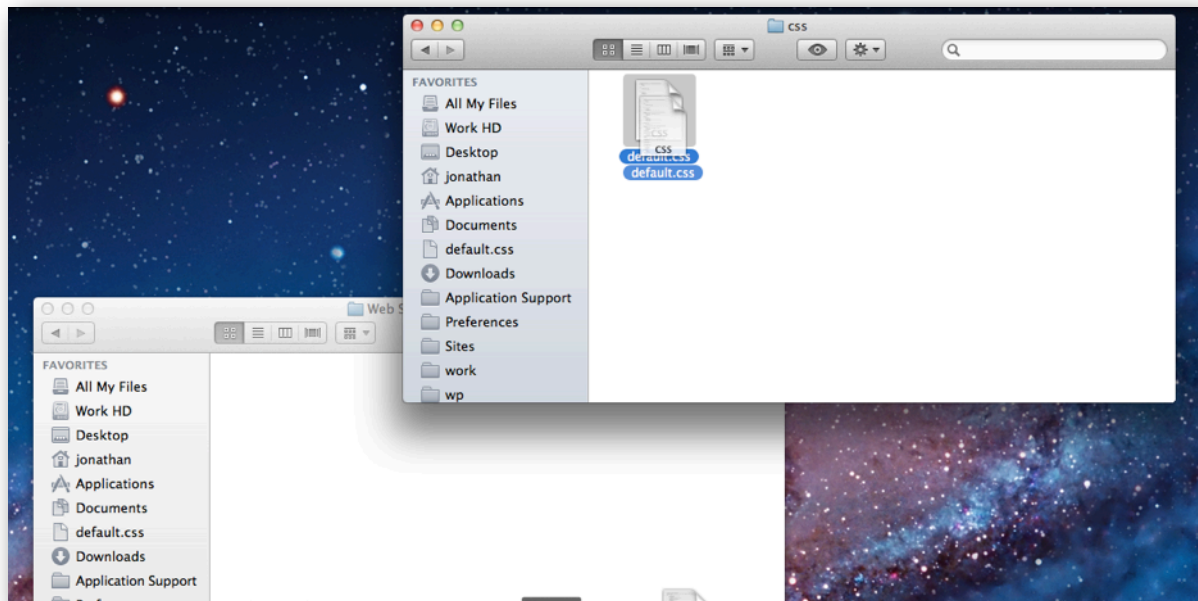
In the “webroot” folder, create four new folders: "css", "files", "img", and "js".



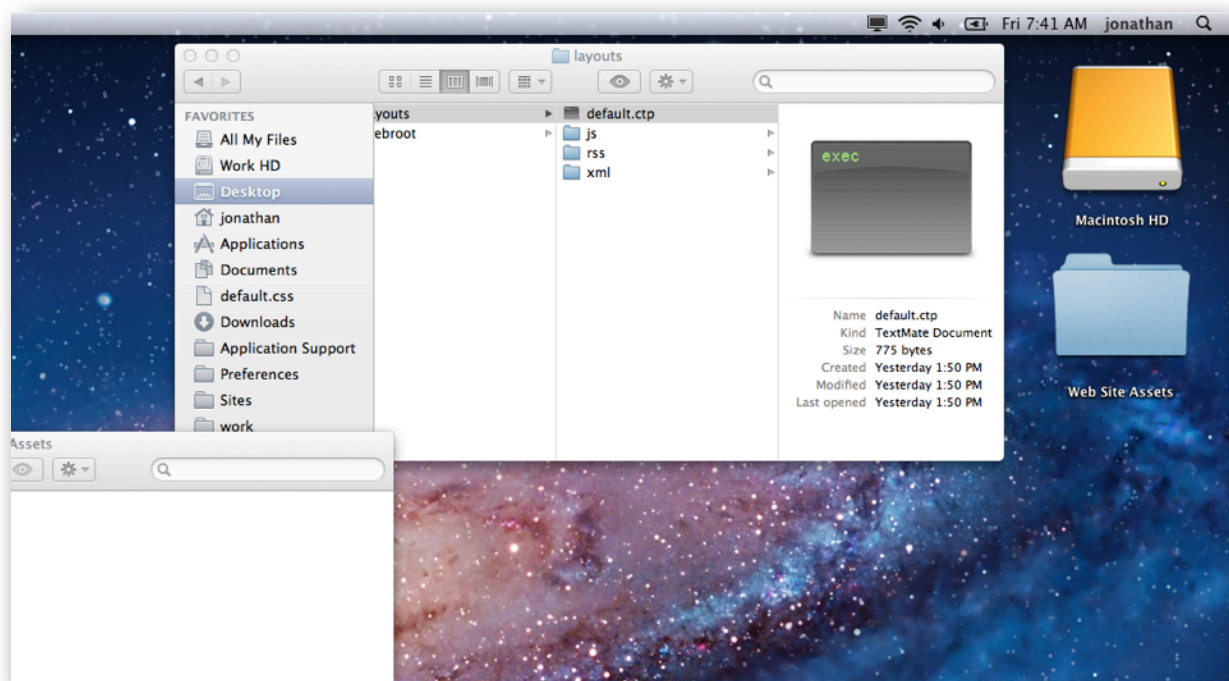
Step 5: Adding your assets

- Drag and drop your assets into the appropriate folders (e.g. graphics into the “img” folder, stylesheets into the “css” folder).

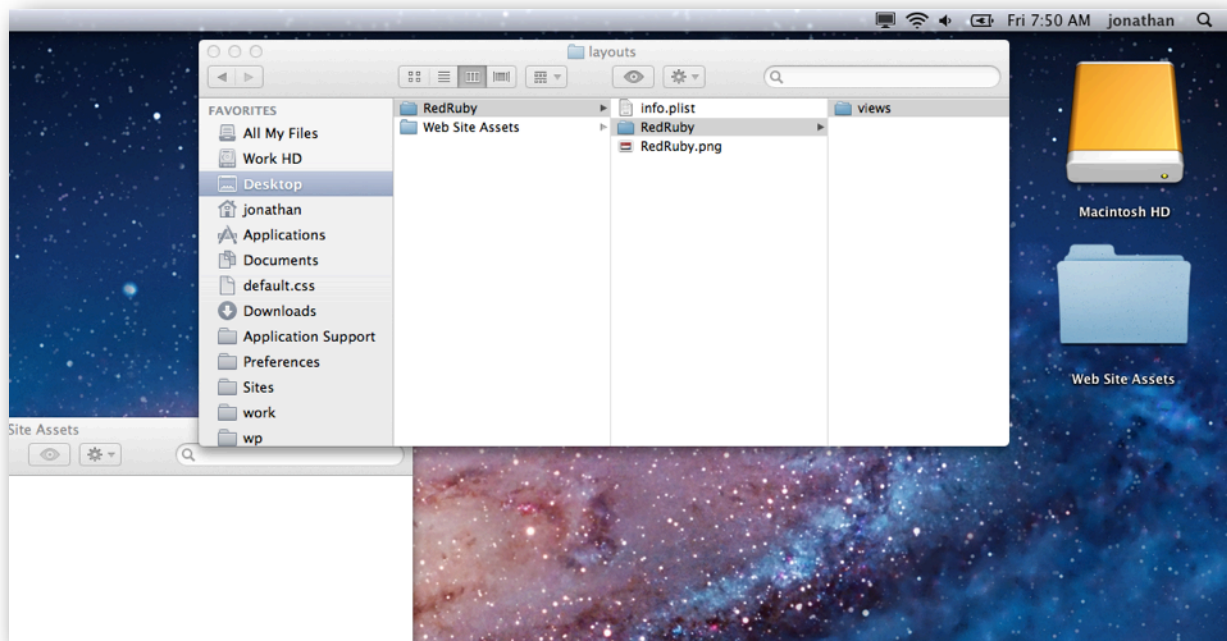




Move your “default.ctp” file into “layouts” folder at the same level with “js”, “rss” and “xml” folders.



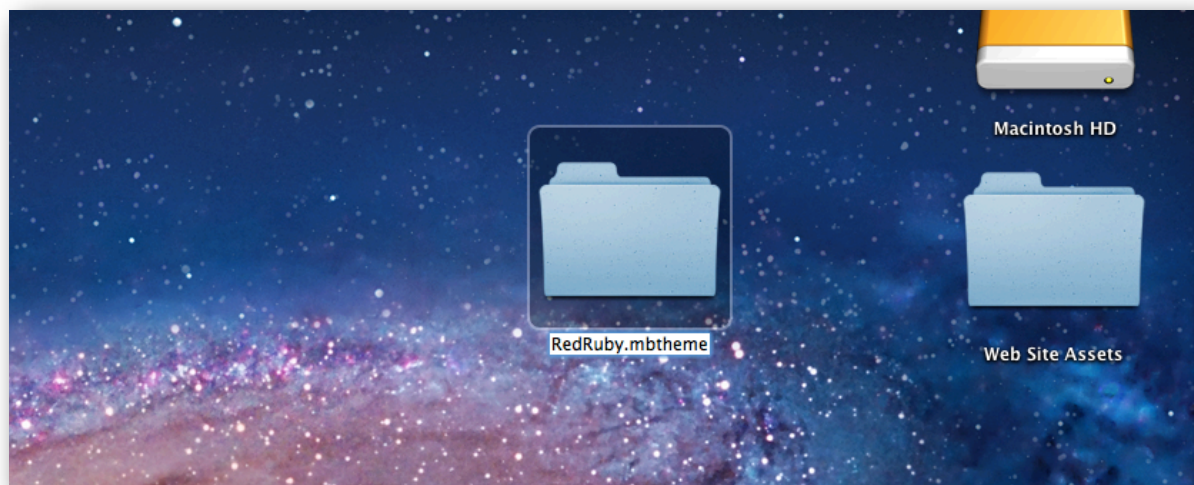
- Move your preview “RedRuby.png” file and the “info.plist” file to the main "RedRuby" directory level. It should be on the same tree level as the second "RedRuby" directory.

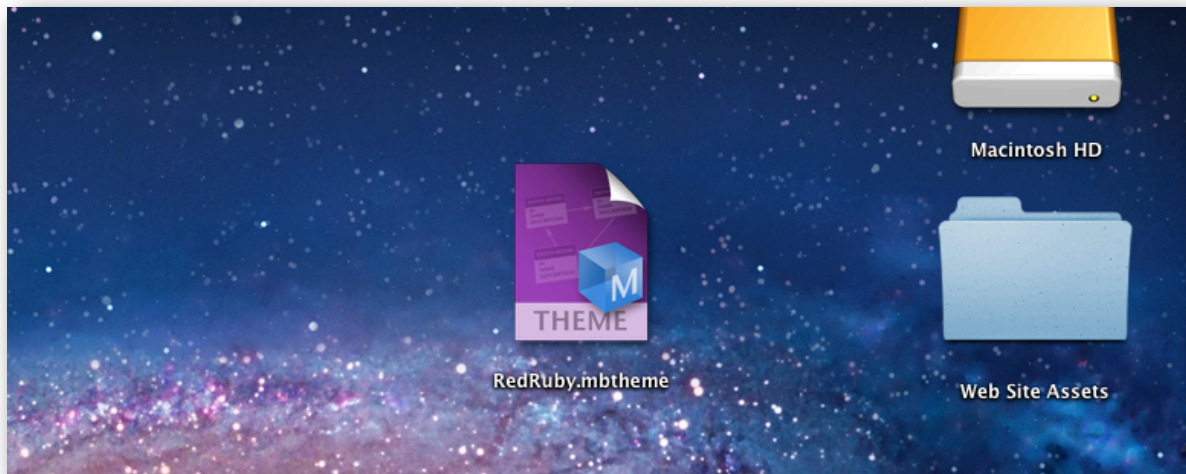


Important: Please note the naming conventions used, make sure the folders and preview image are named the same.

Step 6: Renaming the file with ModelBaker theme extension

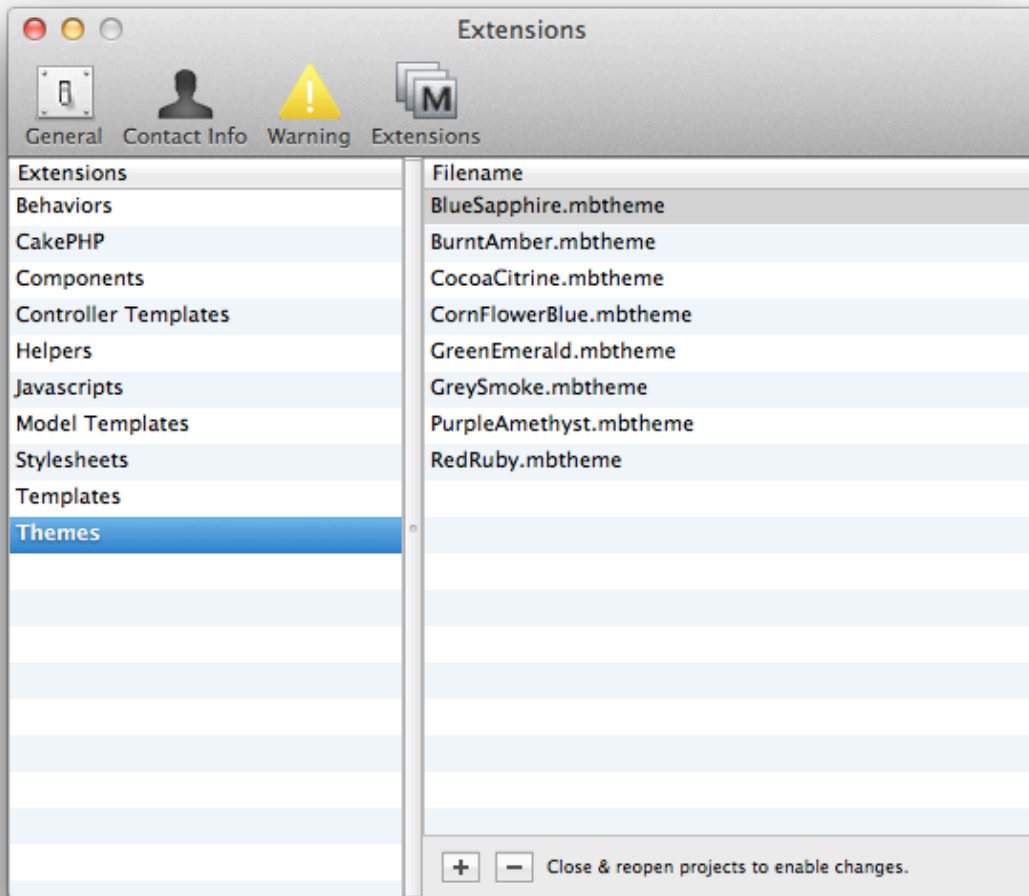
Add the ".mbtheme" file extension to the "RedRuby" folder.



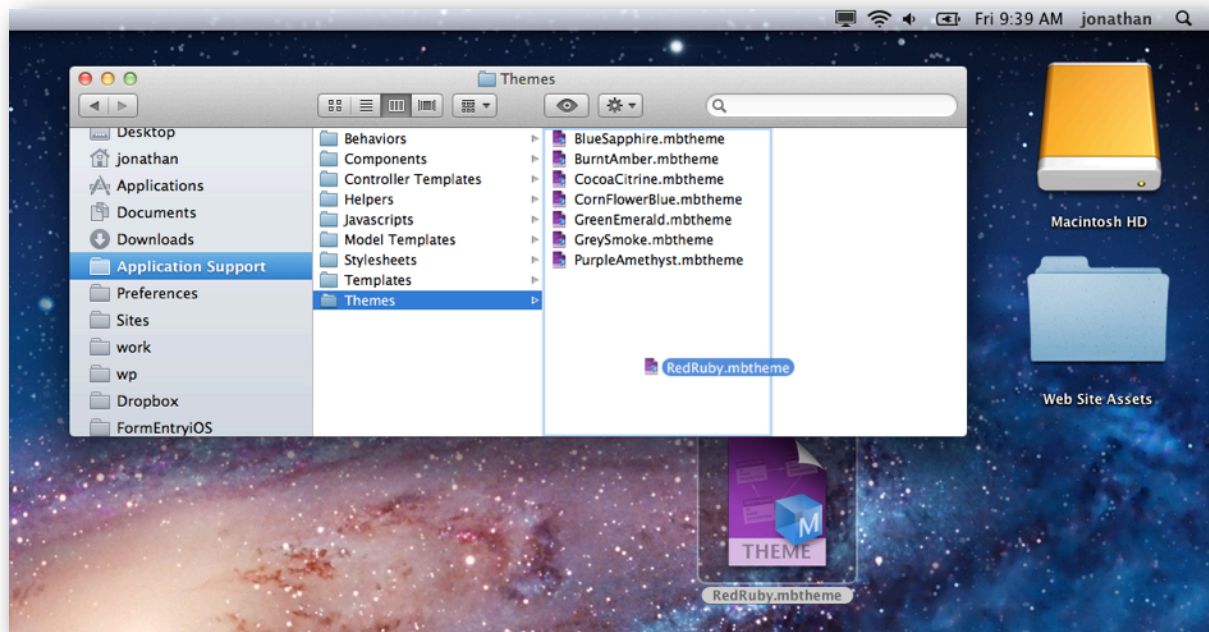


Step 7: Adding the file to “*Application Support*”

You can add the “RedRuby.mbtheme” in two different ways. One by adding it from the Extensions Manager located in the Preference window or by dragging and dropping the theme into the Themes folder.



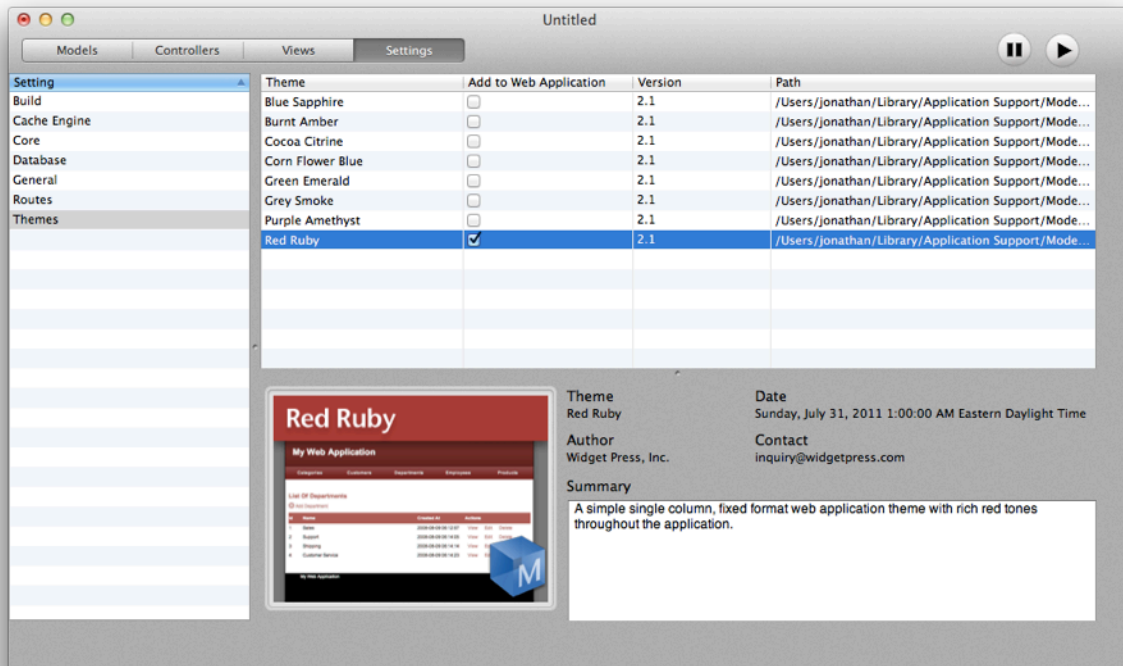
Drag the "RedRuby.mbtheme" file to your "~/Library/Application Support/ModelBaker/Themes" folder. This is how ModelBaker themes are installed. Once installed, relaunch ModelBaker.



Step 8: Testing the new theme

Your custom ModelBaker Theme is now installed.

- In the ‘Settings’ tab (⌘4), select ‘Themes’ from the ‘Setting’ table located on the left hand side of the window.
- You newly added created theme should appear in the ‘Theme’ list.
- Select the new template to apply it to your project.



MOVING TO A HOSTING SERVER

This chapter discusses some of the aspects of moving your web application to a remote hosting server.

Important: *This chapter assumes you have sufficient access and permissions to upload files to your hosting server. It is also assumed you know how to log in and upload files, and that you have created a database. Make sure you know the user name, password, and database name along with any other additional settings.*

Step 1: Changing your Database Settings

Since you have been developing locally on your Mac, you’ve been updating with your local MySQL database settings. We will need to change these settings before we upload our project to your remote web server.

To update your database settings, do the following:

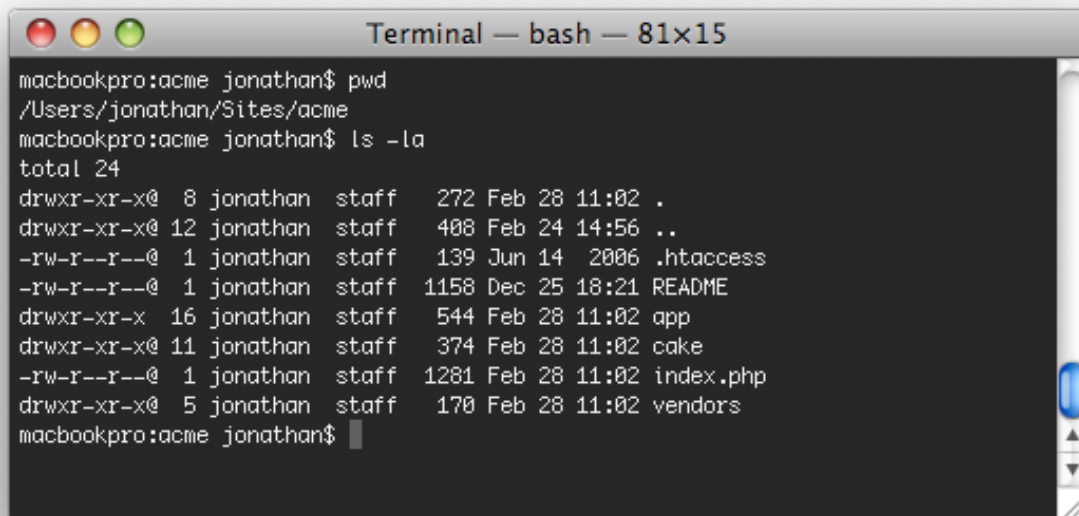
- Duplicate the “/app/config/database.php” file and rename it to “database.local.php”.
- Open the “database.php” file in a text editor and change the default settings to reflect your hosting environment settings. You will typically update the ‘host’, ‘login’, ‘password’, and ‘database’ values. Save then edited “database.php” file.

```
<?php
class DATABASE_CONFIG {
    var $default = array(
        'driver' => 'mysql',
        'persistent' => false,
        'host' => 'localhost',
        'port' => '',
        'login' => 'foo',
        'password' => 'bar',
        'database' => 'acme',
        'schema' => '',
        'prefix' => '',
        'encoding' => ''
    );
}
```

Step 2: Uploading to your Hosting Site

Note: uploading to your hosting site is dependent upon the types of access you have. You may have FTP, SFTP, or browser based uploading access. Please consult your internet hosting provider for the type of access you have.

You will need to upload the contents of the folder located in your build path setting. The build path folder contains all the files needed for your web application. Do not upload the cached files in the “/app/tmp” directory, but do upload the directory structure. Also, make sure you upload everything, including the hidden “.htaccess” files. ***These files are not visible in the Apple Finder.***

A screenshot of a macOS Terminal window titled "Terminal — bash — 81x15". The window shows a user named jonathan at a macbookpro in the directory /Users/jonathan/Sites/acme. The user has run the 'pwd' command, which confirms the current directory. Then, the user has run 'ls -la', which lists the contents of the directory. The output shows a total size of 24, followed by a list of files and directories with their permissions, owner, group, size, and modification date. The files listed are: a hidden directory '.', a hidden directory '..', a hidden file '.htaccess', a file 'README', a directory 'app', a directory 'cake', a file 'index.php', and a directory 'vendors'.

```
macbookpro:acme jonathan$ pwd
/Users/jonathan/Sites/acme
macbookpro:acme jonathan$ ls -la
total 24
drwxr-xr-x@ 8 jonathan  staff   272 Feb 28 11:02 .
drwxr-xr-x@ 12 jonathan  staff   408 Feb 24 14:56 ..
-rw-r--r--@ 1 jonathan  staff   139 Jun 14  2006 .htaccess
-rw-r--r--@ 1 jonathan  staff  1158 Dec 25 18:21 README
drwxr-xr-x 16 jonathan  staff   544 Feb 28 11:02 app
drwxr-xr-x@ 11 jonathan  staff   374 Feb 28 11:02 cake
-rw-r--r--@ 1 jonathan  staff  1281 Feb 28 11:02 index.php
drwxr-xr-x@ 5 jonathan  staff   170 Feb 28 11:02 vendors
macbookpro:acme jonathan$
```

After upload, make sure the remote “/app/tmp” directory is readable and writable. Also, make sure the directories within the “tmp” folder are readable and writable as well.

Important: Make sure the uploaded server does not have cached files from your local development.

Step 3: Installing your Database Schema

When you build ModelBaker projects locally, you are actually dropping and inserting tables into your local version of MySQL. Your entire database schema gets generated with all your database types, relationships, and even random data (if applied) into a local schema script file. You will need to use this file to insert into your remote database hosting environment.

Installing this schema file depends on the access you and the type of admin access you use for your remote database. Please consult your internet hosting provider for the details of your database configuration.

To update your database remotely, you can probably do the following:

- Locate the “/app/config/schema/schema.sql” file.
- If you are using “*phpMyAdmin*”, you can copy and paste the text from this file into the query text area of your web browser and execute the query.
- If you have shell (SSH) access, can typically execute commands (with parameters) from a terminal command line against your remote MySQL database. This would typically look something like:

```
% mysql -hmysql.example.com -ufoo -pbar modelbaker_db <
/www/app/config/schema/schema.sql
```

Here is what the above command line basically does: tell ‘*mysql*’ at the host ‘*mysql.example.com*’ to log in using the user name ‘*foo*’ and the password ‘*bar*’. Then on the database ‘*modelbaker_db*’, import the schema file located at “/www/app/config/schema/schema.sql”.