XML Processing (optional)

tutorial #xml-xml02

James L. Parry
B.C. Institute of Technology

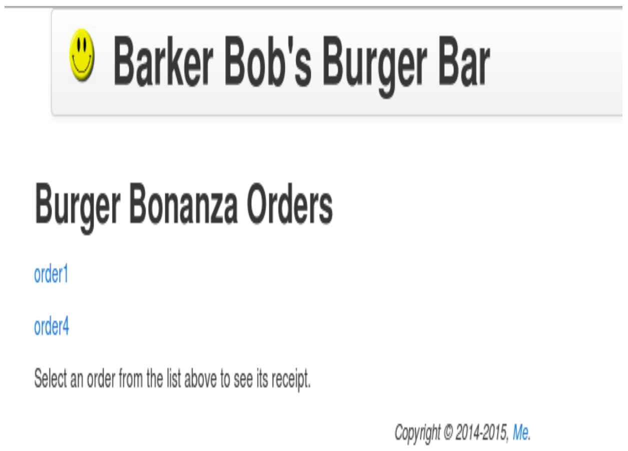
Tutorial Goals

This tutorial is meant to give you some exposure to working with SimpleXML to process an XML document.

It is just a commented walkthrough of the example-simplexml project.

Homepage

Out of the box, the homepage shows a list of all the orders in the data folder.



Burger Bonanza Orders

order1

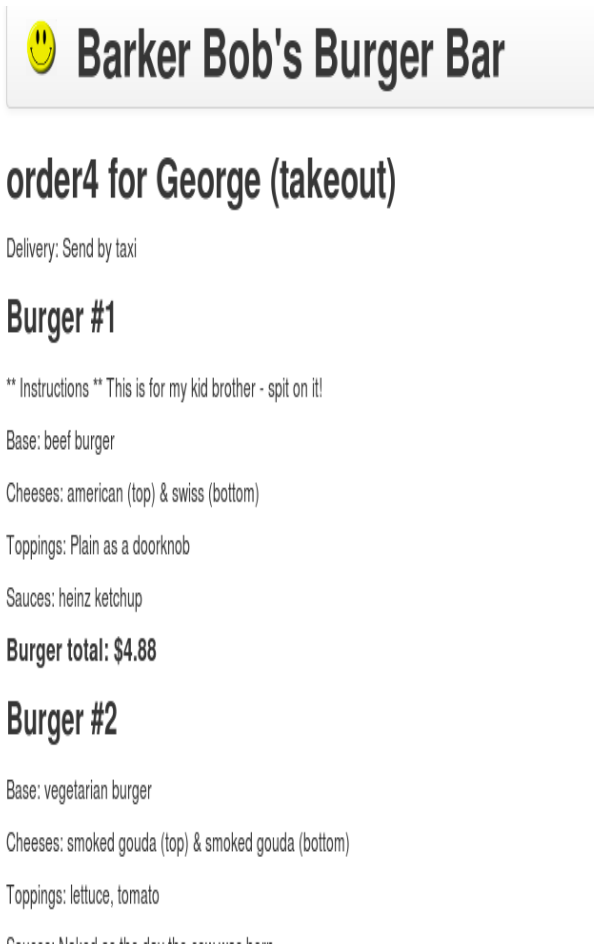
order4

Select an order from the list above to see its receipt.

Copyright © 2014-2015, Me.

Order Details

Clicking on an order causes its details to be shown



Barker Bob's Burger Bar

order4 for George (takeout)

Delivery: Send by taxi

Burger #1

** Instructions ** This is for my kid brother - spit on it!

Base: beef burger

Cheeses: american (top) & swiss (bottom)

Toppings: Plain as a doorknob

Sauces: heinz ketchup

Burger total: $4.88

Burger #2

Base: vegetarian burger

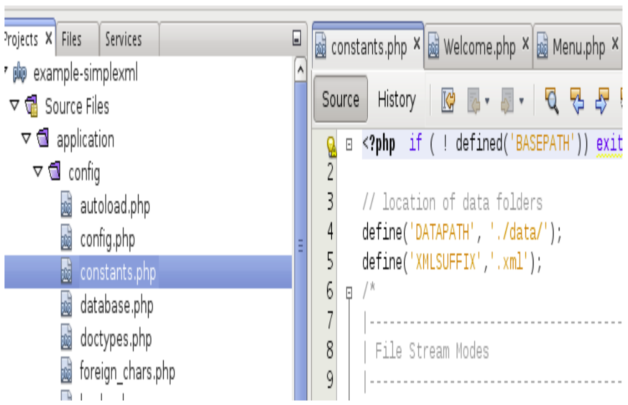
Cheeses: smoked gouda (top) & smoked gouda (bottom)

Toppings: lettuce, tomato

What's Where

If you check the project, nothing is autoloaded, and there is no database.

The only configuration procided for is a couple of constants, one pointing at the folder containing XML data and the other defining the XML suffix. These are convenience constants that came from refactoring.



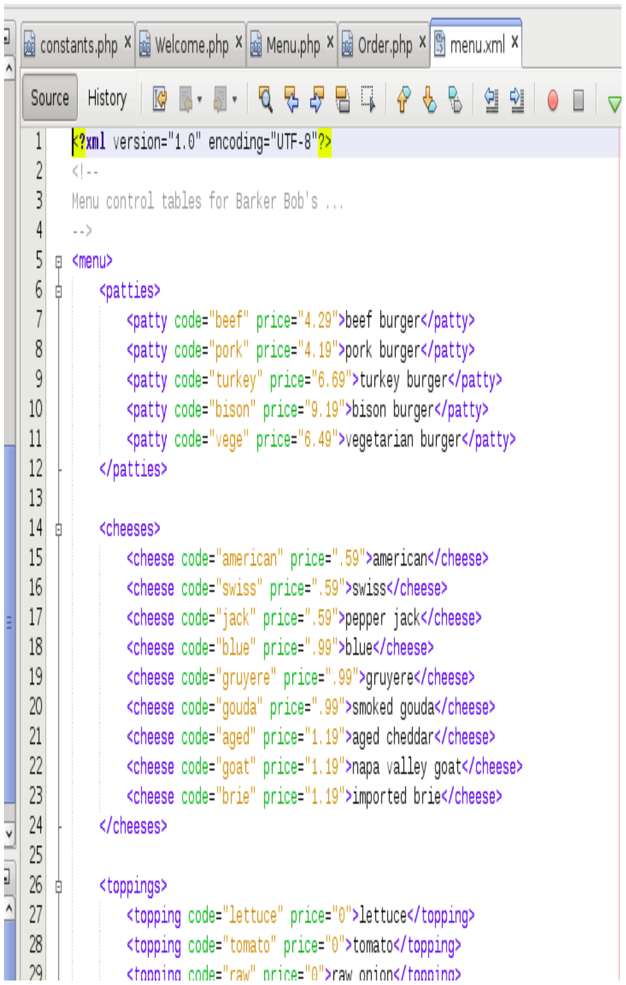
```
<?php  if ( ! defined('BASEPATH')) exit

// location of data folders
define('DATAPATH', './data/');
define('XMLSUFFIX','.xml');
/*
|--------------------------------------
| File Stream Modes
|--------------------------------------
```

Menu Data

The menu data comes from `/data/menu.xml`.

It defines all of the things that might be configured when building one of Barker Bob's burgers.

These ingredient groupings are simply children of the XML document's root element.

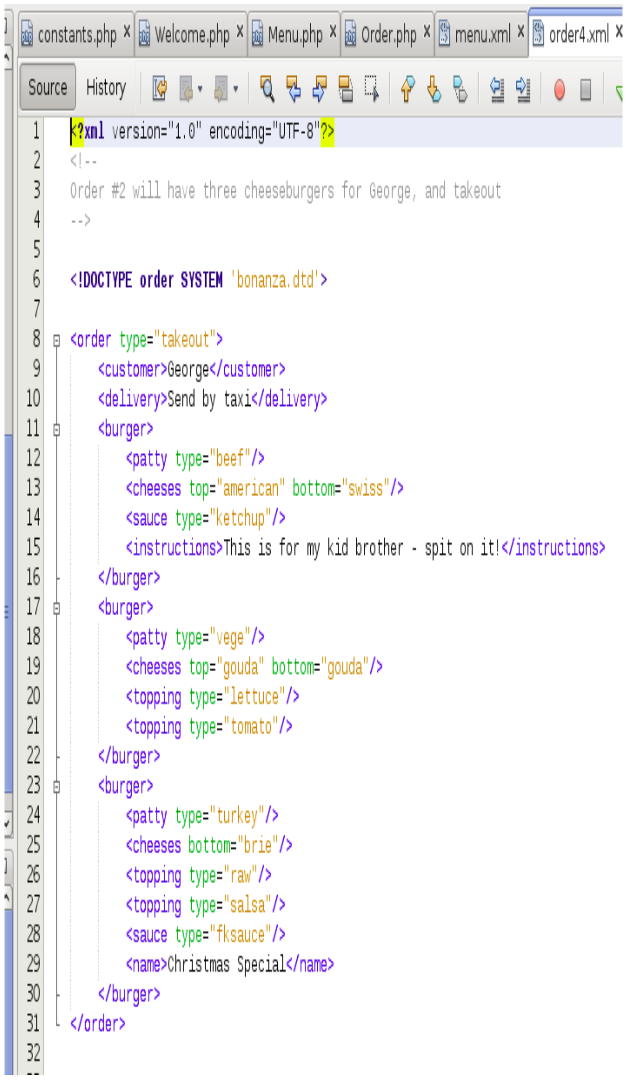


```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--
Menu control tables for Barker Bob's ...
-->
<menu>
    <patties>
        <patty code="beef" price="4.29">beef burger</patty>
        <patty code="pork" price="4.19">pork burger</patty>
        <patty code="turkey" price="6.69">turkey burger</patty>
        <patty code="bison" price="9.19">bison burger</patty>
        <patty code="vege" price="6.49">vegetarian burger</patty>
    </patties>

    <cheeses>
        <cheese code="american" price=".59">american</cheese>
        <cheese code="swiss" price=".59">swiss</cheese>
        <cheese code="jack" price=".59">pepper jack</cheese>
        <cheese code="blue" price=".99">blue</cheese>
        <cheese code="gruyere" price=".99">gruyere</cheese>
        <cheese code="gouda" price=".99">smoked gouda</cheese>
        <cheese code="aged" price="1.19">aged cheddar</cheese>
        <cheese code="goat" price="1.19">napa valley goat</cheese>
        <cheese code="brie" price="1.19">imported brie</cheese>
    </cheeses>

    <toppings>
        <topping code="lettuce" price="0">lettuce</topping>
        <topping code="tomato" price="0">tomato</topping>
        <topping code="raw" price="0">raw onion</topping>
```

Order Data

All of the other XML files in the `/data` folder are order documents, which have been conveniently named.

The contents of one of those is shown to the right.

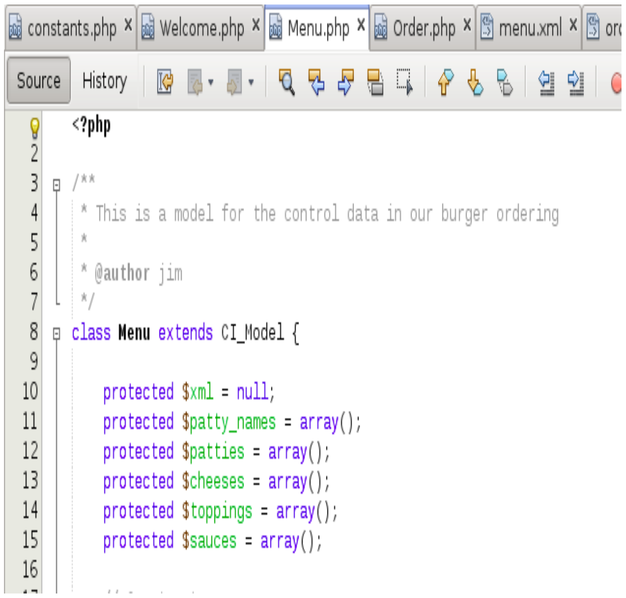


```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--
Order #2 will have three cheeseburgers for George, and takeout
-->

<!DOCTYPE order SYSTEM 'bonanza.dtd'>

<order type="takeout">
    <customer>George</customer>
    <delivery>Send by taxi</delivery>
    <burger>
        <patty type="beef"/>
        <cheeses top="american" bottom="swiss"/>
        <sauce type="ketchup"/>
        <instructions>This is for my kid brother - spit on it!</instructions>
    </burger>
    <burger>
        <patty type="vege"/>
        <cheeses top="gouda" bottom="gouda"/>
        <topping type="lettuce"/>
        <topping type="tomato"/>
    </burger>
    <burger>
        <patty type="turkey"/>
        <cheeses bottom="brie"/>
        <topping type="raw"/>
        <topping type="salsa"/>
        <sauce type="fksauce"/>
        <name>Christmas Special</name>
    </burger>
</order>
```

Menu Model

One of the two models provided is the `Menu` model.

It has an `xml` property for the root element of the XML document, a `patty_names` property holding an associative array to populate drop-down lists, and then properties for each collection of types of ingredients.



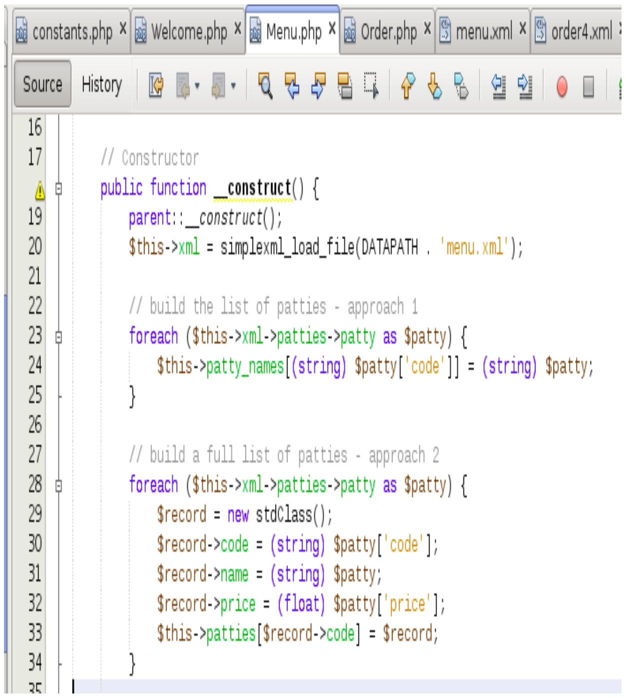
```
<?php

/**
 * This is a model for the control data in our burger ordering
 *
 * @author jim
 */
class Menu extends CI_Model {

    protected $xml = null;
    protected $patty_names = array();
    protected $patties = array();
    protected $cheeses = array();
    protected $toppings = array();
    protected $sauces = array();

```

Menu Model Constructor

The Model constructor loads the `menu.xml` document, and then traverses it in different ways to build the model properties.

`patty_names` is built by constructing a simple associative array, using the code and name of each patty. Note that the SimpleXMLElement pieces are cast as strings.

Note also that the "patty" objects in the the `patties` property are objects constructed onthe fly.

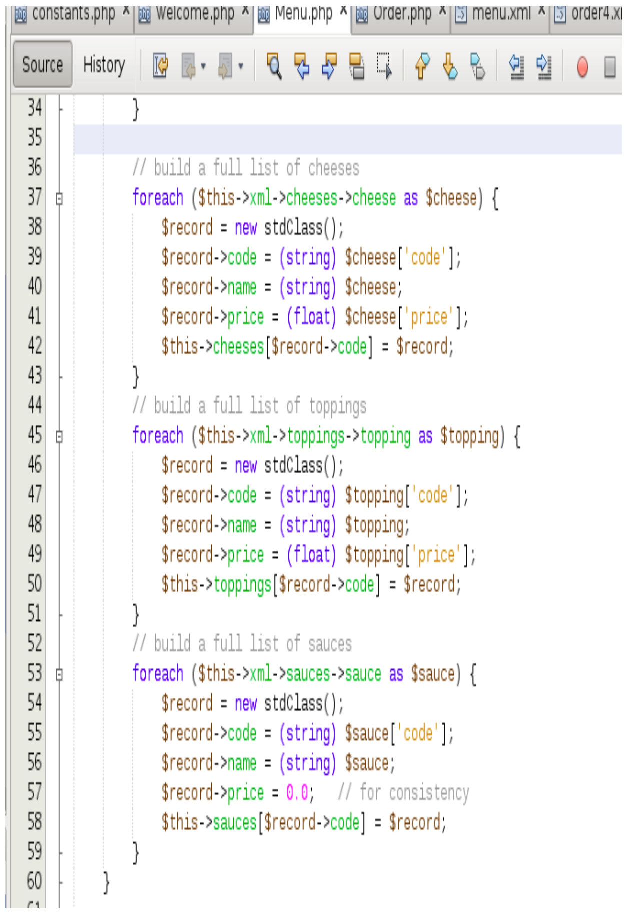


```

    // Constructor
    public function __construct() {
        parent::__construct();
        $this->xml = simplexml_load_file(DATAPATH . 'menu.xml');

        // build the list of patties - approach 1
        foreach ($this->xml->patties->patty as $patty) {
            $this->patty_names[(string) $patty['code']] = (string) $patty;
        }

        // build a full list of patties - approach 2
        foreach ($this->xml->patties->patty as $patty) {
            $record = new stdClass();
            $record->code = (string) $patty['code'];
            $record->name = (string) $patty;
            $record->price = (float) $patty['price'];
            $this->patties[$record->code] = $record;
        }
```

Menu Model Constructor Helpers

Here are the other property constructors, all done similarly.

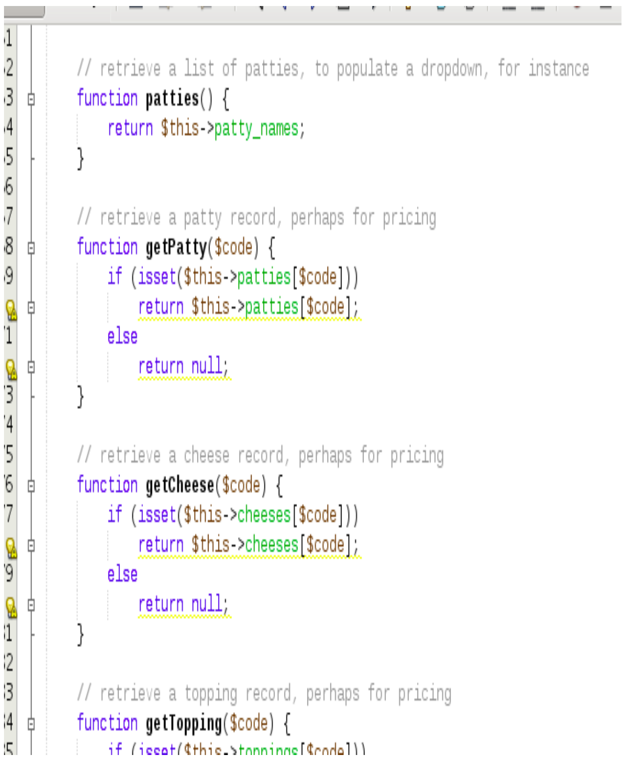
Sauces don't have a price, but one is included in the constructed objects for consistency with the other ingredients.



```
        }

        // build a full list of cheeses
        foreach ($this->xml->cheeses->cheese as $cheese) {
            $record = new stdClass();
            $record->code = (string) $cheese['code'];
            $record->name = (string) $cheese;
            $record->price = (float) $cheese['price'];
            $this->cheeses[$record->code] = $record;
        }
        // build a full list of toppings
        foreach ($this->xml->toppings->topping as $topping) {
            $record = new stdClass();
            $record->code = (string) $topping['code'];
            $record->name = (string) $topping;
            $record->price = (float) $topping['price'];
            $this->toppings[$record->code] = $record;
        }
        // build a full list of sauces
        foreach ($this->xml->sauces->sauce as $sauce) {
            $record = new stdClass();
            $record->code = (string) $sauce['code'];
            $record->name = (string) $sauce;
            $record->price = 0.0;   // for consistency
            $this->sauces[$record->code] = $record;
        }
    }
```

Menu Model Accessors

Accessors are provided to return the array of patty names, or to retrieve individual elements of the ingredient collections.



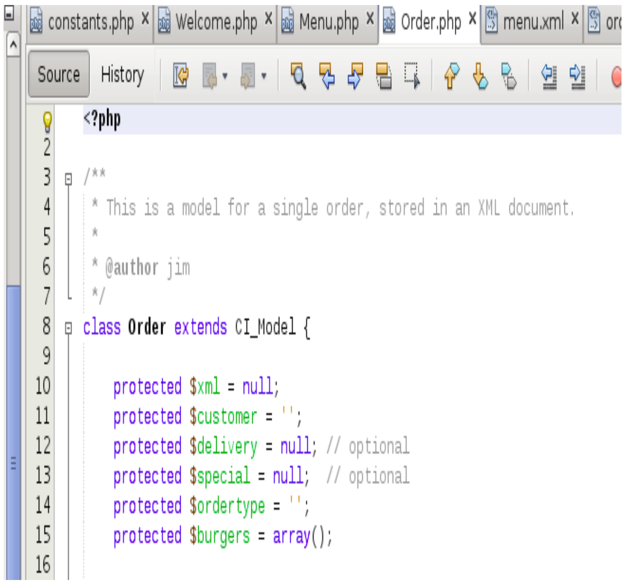
```
    // retrieve a list of patties, to populate a dropdown, for instance
    function patties() {
        return $this->patty_names;
    }

    // retrieve a patty record, perhaps for pricing
    function getPatty($code) {
        if (isset($this->patties[$code]))
            return $this->patties[$code];
        else
            return null;
    }

    // retrieve a cheese record, perhaps for pricing
    function getCheese($code) {
        if (isset($this->cheeses[$code]))
            return $this->cheeses[$code];
        else
            return null;
    }

    // retrieve a topping record, perhaps for pricing
    function getTopping($code) {
        if (isset($this->toppings[$code]))
```

Order Model

The `Order` model encapsulates a single order.

It has an `xml` property to hold the root element of the XML document, like the Menu model.

It has additional properties for order attributes, and then provides for a collection of burger objects that would make up that order.

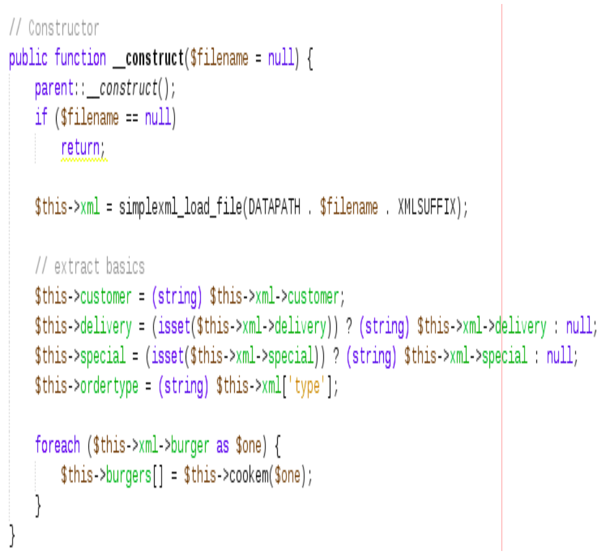


```
<?php

/**
 * This is a model for a single order, stored in an XML document.
 *
 * @author jim
 */
class Order extends CI_Model {

    protected $xml = null;
    protected $customer = '';
    protected $delivery = null; // optional
    protected $special = null;  // optional
    protected $ordertype = '';
    protected $burgers = array();

```

Order Model Constructor

Its constructor is similar to the Menu's, except that the order properties can just be extracted from the XML root element.

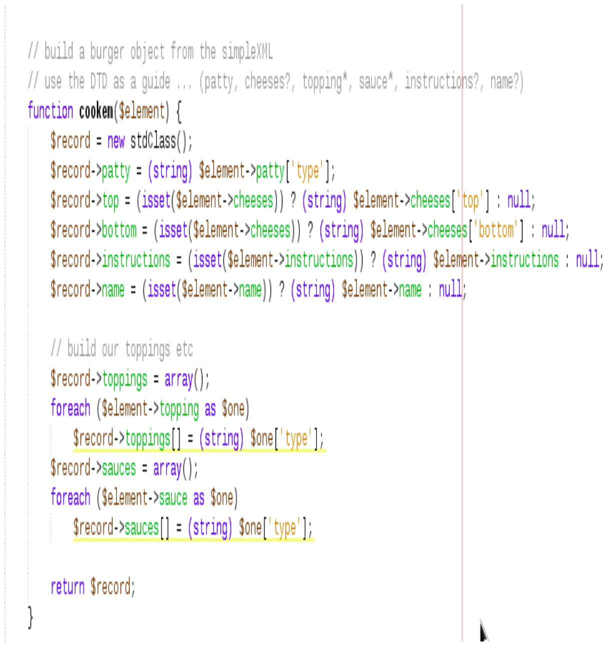
The array of burgers is created by iterating over the "burger" elements inside an order's XML.



```
// Constructor
public function __construct($filename = null) {
    parent::__construct();
    if ($filename == null)
        return;

    $this->xml = simplexml_load_file(DATAPATH . $filename . XMLSUFFIX);

    // extract basics
    $this->customer = (string) $this->xml->customer;
    $this->delivery = (isset($this->xml->delivery)) ? (string) $this->xml->delivery : null;
    $this->special = (isset($this->xml->special)) ? (string) $this->xml->special : null;
    $this->ordertype = (string) $this->xml['type'];

    foreach ($this->xml->burger as $one) {
        $this->burgers[] = $this->cookem($one);
    }
}
```

Order Burder Building

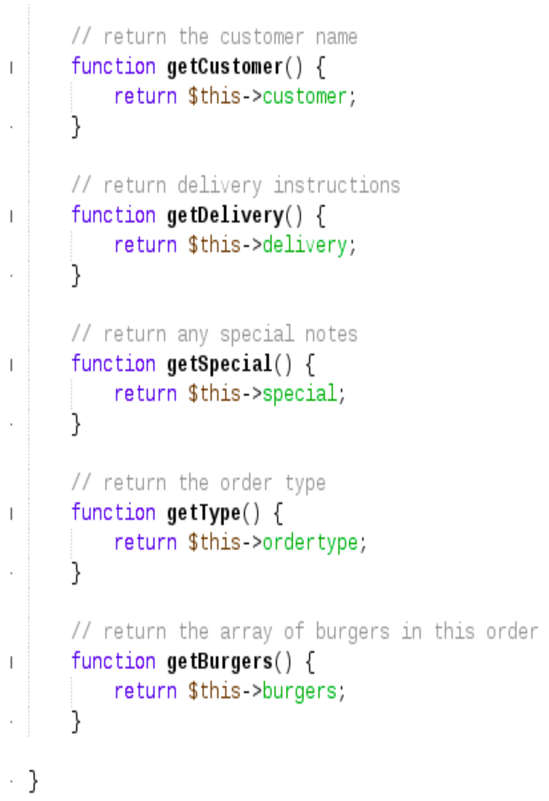
The `cookem` method constructs a burger object on the fly, with some iundividual properties and some that are collections (toppings & sauces).



```
// build a burger object from the simpleXML
// use the DTD as a guide ... (patty, cheeses?, topping*, sauce*, instructions?, name?)
function cookem($element) {
    $record = new stdClass();
    $record->patty = (string) $element->patty['type'];
    $record->top = (isset($element->cheeses)) ? (string) $element->cheeses['top'] : null;
    $record->bottom = (isset($element->cheeses)) ? (string) $element->cheeses['bottom'] : null;
    $record->instructions = (isset($element->instructions)) ? (string) $element->instructions : null;
    $record->name = (isset($element->name)) ? (string) $element->name : null;

    // build our toppings etc
    $record->toppings = array();
    foreach ($element->topping as $one)
        $record->toppings[] = (string) $one['type'];
    $record->sauces = array();
    foreach ($element->sauce as $one)
        $record->sauces[] = (string) $one['type'];

    return $record;
}
```

Order Model Accessors

Accessors are provided to expose an order's properties.

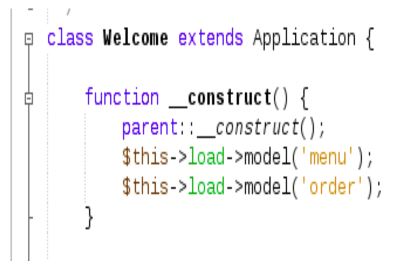


```
    // return the customer name
    function getCustomer() {
        return $this->customer;
    }

    // return delivery instructions
    function getDelivery() {
        return $this->delivery;
    }

    // return any special notes
    function getSpecial() {
        return $this->special;
    }

    // return the order type
    function getType() {
        return $this->ordertype;
    }

    // return the array of burgers in this order
    function getBurgers() {
        return $this->burgers;
    }

}
```

Welcome Controller

Nothing was autoloaded, so the `Welcome` controller's constructors loads the `Menu` and `Order` models.

The `Menu` model will be used inside the controller (`$this->menu`), but the `Order` model is only loaded to get the class definition.

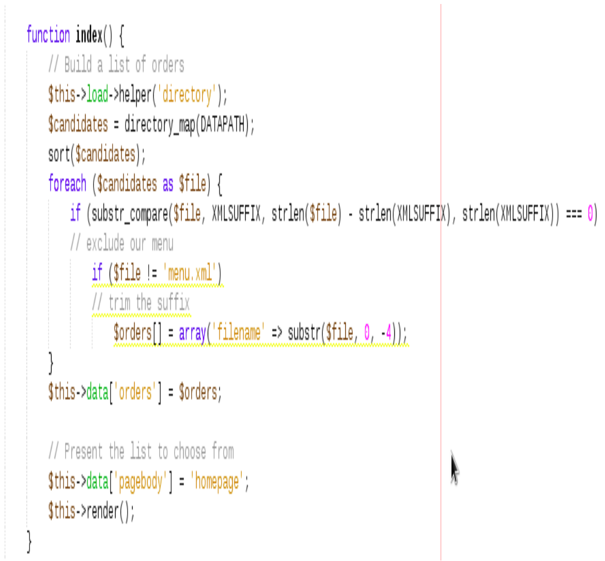


```
class Welcome extends Application {

    function __construct() {
        parent::__construct();
        $this->load->model('menu');
        $this->load->model('order');
    }
```

Welcome Index Method

The default homepage method locates and lists all of the actual "order" files, i.e. those in the `/data` folder, which have an `xml` suffix, and which are not the menu.

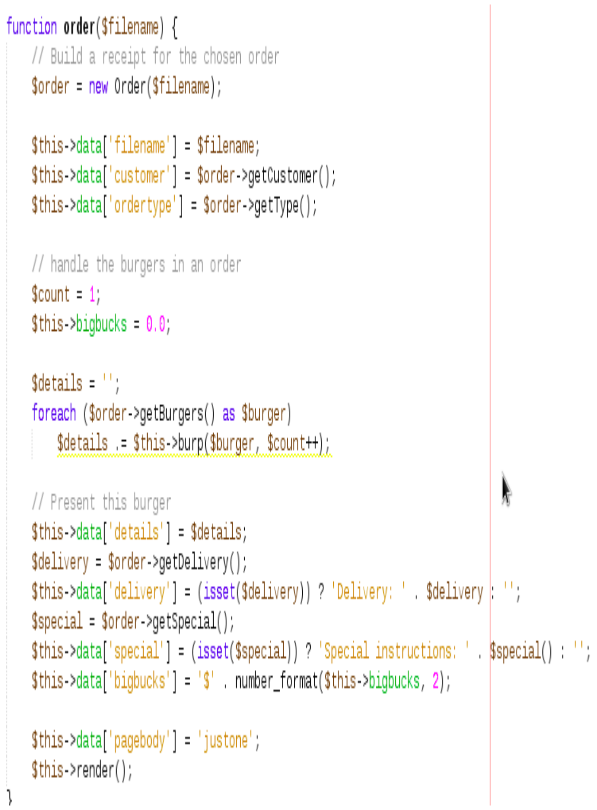
Each order file name is presented with a link to the `order` method of the Welcome controller.



```php
function index() {
    // Build a list of orders
    $this->load->helper('directory');
    $candidates = directory_map(DATAPATH);
    sort($candidates);
    foreach ($candidates as $file) {
        if (substr_compare($file, XMLSUFFIX, strlen($file) - strlen(XMLSUFFIX), strlen(XMLSUFFIX)) === 0)
        // exclude our menu
            if ($file != 'menu.xml')
            // trim the suffix
                $orders[] = array('filename' => substr($file, 0, -4));
    }
    $this->data['orders'] = $orders;

    // Present the list to choose from
    $this->data['pagebody'] = 'homepage';
    $this->render();
}
```

Welcome Order Method

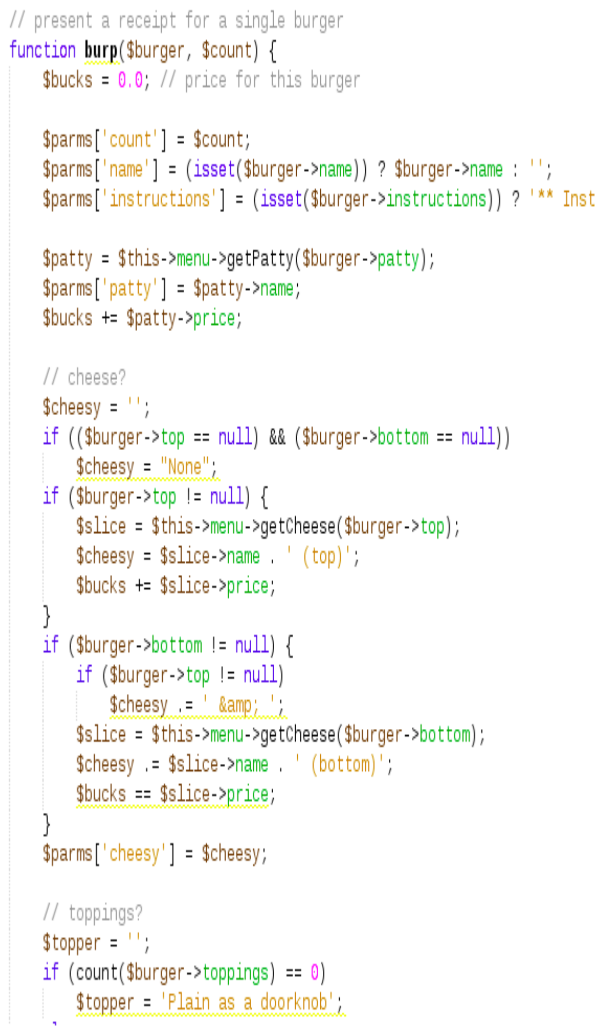
This method constrcuts an `Order` object from the supplied filename, and then builds view parameters from it.



```php
function order($filename) {
    // Build a receipt for the chosen order
    $order = new Order($filename);

    $this->data['filename'] = $filename;
    $this->data['customer'] = $order->getCustomer();
    $this->data['ordertype'] = $order->getType();

    // handle the burgers in an order
    $count = 1;
    $this->bigbucks = 0.0;

    $details = '';
    foreach ($order->getBurgers() as $burger)
        $details .= $this->burp($burger, $count++);

    // Present this burger
    $this->data['details'] = $details;
    $delivery = $order->getDelivery();
    $this->data['delivery'] = (isset($delivery)) ? 'Delivery: ' . $delivery : '';
    $special = $order->getSpecial();
    $this->data['special'] = (isset($special)) ? 'Special instructions: ' . $special() : '';
    $this->data['bigbucks'] = '$' . number_format($this->bigbucks, 2);

    $this->data['pagebody'] = 'justone';
    $this->render();
}
```

Welcome Burp Method

This method extracts the details of just the one burger, and formats this for presentation.



```php
// present a receipt for a single burger
function burp($burger, $count) {
    $bucks = 0.0; // price for this burger

    $parms['count'] = $count;
    $parms['name'] = (isset($burger->name)) ? $burger->name : '';
    $parms['instructions'] = (isset($burger->instructions)) ? '** Inst

    $patty = $this->menu->getPatty($burger->patty);
    $parms['patty'] = $patty->name;
    $bucks += $patty->price;

    // cheese?
    $cheesy = '';
    if (($burger->top == null) && ($burger->bottom == null))
        $cheesy = "None";
    if ($burger->top != null) {
        $slice = $this->menu->getCheese($burger->top);
        $cheesy = $slice->name . ' (top)';
        $bucks += $slice->price;
    }
    if ($burger->bottom != null) {
        if ($burger->top != null)
            $cheesy .= ' &amp; ';
        $slice = $this->menu->getCheese($burger->bottom);
        $cheesy .= $slice->name . ' (bottom)';
        $bucks == $slice->price;
    }
    $parms['cheesy'] = $cheesy;

    // toppings?
    $topper = '';
    if (count($burger->toppings) == 0)
        $topper = 'Plain as a doorknob';
```

Congratulations!

You have completed tutorial #xml-xml02: XML Processing (optional)

If you would take a minute to provide some feedback, we would appreciate it!

The next activity in sequence is: There is nothing further in this course

You can use your browser's back button to return to the page you were on before starting this activity, or you can jump directly to the course homepage, organizer, or reference page.