****

**Signboom Software Design Documentation**

Prepared by: Alison Taylor, Usable Web Designs

Document Version: Draft 1.1

Document Date: July 25, 2016

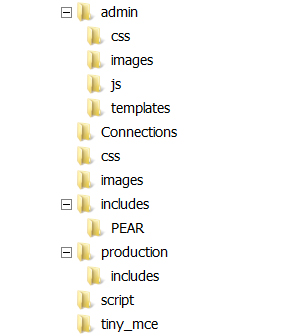
# Ordering System Architecture

## Technology Used

The ordering system is coded in:

* HTML/CSS for page layout
* PHP for database access and overall architecture
* Javascript for most costing calculations (with some minimal use of AJAX). NOTE: the costing calculations and all the database contents are exposed to a modestly savvy web user through Javascript code. This should be changed in future major revisions.
* The database is a mySQL database.

## File Organization

The ordering system code (which is based on a legacy framework) does **not** make use of a controller/template architecture. That is, most of those files have the PHP and HTML/CSS embedded in a single file.

The admin area, which is new code, uses a controller / template architecture. The production system does not.

The admin folder contains the admin system.

The production folder contains the production system.

TinyMCE is the “what you see is what you get” editor that is used in the admin system.

There are also .php files in the top level folder which are not showing in this diagram.

## Software Flow

When an order is placed, the function calls made are in the sequence shown below.

Notation used below:

Example **allorder.js:**ProcessUpload()

The function ProcessUpload() is invoked. The code for that function is in the **allorder.js** file.

**all\_order.php:** the Submit button on the order page invokes **allorder.js:**ProcessUpload() when clicked

**allorder.js:**ProcessUpload()

calls **allorder.js:**DoCompute()

calls **orderutil.js:**GetFreightCharges()

calls **orderutil.js:**completepost()

calls **allorder.js:**CalculateOrderCost() and

calls **allorder.js:**FinishUpload()

which populates line items tprod# and xprod# with tilda-separated values

calls **orderutil.js:**SetupStandardFields()

calls **allorder.js:**OpenPWWindow()

calls **orderutil.js:**StartStandardUpload()

which posts a form (including line items tprod# and xprod#) to **ordprep.php**

which explodes xprod# into an array to separate out the information in each line

and inserts any new shipping address into the shipto database table

and inserts general order information into the ordermast database table

and inserts exploded line item data into the linedetail database table

and sends email to orders@signboom (and appropriate team email) using mailord:bldmsg()

and calls **ordprep.php:**submitOrder()

calls **orderutil.js:**StartEditUpload()

calls **su.js:**StartUpload() which passes info to the file upload software on the FTP server

when the file upload software is done, it calls **orderthanks.php** on the web server

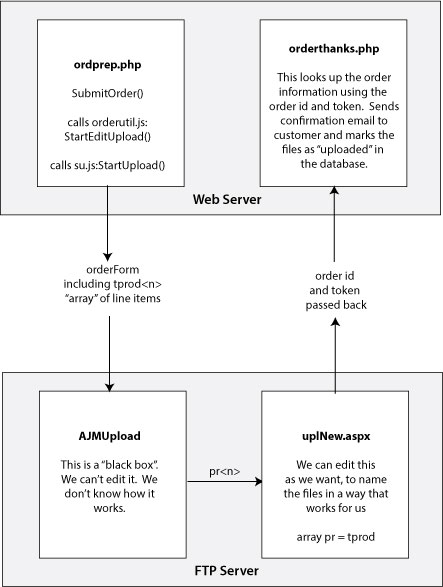
**orderthanks.php** calls **mailord.php:**bldhtml() or **mailord.php:**bldmsg() to send a confirmation email to the customer

## Uploading the Customer's Files to the Signboom FTP Server

The web software invokes AJMUpload uploading software to upload the customer's files to the Signboom FTP server. The AJMUpload software is a “black box”. We don't have the source software and don't know what is in it. I also don't have any documentation on the interface.

The AJMUpload file uploading software is provided by Marcel Madonna, whose email is: marcel\_madonna@ajmsoft.com

Marcel's upload software lives on Signboom's FTP server: upload.signboom.com



**How the Code Used to Work (Before we Introduced Fixtures)**

AJMUpload is invoked by the StartUpload() function in the file su.js which is located on the web server (www.signboom.com).

function StartUpload() {

var myserver = "upload.signboom.com";

var myport = "31164";

winOpts = "height=340,width=500,scrollbars=yes,location=no,toolbar=no,menubar=no,resizable=no,status=no";

//if (document.orderForm.Token) {

// Token = document.orderForm.Token.value;}

//else {

Token = (new Date()).getTime() % 1000000000;

//}

window.open("http://" + myserver + ":" + myport + "/progress.html?Token=" + Token, Token, winOpts);

document.orderForm.action = "http://" + myserver + ":" + myport + "/upload.html?Token=" + Token;

document.orderForm.submit();

}

AJMUpload listens on the port 31164. Since there may be multiple simultaneous uploads, the web software provides a token for each upload, so that AJMUpload can report back the upload progress to each customer; this reporting is done in the window that is opened using the winOpts parameters of the StartUpload() function shown above.

The web software passes the orderForm HTML form up to the AJMUpload software. That form contains **all** the information for the order: ie. everything we see on the order page.

AJMUpload uploads all the files from the customer's computer to the FTP server, and gives them temporary names. The files bypass the web server, never being stored there. The upload.html file is created on the fly by AJMUpload; it displays the upload progress to the customer.

Once the files are uploaded, AJMUpload invokes C:\Inetpub\wwwroot\sbupload\UplNew.aspx, on the ftp server, passing it information so that UplNew.aspx can give the uploaded files meaningful names which include the processing instructions for Signboom staff. (These processing instructions act as a backup to the information given in the Production software system.)

We are able to modify this ASP file and I have done so in the past. It used to have sections to handle the multiple different order types (banner, rigid, roll and finished product); I have simplified it down to handle the single order type (using old code from the banner handling). The pr array in this ASP code contains the same content as the tprod array in our javascript code. (AJMUpload.exe maps the one array to the other.)

When AJMUpload is done uploading the files, it invokes the orderthanks.php file on the webserver, passing in the order id and the token, so that orderthanks knows which order has just been uploaded. orderthanks then sends the customer an order confirmation email and marks that order in the database as having been successfully uploaded.

### How the Code Works Today (Now that we have Fixtures)

AJMUpload assumes that there is a file for every line in the order. This was true until we introduced fixtures. Now we have some lines that don't have files. Prior to August 4, 2013, we were getting errors where line item information was being mapped incorrectly to the files when fixtures were included in orders (except in cases where the fixtures were all in the *last line items* of the order).

We needed a way to pass in tprod1 to tprod10 lines that reflect only those line items with files. So we added xprod<n> elements to the orderForm, populated them with line item information for **all** lines - including those without files - and changed all the web server code use the xprod<n> elements. We then copied into the tprod<n> elements ***only line item information for lines which have files associated***, so that AJMUpload gets the information it needs, in the format it expects it.

### Where Files End up Being Stored on the File Server

When UplNew.aspx renames the files, it also moves them. Three folders are identified in it:

dim gblUploadDir As String = "C:\uploads\"

dim gblCompleteDir As String = "C:\Orders\"

dim gblPacklistDir as string = "C:\Orders\PackList\"

The “black box” AJMUPload code stores the uploaded files in gblUploadDir, so C:\uploads in this case. They are named with tokens such as 123456789.1, 123456789.2, and 123456789.ajmSumm. In this example two files were uploaded in one order (#.1 and #.2) and the information about those files was stored in #.ajmSumm.

Automate doesn't move them from upload... Marcel's software moved them to the Orders folder and from there automate takes over. I will have someone at the office drag the files over to the orders folder then automate will take over.

Sent using Leonard's Cell Phone

On Apr 14, 2017, at 3:09 AM, Alison @ Usable Web Designs <alison@usablewebdesigns.com> wrote:

Hi Len

The renamed files are in c:\uploads on the upload2 machine.

They’ve been there about 10 minutes now. I had expected Automate to move them…?

Alison

The new name is given to each file in the same line of code where it is moved (in function ProcessBanrFile which is now used for all types of media, not just banners):

fi.MoveTo(finew.FullName)

It is moved to gblComlpeteDir which is C:\Orders in this case.

Shipping documents are renamed and moved in the ProcessShippingDoc() function and end up in gblPackListDir which is C:\Orders\PackList in this case.

**Calendar Setting Required for File Server**

**From an email Marcel sent me in late 2014**

Alison, date conversions under .NET are locale specific. The program is not working correctly because the date format established on the machine doesn’t conform to “mm/dd/yyyy”. There are two ways to resolve this.

1. Open IIS Manager

2. Click on the Computer icon in the left-hand ‘Connections’ pane.

3. Double-Click on .NET Globalization

a. Select ‘English (United States) (en-US)’ in ‘Culture’ and ‘UI Culture’

b. Click ‘Apply’ in the Actions pane on the right hand side on the page

c. Restart the web server

Alternatively, you could change sbupload.aspx to use a specific format when converting date. It contains a statement that probably looks something like this:

Date dt = CDate(somedate)

Change it to

Date d = DateTime.ParseExact(somedate, "MM/dd/yyyy", System.Globalization.CultureInfo.InvariantCulture)

Retest when done.

Cheers

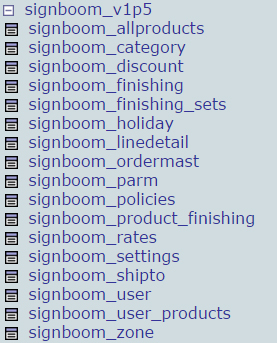
Marcel

## Data Flow

Section to be added

## Database Structure

These are the tables in the database.

****TO DO: Explain what each table holds.

allproducts

category

discount

finishing

finishing\_sets

holiday

linedetail

ordermats

parm

policies

product\_finishing

rates

settings shipto

user

zone

# Ordering System Cost Calculations

## Costing Calculations Overview

The price of an order consists of:

1. Customer discount (either volume-based or customer-specific: whichever is higher)
2. Line item cost of media printed on, cost of media wasted, and cost of ink
3. Line item cost of finishing options
4. Setup cost
5. Cost of rush/hot service
6. Freight cost
7. Taxes: gst and pst

### Customer Discount

Every product, whether media or a fixture, has a total price made up of two components:

total\_cost = cost\_discountable + cost\_nondiscountable

The discountable portion of this total cost can be discounted by the order form. The non-discountable portion cannot be.

The discountable portion of every line item is added to a discountable total for the order. Then a discount is applied to that total. The discount applied is the higher value of:

1. A fixed percentage discount Signboom has extended to that customer, or
2. A percentage discount based on the size (volume) of that particular order. (Note: If the percentage discount is calculated to be less than $5, we don’t discount, since paltry discounts are more likely to annoy clients than please them.)

Discountable and non-discountable costs are typically given per square foot of the media. But for fixtures they are per item.

The calculation of the volume discount (b) is:

percentage = discountable cost in dollars / 100

So if the customer’s order is $1000 (including media, setup, finishing, waste, and ink) their volume discount would be 10%. If their order was $3500, their volume discount would be 35%. But there is a cut-off: we don’t offer discounts of more than 50%.

The code checks to see if the volume discount would be better than the discount that has been extended to that customer’s account. It gives them the higher of the two discounts.

The discount does **not** get applied to the whole order cost. It only gets applied to the discountable portion and the finishing options costs.

### Line Item Cost of Media and Ink

The line item cost is calculated by the CalculateOrderCost() function in the file allorder.js which invokes CalculateLineCost() for each line item.

Below, we discuss the general approach to calculating costs. However, there is a special case, where the line item is a fixture. The special case is the simple formula below.

line\_cost = (cost\_discountable + cost\_nondiscountable) \* quantity

The general case for calculating line item cost is to add up these:

1. Cost of media printed on = (cost\_discountable + cost\_nondiscountable) \* (sqft\_printed)
2. Cost of media wasted = cost\_waste \* sqft\_wasted
3. Cost of ink used = cost\_ink \* sqft\_printed

### Line Item Cost of Finishing Options

The database table signboom\_finishing lists all the finishing options we offer, and indicates which calculation to use to price the option and what numbers to use in that calculation.

A finishing option is a type of finishing which can be applied to a particular type of media (rigid, banner or adhesive). Some examples are shown below for rigid media. Each finishing option has a code. The codes of these examples are given in brackets.

Here are three different options for grommets:

1. applying grommets to top corners (RH-2G)
2. applying grommets to top and bottom corners (RH-4G)

Here are three options for double-sided printing:

1. print both sides with the same file on each side (RB-SF)
2. print both sides with different files on each side (RB-DF)
3. don’t print the back (RB-X)

Here are several options for laminate which can be applied:

1. dry erase laminate (RL-DE)
2. anti-graffiti laminate (RL-AG)
3. premium gloss laminate (RL-PG)

*Understanding the option codes (using RH-2G as an example):*

* RH-2G: These particular example options can only be used in rigid media. That’s indicated by the “R” at the start of their code. (Options that can be used with banners start with “B”. Options that can be used with adhesive media start with “A”.)
* RH-2G: The combination of the first and second letter in the option code indicates a group of mutually exclusive finishing options. For example, the “H” here stands for “hanging”. For a single item, you can choose only one RH option (say RH-2G or RH-4G) **but not both**. Two options which have a *different* second letter can be used together. For example, you **can** use RH-4G and RB-DF and RL-DE together. Here are all the pairs. (The mapping between the two letter code and the name is done in the table signboom\_finishing\_sets.)
* AF (Cutting)
* AL (Lamination)
* AI (Ink Layers)
* BF (Cutting)
* BB (Back Side)
* BI (Ink Layers)
* RF (Cutting)
* RL (Lamination)
* RB (Back Side)
* RH (Hanging)
* RE (Edges)
* RI (Ink Layers)
* RO (Orientation)

For legacy reasons, “Cutting” ended up using the letter “F” (for finishing) instead of “C” (for cutting).

* RH-2G: The last two letters are (as much as possible) an acronym for the option itself. For example, RH-4S (for four standoffs), RH-4H (for 4 holes, at top and bottom corners) and RH-2H (for 2 holes at top corners) and RH-X (for no hanging).
* RH-X: Every group of mutually exclusive finishing options has an –X option (e.g. RH-X) which means no finishing of this type (e.g. “hanging) is required.

*Which finishing options are available for each product:*

The table signboom\_product\_finishing lists all the finishing options which are permitted for each product. For example, the PAV (perforated adhesive vinyl) product has only four possible finishing options:

1. AF-HC: hand-cut to PDF dimensions
2. AF-X: no cutting, leave on roll
3. AL-OC: optically clear laminate
4. AL-X: no lamination

*Calculating the cost of a finishing option:*

We mentioned earlier that the signboom\_finishing option indicates what equation to use to calculate the cost of finishing and what numbers to use in that equation. The columns in the table which hold that information are:

* **Units:** which equation to use
* **Fixed:** one of the numbers to use in the equation, usually a minimum starting cost for that kind of finishing
* **Variable:** the second number to use in the equation, usually multipled by linear feet, square feet or quantity

There are currently just four different equations for calculating the cost of finishing options:

**Perimeter Footage (PF):** Used when the cost of finishing depends perimeter length of the sign. This equation is used for polished edges, rod pockets, hemming, pigtails, grommeting (x grommets per foot), and various types of cutting.

**Square Footage (SF):** Used when the cost of finishing depends on the size of the sign. This equation is used for lamination, multiple ink layers, through cut to shape, kiss cut to shape and required orientation relative to media.

**Each (EA):** Used when the cost of finishing does not depend on the size of the sign. Examples of this are grommets in two or four corners only, holes in two or four corners only, a particular number of standoffs (4, 6, 8)

**Back Side (BS):** Used for double-sided printing only, to capture the cost of the ink and the effort to print the back side.

Below is the actual code used for each equation.

* varcost is the variable number in the equation. fixedcost is the fixed number in the equation.
* quantity is the number of that particular sign to be made.
* width and height are the measurements of the printed sign in inches.
* perimeter is the length of the perimeter of the printed sign.
* Cost\_discountable, cost\_nondiscountable and cost\_ink are calculated as explained in earlier in this section.

case "PF": // Perimeter Footage

option\_cost = ((perimeter \* varcost) + fixedcost) \* quantity;

cost\_options += option\_cost;

break;

case "SF": // Square Footage

option\_cost = ((width/12 \* height/12 \* varcost) + fixedcost) \* quantity;

cost\_options += option\_cost;

break;

case "EA": // Each

// Calculate the number of (fixed) and the cost of (variable) an item added to a sign.

option\_cost = fixedcost \* varcost \* quantity;

cost\_options += option\_cost;

break;

case "BS": // Back Side

// Charge the printing and ink cost for the back side, but not the media cost.

// Don't multiply cost\_disc + cost\_non + cost\_ink by quantity, as they already

// include quantity in them.

option\_cost = ((cost\_discountable + cost\_nondiscountable) \* varcost) +

(fixedcost \* quantity) + cost\_ink;

cost\_options += option\_cost;

printed\_sqfootage = printed\_sqfootage \* 2.0;

break;

### Setup cost

This section not complete.

### Cost of rush/hot service

This section not complete.

### Freight cost

This section not complete.

### Taxes: gst and pst

This section not complete.

# Admin System

## Technology and File Organization

This code was redesigned when ordering system V1.5 came online. It is built using a controller-template model, which is well described in the ook “Build Your Own Database Driven Web Site” from Sitepoint.com.

PHP code is in the controller files (xxx.php) in the top level folder. The HTML code is in the template files in the templates folder. Javascript is in the js folder. Where possible several PHP files make use of a single template.

## Maintaining Customers, Products and Finishing Options

The admin system lets staff add, edit and delete various types of information from the database:

* Customers (and staff user accounts)
* Products
* Finishing options
* Holidays
* “Fudge” factors for costing, shipping (along with the cutoff time)

With products, the user can:

* Choose from a list of products (select-product.php and templates/ select-product.php)
* Add a new product (create-product.php and templates/edit-product.php)
* Edit an existing product (edit-product.php and templates/edit-product.php)
* Disable an existing product (done in the **Edit Product** page)
* Associate finishing options with products (edit-product-finishing.php and templates/edit-product-finishing.php)

Note: create-product.php and edit-product.php are very similar at the controller level (they could - and probably should - be combined into one) and they use the same template.

With finishing options, the user can:

* Choose from a list of finishing options (select-finishing-option.php and templates/select-finishing-option
* .php)
* Add a new finishing option (create-finishing-option.php and templates/edit-finishing-option.php)
* Edit an existing finishing option (edit-finishing-option.php and templates/edit-finishing-option.php)

Note: create-finishing-option.php and edit-finishing-option.php are very similar at the controller level (they could - and probably should - be combined into one) and they use the same template.

With customers, the user can:

* Search for a customer (or staff account) by account name or email address (edit-customer.php and templates/select-customer.php)
* Edit an existing customer (edit-customer.php and templates/edit-customer.php)
* Disable an existing customer (done in the **Edit Customer** page)

Note: There is one controller edit-customer.php which calls both templates (edit-customer.php and select-customer.php). Adding a new account is done via the main website **Create Account** (signup.php) page. Once an account has been created, the admin system can be used to give that customer staff privileges.

## How Finishing Options Work



Finishing options are organized into the **sets** shown in the image to the right. Those that start with A are for adhesive media. B’s are for banner media and R’s are for rigid media. The figure here shows the contents of the signboom\_finishing\_sets table at the time of writing.

The full list of finishing **options** is stored in the signboom\_finishing table. The OptionSet column associates each option with a particular finishing option **set**.

Ideally, the value in the OptionSet column would be the value from the **Code** column in the signboom\_finishing\_sets table. Instead, the OptionSet column is populated with a capitalized version of the **Name** column from the signboom\_finishing\_sets table.

e.g. OptionSet = LAMINATION Name = Lamination

At some point, I may get around to changing this.

In addition, there is one other less-than ideal mapping. For legacy reasons

OptionSet=FINISHING is used instead of OptionSet=CUTTING (Name = Cutting)

When I change the big above, this will get fixed as well. So, for now, the complete mapping is:

* Cutting <-> FINISHING
* Lamination <->LAMINATION
* Back Side <-> BACK SIDE
* Ink Layers <-> INK LAYERS
* Hanging <-> HANGING
* Edges <-> EDGES
* Orientation <->ORIENTATION

The list of valid option types are currently hardcoded into templates/edit-finishing-option.php. Eventually those should be stored in the database and read out of there.

# Drag and Drop Order System

## File Uploading

The Javascript code below (from filedrag.js) communicates with a new version of AJMUpload which Marcel Madonna created. It supports cross-domain scripting via AJAX with CORS.

function upload()

{

// Display message that warns customer to stay on page till order is finished.

DisplayUploadingMessage();

// Prepare information which will be submitted via AJAX call.

var formData = new FormData();

for (var j = 0; j < file\_lists.length; j++)

{

for (var i = 0; i < file\_lists[j].length; i++)

{

var ffile = file\_lists[j][i];

//alert('Uploading file: ' + ffile.name);

var new\_file\_name = 'NEW-' + ffile.name;

formData.append('file-' + i + '-' + j, ffile, new\_file\_name);

}

}

// Create a unique token based on timestamp so we know which AJAX response goes with which request.

Token = (new Date()).getTime() % 1000000000;

// Create AJAX request object.

var client = new XMLHttpRequest();

// Define and register a handler to check the XHR instance's status when receiving an AJAX response.

client.onreadystatechange = function ()

{

//alert("ready state = " + client.readyState + " status = " + client.status);

if ((client.readyState == 4) && (client.status == 200))

{

//alert("statusText = " + client.statusText);

ClearList();

file\_lists = [];

HideUploadingMessage(); // so customer knows they can now leave the page

// Make an AJAX call to uplxml.aspx, which gives the files their desired names

// once they have been uploaded.

// In the regular Signboom order system, uplNew.aspx works out the desired names,

// based on the tprod array which is passed in to it.

// Here in the drag and drop order system, the Javascript code works out the

// desired name (var new\_file\_name) and passes it down as the 3rd parameter in

// the formData.append() call when the files are uploaded.

var clfinish = new XMLHttpRequest();

clfinish.open("get", "http://sb.uplhome.com/uplxml.aspx?Token=" + Token, false);

clfinish.send();

}

}

// Make an AJAX call to upload the files. The formData information gets stored

// in the .ajmSumm files, where it is available to the uplxml.aspx code so the

// files can be given the desired names.

client.open("post", "http://sb.uplhome.com:31444/xmlreq.html?Token=" + Token, true);

client.send(formData); // send formData to the server using XHR

}

The formData keeps track of token-filename pairs for each file uploaded. The token is a unique value created based on the timestamp. The filename is determined by Javascript based on the original filename the customer provided (which includes information about due date, dimensions, finishing options, hardware required and a reference name) and the product drop-box the file was dragged into.

The AJAX call to http://sb.uplhome.com:31444/xmlreq.html triggers the upload of the files. A service listening on port 31444 uploads the files and gives them names like <token>.1 and <token>.2. Information about the files (such as the token-filename pairs) is stored in the file <token>.ajmSumm (short for AJM Summary).

When a status 200 message is returned, the upload associated with that token is complete. A subsequent AJAX call to http://sb.uplhome.com/uplxml.aspx causes the files associated with that token to be renamed using the token-filename pairs stored in the .ajmSumm file.

# Production System

## Technology…

Section to be added