Using Formulize and Pageworks to Create Custom Applications

A Conceptual Overview and API Documentation

August 28, 2012

Table of Contents

Introduction

This document explains some of the different scenarios for using Freeform Solutions' Formulize module, and the legacy Pageworks module, as well as custom PHP code using the Formulize API. Formulize and Pageworks are tools for creating data-entry applications that model business processes, and they provide easy access to the recorded data for reporting and analysis purposes.

Wherever there is a reference to Pageworks, you could also just use a plain PHP file. The only significant difference, is that Pageworks has already included the mainfile.php for you, and has included some other Formulize files to help make things work. But in general, everything you can do in Pageworks, you can do in plain PHP too, as long as you include the necessary files along with your code.

This document is a collection of concepts, examples and reference material that we hope provides a basic understanding of the tools available. We encourage you to post questions and feedback to the support forums at http://www.freeformsolutions.ca/formulize.

Creating applications involves two types of work. First there is setup and configuration of forms and relationships of forms. Second, you can use the screen system in Formulize to customize how things are presented to users. See the Formulize Getting Started Guide and tutorial videos for more information.

However, in some cases that's not enough, you might need some simple PHP programming. That's where this API reference comes in.

About Storing HTML in Forms

Formulize converts all HTML characters to their "htmlSpecialChars" equivalents, as a security precaution. This means that if you type HTML into a textbox, you cannot then get that HTML via the getData function and just output it to the screen. You can convert the HTML special characters to their regular equivalents using the XOOPS text sanitizer class, or the PHP function html_entity_decode. There is also an option on each Pageworks page to allow HTML characters from the DB onto the screen. If you select this option, Pageworks will do the conversion for you.

PART 1 – Creating Custom Pages with Pageworks or PHP

Use screens!!

You need a really good reason to do anything that is discussed here. Screens in Formulize will do 95% of what you need, hopefully more. But once in a while, you might need to dip into code. If so, the info here will be relevant.

Overview

Formulize is the form creation module and handles all interactions with forms and with data. Pageworks, on the other hand, is the optional interface building tool that gives you extra control over what users see on the screen and how they can interact with what they see. Creating an application is a matter of creating the necessary forms in Formulize, to act as data-entry screens and data containers. After that, the "Form Menu" block in Formulize may provide all the basic interaction with the forms that you need. Or you might use a custom menu module, like iMenu, to create the navigation you want to use.

If for some reason you can't make a Formulize screen that suits your needs, then you can always go to Pageworks, or PHP directly, and create whatever specialized interfaces you require to either those forms or to the data that has been submitted through them, or both.

Pageworks is a very simple module really. All it allows you to do is create individual pages for your website, that are accessible through the following URL structure:

http://www.yoursite.com/modules/pageworks/index.php?page=34

That address will display page number 34. Pageworks lets you specify custom PHP code that should be executed when page 34 is accessed, thereby creating a page for the visitor to see.

Remember, Pageworks is just a container for PHP code, but if you want to do it all manually, you certainly can. For example, here's a basic PHP page that you could put in the root of your website, in the same location as the mainfile.php:

```
include "mainfile.php"; // bootstrap your website
include "header.php"; // invoke the page template

print "Hello World"; // put any Formulize API calls here to show forms,
etc.

include "footer.php"; // finish the page and dump it all, including the
template, to the browser
```

Linking to Pages, Forms and Entries

The normal way to integrate Pageworks pages into the navigation scheme for your site, is to use a custom menu module. We like iMenu and have included an updated release of that module along with Formulize and Pageworks.

Each form is also uniquely addressable through the URL, and even each entry in each form. This URL displays entry 132 from form 5. The "ve" parameter is optional; leaving it out displays a blank form:

http://www.yoursite.com/modules/formulize/index.php?fid=5&ve=132

Creating a page in Pageworks

For details about the various functions used in these examples, consult the reference sections of this document.

A Basic Page

Fundamentally, Pageworks is a container for PHP code that gets executed when the page is viewed. Here's how to make a very simple page in Pageworks:

1. Add a new page in Pageworks. Give it a name and title. Click Save.
2. In the Template Box, type this:
   ```
   print "Hello World";
   ```
3. Click the Save button. Then on the main administration page for Pageworks, click the Permissions link at the top, and grant access to your new page, for the appropriate groups.

This page will simply display the words "Hello World" on the screen.

Displaying a Form

Suppose you want to have an activity log form, and you want users to be able to see a list of the entries, and the activity log form on the same page. If users click on an activity log entry in the list, then the form will reload with the details for that entry. This kind of interface might be useful if you don't want to present the full blown "list of entries" screen to the users.

First, let's assume you've created the form already, and it's form number 12 in your site. To display the form, do this:

1. Add a new page in Pageworks. Give it a name and title. Click Save.
2. In the Template Box, type this:

   ```
   displayForm("12");
   ```

3. Click the Save button. Then on the main administration page for Pageworks, click the Permissions link at the top, and grant access to your new page, for the appropriate groups.

That will display the form on the screen, just the same as if someone went to the form via the Form Menu (note: you will need to have setup permissions within Formulize for accessing the form itself).

Next, we need to get a list of entries in the form. Change the contents of the Template Box on your Pageworks page to this:

```
// getData is in the /modules/formulize/include/extract.php file
$data = getData("", "12");  // get all the entries in form 12

// getCurrentURL is in /modules/formulize/include/functions.php
$currentURL = getCurrentURL();
// "Activity Log" is the title of the form
// "activity_name" used in the display function is the data handle
// for an element in the form, as specified when that element was
// created
foreach($data as $entry) {
      $ids = internalRecordIds($entry, "Activity Log");
      $id = $ids[0];
      print "<p><a href=" . $currentURL . "&entry=$id>" .
display($entry, "activity_name") . "</a></p>";
}

displayForm("12");
```

This page will now display a list of all the activities that have been entered in the form, and each one will be a clickable link. The destination of the link will be the current page, with an extra parameter in the URL: entry=$id, where the $id is the ID number of that activity's entry in the form.

Lastly, we need to make the page aware of the 'entry' parameter and cause it to display that entry in the form. Change the line:

```
displayForm("12");
```

To this:

```
displayForm("12", $_GET['entry']);
```

Now, if there is a value for 'entry' in the URL, it will be used to display that entry in the form.

Special Variables Passed By Forms

There are some special flags passed by Forms in the $_POST array which are useful when creating advanced applications in Pageworks. A couple of the most important are:

`$_POST['form_submitted']` – this is true if the user has just clicked the Save button. It is not present otherwise.

`$_POST['lastentry']` – this is present if the user has just clicked the All Done button, and will contain the ID number of the last entry that was saved, if any. It is not present at any other time.

Displaying Pageworks Pages in Blocks

You can display a Pageworks page inside a block, if you want it to appear on a specific part of the page, or a specific kind of page in your site.

To do this, make a custom PHP block, and use this code for the contents of the block:

```
$page = 5;
include XOOPS_ROOT_PATH . "/modules/pageworks/index.php";
```

That will cause page number 5 to appear in that block.

You can also refer to a Pageworks page via a URL, if you need to include a pageworks page via some "web services" style system. The following URL will return the HTML output for the Pageworks page, without the template surrounding it (the "block=1" part is the key feature):

http://www.yoursite.com/modules/pageworks/index.php?page=5&block=1

PART 2 – API Reference

Functions Summary

There are several different functions available for use with Formulize. These functions let you do many things with forms and data, including display forms or parts of forms, display the standard list of entries page, display calendars based on the data in a certain form or Framework, or even extract data from a certain form or Framework.

In these docs, 'Framework' is used instead of 'Relationship' when referring to sets of forms and their data.

There is also a data handling class which you can read more about in the PDF called *Inside Formulize: A Developer's Guide*. It has many useful methods for getting data from forms more precisely than with the getData function described below.

Functions for displaying forms, parts of forms, or entries in forms:

- **displayForm** – display a form, or only certain elements in the form
- **displayFormPages** – display a form with a multi-page layout including skip logic
- **displayEntries** – display the standard "list of entries" page with all its features
- **displayGrid** – display a grid of form elements with custom headings
- **displayCalendar** (and displayFilter) – display a calendar view of entries in a form

Functions for displaying form elements, or buttons that will modify values when clicked:

- **displayElement** – display a specific element (including with a value from an entry)
- **displayCaption** – display the caption for a form element
- **displayDescription** – display the descriptive text for a form element
- **htmlForElement** – display the markup for an element, without a name attached
- **displayFilter** – display a drop down list with options based on an element
- **displayElementSave** – display a save button
- **displayButton** – display a custom button that will change an entry's value if clicked

Functions for reading and writing in the database, and parsing that data:

- **formulize_getCalcs** – get the calculations that are part of a saved view
- **formulize_writeEntry** – write values to an entry in the database
- **getData** – gather a dataset from the database using search terms and filters
- **internalRecordIds** – get the ID numbers of certain entries within a dataset
- **resultSort** – sort a dataset
- **resultSortRelevance** – sort a dataset based on the prevalence of search terms

Functions for displaying data that has been extracted from the database:

- **display** – return a value from one element in one form in a dataset
- **displayPara** – return values from textboxes as HTML paragraphs
- **displayBR** – return values from texboxes as HTML with <br> tags
- **displayList** – return values from textboxes as an HTML list
- **displayTogether** – return values with custom HTML joining each one

displayForm

The Scoop

This function is the basis for how forms are displayed. You should use Form screens for this purpose. If you need to do something that is not yet configuration option in Form screens, but is available in the API, then let's work together to add that capability to Form screens!

Syntax

```
displayForm($formframe, $entry, $mainform, $done_dest,
$button_text, $settings, $titleOverride, $overrideValue,
$overrideMulti, $overrideSubMulti, $viewallforms)
```

This function draws a form on the screen. When someone fills in the form and saves the data, the form action reloads the current URL and saves any data that is posted from the elements in the form. The various parameters are below.

Examples

Display the form with ID number 4 on the screen:

```
displayForm(4);
```

Display the entry with ID number 245 in form 4 for editing:

```
displayForm(4, 245);
```

Display a Framework of forms on the screen, using the "profile form" as the main form of the Framework (ie: it is the first form drawn, with other forms included based on their relationship to the main form):

```
displayForm(99, "", "profile form");
```

Display the entry with ID 188 in the profile form from Framework 99, for editing:

```
displayForm(99, 188, "profile form");
```

Display a form, and set it so that when the user clicks "All Done" they get taken to the front page of your XOOPS site:

```
displayForm(17, "", "", XOOPS_URL);
```

Display a form, and set it so that the "All Done" button reads "Back to the list":

```
displayForm(17, "", "", "", "Back to the list");
```

Display a form, and set it so that the "All Done" button reads "Back to the list", and when the user clicks that button, they will go to a pageworks page number 33:

```
displayForm(17, "", "", XOOPS_URL . "/modules/pageworks/index.php?page=33",
"Back to the list");
```

Display a form with no "All Done" button at all (only a Save button will appear at the bottom):

```
displayForm(17, "", "", "", "{NOBUTTON}");
```

Display a form and force the default value of a selectbox to be "East Region":

```
displayForm(63, "", "", "", "", "", "", "East Region");
```

Display only certain elements in form 63, based on the ID numbers of those elements:

```
$ff['formframe'] = 63;
$ff['elements'] = array(87, 88, 89, 90);
displayForm($ff);
```

Parameters

`$formframe` `- number, or array - required`

This is either the ID number of the form to draw, or the ID number of the Framework you want to access. It is required and cannot be omitted.

If it is an array, it must have two keys: 'formframe' which corresponds to the regular $formframe variable above, and 'elements' which is itself an array of element IDs that should be displayed. Only those element IDs will be displayed, which allows you to show only part of a form at once.

`$entry` `- number - optional`

This is the ID number of a specific entry which you want displayed in the form so the visitor can update that entry. By default this is empty.

`$mainform` `- number - optional`

This is the ID number of the form. By default this is empty. It is only necessary to specify this if `$formframe` is a Framework ID.

`$done_dest` `- string - optional`

All forms have two buttons at the bottom. Save, and a Done button. When the user clicks Save, the form stores their information and reappears with either the information they have just entered, or in some cases with a blank form so they can fill in another entry. The Done button is meant to signify that the user is finished entering or updating their data and wants to go back to wherever they were before.

To accommodate the fact that the proper destination for the Done button could be just about anywhere, you can specify an URL with this string. If no URL is specified, then the current URL is used. By default this string is empty.

`$button_text` `- string or array - optional`

If specified, this string will be used instead of the default text for the Done button (which is "All Done"). By default this string is empty. You can pass the text "{NOBUTTON}" to this parameter, in order to remove the All Done button from the form.

If this is an array, then the first value (key 0) should be a string that will be used instead of the default text for the Done button. The second value (key 1) should be a string that will be used instead of the default text for the Save button.

If both buttons have the value "{NOBUTTON}" then the bottom row of the form will be omitted. This may be desirable if you are creating some other mechanism to submit the form in your Pageworks page.

`$settings` `- array or string - optional`

This is a special array used to pass information between various other functions and displayForm. By default this array is empty and you should specify it as "" if you need to use any of the subsequent parameters.

There is one special use for this parameter which you may wish to use in certain special cases. If you are calling displayForm in a custom Pageworks, but do not want the form to display again after the user clicks the Save button, then set this parameter to "{RETURNAFTERSAVE}". This will cause the data to be saved when the user clicks the Save button, but then the displayForm function will immediately return and you can continue with other logic in your Pageworks page.

`$titleOverride` `- 1, 0 or "all" - optional`

The "all" setting causes all the normal heading information in the form to be stripped out ("About this entry…", etc). Can be useful in conjunction with the $formframe['elements'] option described above.

When displaying a framework of forms, containing forms that are in a one-to-one relationship with the unified display option turned on, then these forms will be drawn on screen as if they are one form: by default, the second and subsequent forms will be draw without title bars. However, if this parameter is set to 1, then the second and subsequent forms will be draw with their title bars, identifying which form they are. By default this is set to zero, or empty.

`$overrideValue` `- string or array - optional`

If there is a selectbox in the form, then you can set a default selection for that selectbox, other than the first value in the list (which is what is selected by default). To do this, specify a string, or an array of strings here. If there is a date box or boxes in the form, then you can specify a date to be used as the default (applies to all the date boxes). An array can contain mixed overrides, ie: one value could be for a selectbox, and another would be for a date, the system would sort them out (based on date overrides having to be in YYYY-mm-dd format).

Example:

```
$overrides[0] = "East Region";
$overrides[1] = "2005-09-16";
displayForm(63, "", "", "", "", "", "", $overrides);
```

This parameter only has an effect if no entry is being displayed, ie: it only affects the drawing of blank forms for new entries. By default, this is empty.

`$overrideMulti` `- 1 or 0 - optional`

All forms have a particular setting regarding how many entries a user is allowed to have in the form. That setting can be either one per group, one per user or more than one per user. In the case of forms which are set at "more than one per user" then if there is no `$entry` specified, ie: the user is entering new data, then the form redraws blank after the user saves their data. This is so the user can carry on entering more entries with more data. However in some cases you may wish the form to redisplay the entry the user just made, even though the form allows the user more than one entry. To override the default behaviour and redisplay the entry the user just made, set this to 1.

Alternatively, if you have a form that is set for only one entry per user, in some cases you would rather the form redraw blank. Usually this is when anonymous users are working with the form. Setting this to 1 will also override the default "single-entry" form behaviour so they behave like multi-entry forms.

By default this is set to zero, or empty.

`$overrideSubMulti` `- 1 or 0 - optional`

Just like overrideMulti, but used to override the default multi-entry form behaviour for subforms that are part of a framework that is being displayed.

`$viewallforms` `- 1 or 0 - optional`

This is an override setting, which is used to force the display for all forms in a Framework, regardless of the user's permissions to see them.

displayFormPages

The Scoop

This function is the basis for how multipage forms are displayed. You should use Multipage Form screens for this purpose. If you need to do something that is not yet configuration option in Multipage Form screens, but is available in the API, then let's work together to add that capability to Multipage Form screens!

Syntax

```
displayFormPages($formframe, $entry, $mainform, $pages,
$conditions, $introtext, $thankstext, $done_dest, $button_text,
$settings, $overrideValue)
```

This function is used to display a form as a series of pages instead of as one long form. Each page has certain elements from the form on it, and each page can also have certain conditions attached to it, controlling whether it is displayed or not. The conditions are based on the answers to questions in the form. This allows you to create "skip-logic" typically found in surveys, where certain questions appear or don't appear depending on what the user answered to a previous question.

The syntax for this function's parameters is rather complicated, because of the amount of information you need to specify in order to control the behaviour of a multi-page form. It consists mostly of a series of arrays, including serialized arrays of arrays. There is an example below, with detailed explanations in the Parameters section following that.

Example

This is a two page form, where the second page is displayed only if the answer to element 147 on page 1 is "Yes".

```
$formframe = 22; // use form number 22 - not a framework
$entry = ""; // show the form blank so a new entry can be saved
$mainform = ""; // no mainform since we're not using a framework

// create an array that describes which elements appear on which pages.
// Note: this array just controls which elements appear on the page, while
// the order that the elements appear in is still determined by the order
// specified in the Formulize admin pages.
```

```
$pages[1][] = 146;
$pages[1][] = 147;
$pages[1][] = 148;
$pages[1][] = 149;
$pages[2][] = 150;
$pages[2][] = 151;
$pages[2][] = 152;

// setup a condition to control the display of page 2.
// Conditions are made up of an array containing one or more sets of
// elements, operators and terms that should be compared against the
// answers in this form entry so far.
$element[] = "147";
$op[] = "=";
$term[] = "Yes";
$conditions[2] = array($element, $op, $term);

displayFormPages($formframe, $entry, $mainform, $pages, $conditions);
```

Parameters

$formframe - number - required

This is either the ID number of the form to draw, or the ID number of the Framework you want to access. It is required and cannot be omitted.

$entry - number - optional

This is the ID number of a specific entry which you want displayed in the form so the visitor can update that entry. By default this is empty.

```
$formframe, $entry, $mainform, $pages, $conditions, $introtext, $thankstext,
$done_dest, $button_text, $settings, $overrideValue
```

$mainform - number - optional

This is the ID number of the form. By default this is empty. It is only necessary to specify this if `$formframe` is a Framework name or ID.

$pages – array

This is a multidimensional array that describes which elements should appear on which pages. The first key is the page number that the element should appear on. The value for that key is another array containing the element IDs of all the elements that should appear on the page. The order of the elements doesn't matter; their visible order on the screen is controlled by their order setting in the Formulize admin screen. You can have an unlimited number of pages in a form.

You can also specify pages in your multi-page form that don't contain any elements. You may want a page of text to appear at a certain point and then more form pages to appear afterwards. You can do this by using the string "HTML" or the string "PHP" as the first "element" on a page. The second "element" is the text, either HTML or PHP, to be output to the page, ie:

```
$pages[4][] = "HTML";
$pages[4][] = "<h1>This is an HTML page in the middle of my form</h1>";
```

`$conditions` – `array - optional`

This parameter contains all the conditions to apply to a page, to determine whether it should be displayed or not. You can have multiple conditions for a page, and every page in your form can have conditions (though it doesn't make sense to have a condition on the first page).

The first key in this array is the page number that the condition applies to. The value for that key is itself array, with three values in it. Each value is an array, each with an equal number of values. The first array is the elements for each condition, the second is the operators for each condition, and the third is the terms for each condition. So if the elements, operators and terms arrays each have two values, then that means there are two conditions for this page. The first values in each array make up the first condition, and the second values in each array make up the second condition.

The example above hopefully makes this clear. Keep in mind, the example above contains only one condition on one page.

`$introtext` – `string - optional`

This string should contain HTML that gets printed above the first page of the form. This HTML only appears above the first page.

`$thankstext` – `string or array – optional (highly recommended)`

This parameter can be a regular HTML string that will get printed on a final page, after the last specified page in the $pages array is completed by the user. Alternatively, this parameter can be an array with two values, just like the optional non-form pages. The first should be either "HTML" or "PHP" indicating whether the thank you text should be interpreted as HTML or PHP. The second value in the array should be the actual text, either HTML or PHP.

`$done_dest` – `string – optional (highly recommended)`

By default, when a user gets to the end of a multi-page form, they will be given a link to click to continue browsing the website. The default destination for this link is the first page of the form. In almost every case, another page in the website would be better than the first page of the form, but only the application developer knows what that page is. You should specify that page as the value of this parameter.

`$button_text` `- string - optional`

This parameter controls the text of the link on the thank you page which users see after completing the form. It has a generic default value, but something more specific would be better in most cases.

`$settings` `- array - special`

This is a special array used to pass information between various other functions and displayFormPages. By default this array is empty and you should specify it as "" if you need to use any of the subsequent parameters.

`$overrideValue` `- string or array - optional`

This value is passed through to the form and behaves just like the displayForm parameter of the same name.

displayEntries

The Scoop

This function is the basis for how lists of entries in forms are displayed. You should use List of Entries screens for this purpose. The API is very minimal compared to the number of configuration options available in List of Entries screens. If you need to push this part of the envelope for some reason, let's talk.

Syntax

```
displayEntries($formframe, $mainform, $loadview, $loadonlyview,
$viewallforms)
```

This function calls up the main interface to display lists of entries. This interface includes all reporting features as well, such as searches, sorts, calculations, exporting of data and saving of views.

Parameters

`$formframe` `- number - required`

This is either the ID number of the form to draw, or the ID number of the Framework you want to access. It is required and cannot be omitted.

`$mainform` `- number - optional`

This is the ID number of the form.   By default this is empty. It is only necessary to specify this if `$formframe` is a Framework name or ID.

`$loadview` `- string or number - optional`

This variable identifies a saved view which will be loaded instead of the default view of the entries in this form. `$loadview` can have the following kinds of values:

- 12 – where 12 is the ID number of a saved view in the 2.0 beta system.
- "Name of a view" – where the string equals the name of a saved view in the 2.0 beta system. If more than one view share the same name, then the first one found will be used.

`$loadonlyview` `- 1 or 0 - optional`

This parameter is intended for use with a loaded view. If this parameter is set, then none of the standard views are available in the Current View drop down list. This prevents users from changing the "scope" of the view to include other groups besides the ones that the loaded view uses.

`$viewallforms` `- 1 or 0 - optional`

This is an override setting, which is used to force the display for all forms in a Framework, regardless of the user's permissions to see them.

displayGrid

The Scoop

This function is used to display "grid" elements in forms. Chances are, if you want to customize the appearance of a form, you'll get a lot farther writing your own HTML and using the displayElement function.

Syntax

```
displayGrid($fid, $entry, $rowcaps, $colcaps, $title,
$orientation, $startID, $finalCell, $finalRow)
```

This function displays a table on the screen, containing cells that each have one element from a form in them. It is used to provide an alternative way of displaying a part of a form on the screen, since the default layout of the displayForm function is not always appropriate.

Example

One of the most useful applications of the displayGrid function is to provide a spreadsheet-like series of textboxes. Suppose you have a form that asks for budget information. Suppose the form has textbox elements like this:

Salaries – Budgetted:
Salaries – Actual:

Marketing Costs – Budgetted:
Marketing Costs – Actual:
And so on…

Such a form would be difficult to read and fill in if displayed in the normal way. But it could be easily displayed as a grid using this function:

```
$rowcaps[0] = "Salaries";
$rowcaps[1] = "Marketing Costs";
// continue the $rowcaps array for each set of elements in the form
$colcaps[0] = "Budgetted";
$colcaps[1] = "Actual";
displayGrid("27", "", $rowcaps, $colcaps);
```

This will now display the form something like this:

	Budgetted	Actual
Salaries		
Marketing Costs		

Parameters

`$fid` `- number. Required`

The ID number of the form that you want to display.

`$entry` `- number. Optional`

The ID number of the entry that should be displayed in the form for editing. By default, the form is displayed blank and a new entry is created when the form is saved.

If the form is a one-entry-per-user or one-entry-per-group form, then any existing entry will be displayed in the form if this parameter is left empty.

`$rowcaps` `- array. Required`

This array contains the text that should be used in the left margin of each row. The table will have as many rows as there are values in this array.

`$colcaps` `- array. Required`

This array contains the text that should be used in the heading row above each column. The table will have as many columns as there are values in this array.

`$title` `- string. Optional`

If present, the text in this parameter will be used as the title above the form, instead of the actual name of the form.

`$orientation` `- "horizontal" or "vertical". Optional`

This parameter controls the direction of the background shading in the table. Horizontal shading uses alternate colours for each row. Vertical shading uses alternate colours for each column. By default, this is set to "horizontal".

`$startID` `- number. Optional`

By default, the grid starts with the first element in the form and continues adding elements to the form as long as there are columns and rows to draw in the table. You can use this parameter to specify a particular element to start the grid with, and then the elements will be drawn in from that point in the form onwards according to their order in the form.

This is useful to display only part of a form in a grid, if other elements of the form are better suited to displaying in other ways.

This can be particularly useful when used in conjunction with the option to specify $formframe as an array in the displayForm function. Doing so lets you have a very long form in Formulize, and you can then then display only certain parts of the form at a time.

`$finalCell` `- array. Optional`

If this array is passed to the function, then the values in this array are drawn in the table cells at the end of each row. They are placed in their own column. The values should be valid HTML, and should not include the <td> and </td> tags.

This parameter is useful if you require some kind of summary information at the end of each row. If you query the form for the data in a particular entry prior to calling the displayGrid function, then you could parse the data and calculate totals or other information that would be useful to display in the final column.

This parameter must have numeric keys.

`$finalRow` `- string. Optional`

If present, the value of this parameter is drawn in the table as a final row. The value should be valid HTML, and should not include the <tr> and </tr> tags. Based on the number of values in the $colcaps array, you can determine how many cells you need to include in the row. Don't forget that there is one left margin column created for the row captions, and one right margin column created if the $finalCell array is present.

displayCalendar

The Scoop

This function lets you display the entries in a form, plotted on a calendar. It's quite useful, and as of August 2012, it has no corresponding screen type. If you need to use this API function, perhaps we could work together on creating a real screen type in Formulize to render calendars instead.

Syntax

```
displayCalendar($formframes, $mainforms, $viewHandles,
$dateHandles, $filters, $viewPrefixes, $scopes, $hidden, $type,
$start, $multiPageData)
```

This function displays a calendar on the screen, and populates the calendar with data from one or more forms and/or frameworks of forms. Currently only a large month view is supported (the month will take up most of the screen – not suitable for use in blocks).

Several of the main parameters that this function accepts must be set up as arrays that are in-synch with each other, ie: key 0 in each array corresponds to the first dataset, key 1 corresponds to the second dataset.

Examples

Display a calendar style view of the data in form 4. Suppose form 4 is a task tracking form which includes a field called "Deadline", that has ID number 21, and the form also includes a field called "Task Name", that has ID number 15. The Task name will be printed on the calendar in the box corresponding to the deadline

```
$form[0] = 4;
$text[0] = 15;
$date[0] = 21;
displayCalendar($form, "", $text, $date);
```

Suppose form 4 has been added to a framework called "Task Info". The Deadline field has been given the handle "deadline" and the Task Name field has been given the handle "name". Suppose also that you only want tasks that have been assigned to "John Smith" to be shown on the calendar.

```
$framework[0] = 51; //Task Info framework
$mainform[0] = 21; //task tracker form
$text[0] = "name";
$date[0] = "deadline";
$filter[0] = "personname/**/John Smith";
displayCalendar($framework, $mainform, $text, $date, $filter);
```

(See the write up for `getData` for details about the filter syntax.)

Suppose that, instead of the name of the task being drawn on the screen, you want "Due Today! – " to be used as a prefix before the task name. Add one more parameter:

```
$prefix[0] = "Due Today! - ";
displayCalendar($framework, $mainform, $text, $date, $filter, $prefix);
```

Suppose you want to display meeting times and task due dates on the same calendar:

```
$framework[0] = 51; //Task Info framework
$mainform[0] = 21; //task tracker form
$text[0] = "name";
$date[0] = "deadline";
$prefix[0] = "Task Deadline: ";
$framework[1] = 55; //Staff Info framework
$mainform[1] = 17; //meeting bookings form
$text[1] = "meeting";
$date[1] = "date";
$prefix[1] = "Meeting:";
displayCalendar($framework, $mainform, $text, $date, $filter, $prefix);
```

Note that you can use the same data source as an input to the calendar multiple times. You might want to do this if there are two different dates in a particular form or framework and you want both of them displayed on a calendar. ie: an order tracking form might have an order date and a ship date on it, so you would use the same form as the data source twice, but each time you would use different fields for the dateHandle and viewHandle.

Parameters

`$formframes` `- array - required`

This is an array of all the form IDs or framework IDs to use as data sources for this calendar. Required.

`$mainforms` `- array - optional`

This is an array of the main forms for each framework. If a given array key in `$formframes` refers to a Framework, then the corresponding array key in `$mainform` must be a valid main form ID for that Framework. Otherwise, it is not required.

`$viewHandles` `- array - required`

This is an array of the element handles of the elements that are to be displayed on the calendar. ie: if you want task names displayed on the calendar, then pass in a reference to the textbox element labelled "Task Name". This array is required, and there must be a valid handle for each dataset.

You can specify multiple handles for each dataset. For instance, you may want to include the name and the city on a calendar view of conference dates. If you specify an array of handles for a given dataset, then the values for all those handles will be used. Example:

```
$viewHandles[0][0] = "name";
$viewhandles[0][1] = "city";
```

`$dateHandles` `- array - required`

This is an array of the element handles of the elements that contain the dates on which each entry should be displayed. ie: if you want task names displayed on their due date, then pass in a reference to the datebox element labeled "Deadline". This array is required, and there must be a valid handle or ID for each dataset.

If you have two date fields in the underlying form, you can use one as the start date and one as the end date, which allows you to plot a date range on the calendar. To specify a start and end date instead of just a single date, use an array. The first value in the array is the handle of the start date, and the second value is the handle of the end date. Example:

```
$dateHandles[0][0] = "startdate";
$dateHandles[0][1] = "enddate";
```

`$filters` `- array - optional`

This is an array of filters to apply to the data extraction operation for each dataset, if any. The syntax of the filters is discussed on page 38. This array is optional, and if it is present, a filter is only required for each dataset that you want to apply a filter to.

`$viewPrefixes` `- array - optional`

This is an array of text strings to append to the beginning of the text returned by the `$viewHandle` for each dataset. See the examples above for an illustration. This array is optional, and if it is present, a prefix is only required for each dataset that you want to apply a prefix to.

`$scopes` `- array - optional`

This is an array of scopes to use when getting data to populate the calendar with. A scope limits the entries returned to only those that were made by users in a certain group or groups. Valid values are:

- "mine" – include only entries made by the current user.
- "group" – include all groups that the user is a member of.
- "all" – include all groups.
- a list of specific group IDs separated by commas and with commas at the front and back. ie: ,5,12,14,

This array is optional, and if it is present, a scope is only required for each dataset that you want to apply a scope to.

`$hidden` `- array - optional`

This is an array of values that you want remembered when the page reloads. This is meant primarily for situations where you have other form elements manually created in a Pageworks page that calls `displayCalendar`. In that situation, when a user clicks on a part of the calendar interface, the state of the manually created form elements will be lost unless they are include in `$hidden`, and the manual code is programmed to look for them in $_POST on page loading.

`$type` `- string - optional`

This is a string meant to indicate what sort of calendar to draw. Currently, only "month" is supported. Month is the default value, and this string is optional.

`$start` `- string - optional`

This string is used to indicate the starting year and month for the calendar. The format of the string must be: YYYY-mm, ie: 2005-09. This string is optional. The default value is the current month.

`$multiPageData` `- array - optional`

This parameter is used to integrate a multi-page form with a calendar. If a multi-page form is used with a calendar, then this array must contain four values, each corresponding to the following parameters for displayFormPages: $pages, $conditions, $introtext, $thankstext.

displayElement

The Scoop

This function is used to render all elements. You would use it if you're making a special version of a form with your own markup and you want to control where/how each form element appears on that page. Inside Pageworks, this function triggers certain other code to be invoked, so that the elements will save data as expected. If you use this in plain PHP, then you will have to make sure that /modules/formulize/include/readelements.php is included on the page that data is submitted to, or else nothing will be saved!

Syntax

```
displayElement($framework, $element, $entry)
```

This function is used to display a particular element from a form on page. Each element displayed is tied back to a particular entry in the form, so the value displayed reflects the current state of that entry, and if the value is modified and saved, that entry in the form is updated. If this function is used in a Pageworks page, a "Save" button will appear at the bottom right of the page, allowing users to save any changes they may make to the state of any elements in the page.

This function is aware of the display settings for an element. So if you try to display an element to a user who is not a member of a group that is allowed to see the element, then the element will not display.

This function will return the string "rendered" if it successfully renders the element. It returns "not_allowed" if the user is not supposed to see the element, and "invalid_element" if you do not pass a valid $element to it.

Also of note: there is a strange sounding configuration option for all form elements called "Include as a hidden element for users who can't see it". This has an effect when a new entry is being created. If an element is passed to displayElement, but the user is not allowed to see it, and the "Include as hidden…" option is on for the element, then the element will be included as a hidden element in your form, and the default value for the element will be submitted with the form. This is useful if you must set certain default values in the database for your application to function correctly. If the displayElement function renders an element this way, then it returns the string "hidden".

FYI: the displayGrid function makes use of the displayElement function to present the elements in the table. Consulting the source code for displayGrid may be instructive to an advanced developer.

Examples

Suppose you have a task list form and there is a yes/no question in the form indicating if the task is complete. That question has element number 43. You want that yes/no option to appear beside each entry in the task list that you draw on a particular Pageworks page.

```
$tasks = getData("Tasklist", "tasks");  // details on getData are below

foreach($tasks as $onetask) {
      // see below for details of the internalRecordIds function
      $ids = internalRecordIds($onetask, "tasks");
      print display($onetask, "name");
      displayElement("", 43, $ids[0]);
      print "<br>";
}
```

The output of the above code would be a list of tasks from the Tasklist framework, and beside each one would be element 43 from the form, showing the value of that element for that particular task. In this example, that is supposed to be a yes/no radio button indicating whether the task is complete or not.

Parameters

$framework `- deprecated`

For historical compatibility, this is the first parameter of the function, but it should always be empty now. Just pass in “”.

$element `- number, string or object - required`

If a number, this parameter is treated as the element ID number of the element you want to display. If a string, it must be the handle of an element. If an object, it must be the element object that you want rendered.

$entry `- number - optional`

This parameter is the ID number of the entry that the value in this element should be based on. Optional. If omitted, then any value entered into the element and saved by the user will result in a new entry being created in the form.

displayCaption

The Scoop

This function will print out the caption for an element. It is useful in making custom forms.

Syntax

```
displayCaption($formframe, $element)
```

This function returns the caption for an element and nothing else. Useful with displayElement to provide custom control over a form's layout.

Parameters

$formframe `- string - optional`

For historical compatibility, this is the first parameter of the function, but it should always be empty now. Just pass in “”.

$element `- number, string or object - required`

If a number, this parameter is treated as the element ID number of the element you want to display. If a string, it must be the handle of an element. If an object, it must be the element object that you want rendered.

displayDescription

The Scoop

This function will print out the descriptive help text for an element, if any. It is useful in making custom forms.

Syntax

```
displayDescription($formframe, $element)
```

This function returns the descriptionfor an element and nothing else. Useful with displayElement and displayCaption to provide custom control over a form's layout.

Parameters

`$formframe` `- string - optional`

For historical compatibility, this is the first parameter of the function, but it should always be empty now. Just pass in “”.

`$element` `- number, string or object - required`

If a number, this parameter is treated as the element ID number of the element you want to display. If a string, it must be the handle of an element. If an object, it must be the element object that you want rendered.

htmlForElement

The Scoop

This is the most recent addition to the API. It will give you the markup that would be used for an element. But what it returns will have an arbitrary name assigned, so you can embed the markup in your page without fear of it creating a new entry in the form, or otherwise altering the values that are in the form. This is useful if you need to display a copy of an element to the users, because it has the right choices in it that you want to display, but when users interact with that copy, you don't want to them to end up creating or updating entries in a form.

Syntax

```
print htmlForElement($handle, $nameForHTML, $entry_id);
```

This function returns the markup for the specified element, and the name attribute in the markup will be whatever you pass in (if anything, there's a default name if you pass in nothing). You can then do whatever you want with the markup, probably print it out to the screen at some point.

Parameters

`$handle` - `string - required`

This must be the data handle for the element that you want to display. The data handle is specified on the Settings tab when configuring the element. You could probably pass in the element ID too and this would still work, or even a full element object. But the default assumption is that a handle is what's being used, and that's all that's guaranteed to work in the future.

`$nameForHTML` - `string - optional`

This is the value that the name attribute in the markup will have. If you don't specify anything, then “orphaned_formulize_element” will be used.

`$entry_id` - `string or number - optional`

If you want the markup to be based on a specific entry in the form where this element is from, then you can specify the ID number of the entry here. Technically “new” is a valid value for this parameter too, and that is the default value if you don't specify something yourself. When set to “new” (default) then a blank element will be returned.

displayFilter

The Scoop

This function is deprecated, though it has some specific uses in conjunction with calendars. You should use htmlForElement if you want to do this kind of thing.

Syntax

```
displayFilter($page, $name, $filtername, $element, $overrides)
```

This function is used to put a dropdown list on the same Pageworks page as a calendar, and the dropdown list operates as a filter that limits the items drawn on the calendar to only ones that match the value selected in the list. The filtering is based on an element in the underlying form, which you specify as one of the parameters of the function. The values for that element become the options in the dropdown list.

For example, if you have an event calendar and each event is classified according to regions – north, south, east and west – and there is an element in the form which is used to specify the region, then you could use this function to create a filter with those four options in it and selecting east, for example, would result in only events that occur in the east region being displayed.

Currently, this function works only when tied to an underlying selectbox in the form.

Parameters

`$page` `- number - required`

This must be the ID number of the Pageworks page where the displayFilter `function` is being used.

`$name` `- string - required`

This is a name to be used as the name of the form that the filter creates for itself. It can by anything, as long as it doesn't conflict with any existing forms that are created in the Pageworks page.

`$filtername` `- string - required`

This is a name to be used as the name of the filter itself within its form. It can be anything.

`$element` `- number - required`

This must be the element ID number of the element on which the filter options should be based.

`$overrides` `- array - optional`

This is an array containing a default value for the filter.

displayElementSave

The Scoop

This function is intended for use with Pageworks only. If you're making a custom form page in PHP, put the submit button wherever you want. Just make sure that /modules/formulize/include/readelements.php is included by the receiving page, or else no data will be saved.

Syntax

```
displayElementSave($text, $style)
```

This function is used to manually place a Submit button on the page. By default, Pageworks will add a button labeled "Save" to the lower right corner of any page where the displayElement or displayGrid functions are used. However, you may wish to manually specify what such a button should be labeled and/or where it should appear on the page.

If so, then use the displayElementSave function to place the button on the page.

Parameters

`$text` `- string - optional`

This is the text that will appear on the button. By default is it "Save".

`$style` `- string - optional`

This text will be rendered as a CSS inline style for the button.

displayButton

The Scoop

This function was a really old way of doing in Pageworks what you do now with custom buttons in list-of-entries screens. Chances are you won't ever be using this. Explore custom buttons in list-of-entries screens instead. Get in touch if you really want/need to use this and let's talk.

Syntax

```
displayButton($text, $element, $value, $entry, $action, $type,
$framework)
```

This function displays a clickable button, or text link, on a Pageworks page, and if the user clicks it then the value of a particular element in a particular entry in a form is altered. Like displayElement, this provides a way for users to interact with data in forms without actually working with the form itself. In this case, the users don't even interact with the actual element associated with the value.

This function does not work with linked selectbox elements.

Examples

Suppose you have task list, and you want to list the tasks, and give users a big button they can click to say the task is done. There is an element in the form which indicates whether the task is done or not, but rather than giving the user that element to work with, you just want to give them a button to click. The element in the form is number 43 and is a checkbox labeled "task completed".

```
// assume $task is the output of a previous getData function

foreach($task as $onetask) {
      // for details of the internalRecordIds function see below
      $ids = internalRecordIds($onetask, "task");
      print display($onetask, "name");
      displayButton("Done!", 43, "task completed", $ids[0], "replace");
      print "<br>";
}
```

Suppose you have list of activities and you want to let users sign up for them simply by clicking a button. You could create a form where each entry is an activity, and include a textarea box in the form where user's names will be entered. Then make a Pageworks page that lists the activities that have been entered, and use displayButton to make a button beside each activity and if a user clicks on the button, their name will be added to the textarea box.

```
// assume $activities is the output of a previous getData function

foreach($activities as $activity) {
      // for details of the internalRecordIds function see below
      $ids = internalRecordIds($onetask, "task");
      print display($activity, "name");
      displayButton("Sign Me Up!", 25, $xoopsUser->getVar('name'), $ids[0],
"append");
      print "<br>";
}
```

Parameters

`$text` `- string - required`

This is the text that will appear on the button (or in the link)

`$element` `- number, string or object - required`

If a number, this parameter is treated as the element ID number of the element where the value is going to be modified.

If a string, this parameter is treated as the framework handle of the element. (See the $framework parameter below.)

If an object, this parameter must be the element object that you want the button to modify the value for.

`$value` `- string - required`

This is the value that should be used to modify the existing value of the element.

`$entry` `- number - optional`

This is the ID number of the entry where the value is going to be modified. Optional. If not specified, then a new entry is created with the value.

`$action` `- string - optional`

Valid values are: replace, append or remove. This is an optional parameter which indicates the type of modification to perform. Replace overwrites the current value with the one specified for the $value parameter here. Append adds $value to the end of the current value for the element. Remove searches the current value of the element for $value and if found, removes that text from the current value. The default is "replace".

`$type` `- string - optional.`

This optional parameter controls whether the thing the user clicks is a button or an HTML link. There is no functional difference between them, it is purely aesthetic. The default is "button". To specify a link, use "link".

`$framework` `- string - optional`

The name of a framework that you are working with. Specify a handle for any form element in the framework using the $element param (see above).

formulize_getCalcs

The Scoop

This function was designed to let you grab some calculations that are part of several saved views, so you can then display those calculations in some kind of dashboard. This way people don't have to move between many different saved views to see the numbers. Formulize can do the SQL and math for you, and you just format the output on a custom PHP page.

Syntax

```
formulize_getCalcs($formframe, $mainform, $savedView, $handle,
$type, $grouping)
```

This function is used to retrieve the calculations that are in effect for a certain saved view. The raw HTML output for the calculations is returned as a multidimensional array with the following structure:

```
$resultArray
  [element handle]
    [calculation type]           - sum, min, max, count, avg or per
      [auto incrementing number] - one for each grouping in effect
        ['result' or 'grouping'] - 'result' is the HTML for the calculation
                                 - 'grouping' is a comma separated list of
                                   all the values that this calculation has
                                   been grouped by
```

You can then iterate over the returned array to manually do something to display the calculations. This is useful for creating custom dashboards of the current state of data in forms. This example code will print a simple listing of all the calculations returned by the function:

```
// $calcs is the result of a call to
// the formulize_getCalcs function
function printCalcResult($calcs) {
     $element_handler = xoops_getmodulehandler('elements',
'formulize');
     foreach($calcs as $handle=>$thisCalcData) {
          $elementObject = $element_handler->get($handle); //
only works when handle and ele_id are the same
          $caption = $elementObject->getVar('ele_caption');
          print "<h3>$caption</h3>\n";
          foreach($thisCalcData as $type=>$results) {
               if(count($results)>1) {
                    foreach($results as $result) {
                         print "<p><b>Grouped By: ".implode(",",
$result['grouping'])."</b></p>\n";
                         print $result['result'];
                    }
               }
               else {
                    print $results[0]['result'];
               }
          }
     }
}
```

Parameters

`$formframe` `- number - required`

This is either the ID number of the form to draw, or the ID number of the Framework you want to access. It is required and cannot be omitted.

`$mainform` `- number - optional`

This is the ID number of the form. By default this is empty. It is only necessary to specify this if $formframe is a Framework name or ID.

`$savedView` `- string or number - required`

This must be either the ID number of the saved view, or its name as typed when it was saved. This is the saved view that contains the calculations you are looking for.

`$handle` `- string or number - optional`

If included, this value will be used to limit the calculations returned to only those performed on this element.

$type – `string or number - required`
If included, this value will be used to limit the calculations returned to only those of this type.

$grouping – `string or number - required`
If included, this value will be used to limit the calculations returned to only those with this value as one of the calculations grouping values. Note that this will match the value that the grouping is performed with, not the element that the grouping value comes from. For example, if you have a series of sum totals being grouped by a "province" element, then if you specify "Ontario" for this parameter, only the calculations where the province is "Ontario" will be returned by the function.

formulize_writeEntry

The Scoop

This is a key function, that is used to save a value or values to the database for a specific entry. It is a handy “front end” to the underlying data handler method that does the actual writing, since this function will also intelligently manage the entry ownership data, among other things, all as a single operation.

Syntax

```
formulize_writeEntry($values, $entry, $action, $proxyUser,
$forceUpdate, $writeOwnerInfo)
```

This function is used to write data to an existing entry, or to create a new one.

Parameters

$values – `array`

This is an array of the values to write to various elements in the form. The keys must be either the element ids that are being written to, or the element handles. The values must be the values to write to the element identified by the key. The elements must all be from the same form!

$entry – `number or string`

The ID number of the entry being written to, or "new" for creating a new entry.

`$action` `- deprecated - no effect`

`$proxyUser` `- number - optional`

The ID number of the user who should be recorded as the creator of this entry, if this is a new entry, and the current user's ID should not be used.

`$forceUpdate` `- true or false - optional`

By default this is false, but if it is set to true, then the query will be executed even on a GET request, when database updates are normally disallowed.

`$writeOwnerInfo` `- true or false - optional`

By default this is true, but if set to false, then the entry ownership information for a new entry will not be written to the the entry_owner_groups database table.

getData

The Scoop

One of the most important and oldest functions in the API. Use this to retrieve datasets from the database. A key to using it effectively, is understanding the structure of the data that it returns to you, which is in a multidimensional array. Unfortunately, there is no support for specifying the fields you want back, so all are returned. This degrades performance in some situations, and would be a prime candidate for optimization by eager coders. :-)

Syntax

```
getData($framework, $form, $filter, $andor, $scope, $limitStart,
$limitSize, $sortField, $sortOrder)
```

Parameters

`$framework` `- number - optional`

This is the ID number of the Framework you want to access. It can be omitted if you are querying a form directly. By default is it empty.

`$form` `- number - required`

This is the ID number of the form. It is a required parameter and cannot be omitted.

`$filter` `- number or string or array - optional`

If this is a number, then that number is treated as the ID of the record that you want to return. Only that record will be returned.

If this is a string, then it is assumed to be in the format described below, containing filtering options to apply to the query that extracts the data from the database. By default this is empty, no filter is applied.

This is formatted as a string (instead of an array which might seem more logical) in order to make it easier to work with filter parameters using the GET method if necessary. (Not that anyone every has actually done that!!) The format of the string is as follows:

handle1/**/term1][handle2/**/term2][etc…

The filtering is very simple, it only looks for a 'LIKE' match within the field.

The handles must be the data handles for the elements in question (or the element IDs).

Example:

prov/**/ON][city/**/b

Assuming prov is the data handle for an element that contains provinces, and city is the data handle for an element with city names, then the filter above will return data pertaining to all cities in Ontario that have the letter b in their name.

Note that the][are only used to separate search terms, they do not precede or conclude the filter string.

Operators

You can specify particular operators to use for each part of the filter, if LIKE is too general for your needs. To do this, add on another term separated by /**/, for example:

prov/**/ON/**/!=

This filter will return data where the value of the prov field is not equal to ON.

Newest

There is a special kind of filter that is **valid only for a date field**. This is called the "Newest" filter and it is specified as follows:

HandleForDateField/**/term/**/newestX

Where X is a number indicating the number of records you want returned. This special filter simply involves the addition of the extra term "newest" plus the number.

The normal term is ignored if the special newest term is specified. **Note again: this is only valid for date fields.** ie: this will return the X number of records with the most recent dates in that field.

Example:

date/**/whatever/**/newest5

This example will return the records with the five most recent dates in the field identified by the handle Date.

Metadata Filters

There are five special field names that are reserved for metadata, ie: information about an entry. They are: creation_uid, mod_uid, creation_datetime, mod_datetime and creator_email. creation_uid is the ID of the user who created the entry. It never changes. mod_uid is the ID of the user who last modified the entry. creation_datetime is the date the entry was created. mod_datetime is the date the entry was last modified. creator_email is the email address of the user who created the entry (taken from their account profile). You can use these four special field names as part of a filter string:

"creation_uid/**/" . $xoopsUser->getVar('uid') . "/**/!="

That is a valid filter string which will return entries that were not created by the current user.

If $filter is an array, then it can be structured as a series of filter strings, each with their own and/or setting. This technique is used to handle more complex boolean operations than can be handled by a single filter string with one common and/or value (which is set by the next parameter).

The format for the array is as follows:

`$filter[0][0]` … this is the and/or setting for the first filter string. Valid values are "and" or "or".

`$filter[0][1]` … this is the first filter string to use. It follows all the syntactic rules as described above.

`$filter[1][0]` … this is the and/or setting for the second filter string.

`$filter[1][1]` … this is the second filter string to use.

`$filter[2][0]` … and so on….

`$andor` `- AND or OR - optional`

This parameter specifies how to interpret multiple filters, ie: a filter string that contains more than one set of handles and terms separated by][. By default it is set to AND, and so *the overlapping* set of records found by all the filters will be returned. If it is set to OR, then all records found by all filters will be returned.

For example, in the example above, prov/**/ON][city/**/b, if `$andor` is set to OR, then all records where either one of "province contains ON" or "city contains letter b" will be returned. If `$andor` is set to AND or left at the default setting, then only records where both those conditions are true will be returned.

`$scope` `- string or array - optional`

This parameter is used to filter records according to the group membership of users. This parameter is used extensively by the logic within the other main functions, but this should not need to be used by basic applications. By default this parameter is empty (entries by all users from all groups are returned).

The easiest way to use this parameter is to pass an array of group IDs. Those groups will be used as the scope.

A more complex, but valid way to use this parameter is to pass a specially formatted string:

uid = X OR uid = Y OR uid = Z and so on.

This of course puts the burden of determining the user ids on the application developer, but this is a useful approach if you want to limit the scope to a group of users that is smaller than any particular group in the website.

`$limitStart` `- number - optional`

A number indicating the position (not ID) of the first record that we should retrieve from the database (corresponds to the first value of the LIMIT statement for MySQL).

`$limitSize` `- number - optional`

A number indicating how many records we should retrieve from the database (based on the starting position above – corresponds to the second value of the LIMIT statement for MySQL).

`$sortField` `- string - optional`

The handle of the element that we should sort the query by.

`$sortOrder` `- string - optional`

Either "ASC" or "DESC" indicating the order of the sorting.

Output of getData

The most important thing to understand about getData is what it gives you. It is helpful to know the basics of the output array that it creates. Although the display functions discussed in the next section eliminate the need to refer to all the bits and pieces of the array, it may help in building certain systems if you understand the structure of this array.

The abstract description of this array would be something like this:

```
$array [master id][form handle][record id][field handle][value id] = value
```

The only values contained in the array are the actual pieces of information submitted through the form(s) that was queried. **The array simply includes a whole bunch of array keys to help identify where each value came from.**

Here's an example of real data. Assume the result array is called "data":

```
$data[0][profile][22][name][0] = "Geoff"
$data[0][profile][22][age][0] = "99"
$data[0][activity log][49][activity_name][0] = "work on workshops "
$data[0][activity log][49][activity_characteristics][0] = "hard"
$data[0][activity log][49][activity_characteristics][1] = "boring"
$data[0][activity log][55][activity_name][0] = "documentation"
$data[0][activity log][55][activity_characteristics][0] = "hard"
$data[0][activity log][55][activity_characteristics][1] = "literary"
$data[1][profile][16][name][0] = "Cory"
$data[1][profile][16][age][0] = "66"
$data[1][activity log][31][activity_name][0] = "website audit"
$data[1][activity log][31][activity_characteristics][0] = "time consuming"
$data[1][activity log][31][activity_characteristics][1] = "challening"
$data[1][activity log][31][activity_characteristics][2] = "boring"
$data[1][activity log][38][activity_name][0] = "site hacking"
$data[1][activity log][38][activity_characteristics][0] = "challenging"
```

So, that is a data set with two results in it. Each result is made up of three separate entries in two different forms. One entry is from the profile form, and two entries are from the activity log form. The fact that there are only two activity log entries per main result is just chance. Obviously over time the two users would create several activity log entries and would probably not create the exact same number.

The first number identifies the main result that each "line" belongs to. The second value is the name of the form that the data is taken from. The third number is the ID number of the entry in that form which this data is coming from. This is an important point.

Remember that the data in each form is stored sort of like this:

Activity Log Form:

```
31,activity_name=website audit
31,activity_characteristics=time consuming, challenging, boring
38,activity_name=site hacking
38,activity_characteristics=challenging
49,activity_name=work on workshops
49,activity_characteristics=hard, boring
55,activity_name=documentation
55,activity_charateristics=hard, literary
```

Profile Form:

```
16,name=Cory
16,age=66
22,name=Geoff
22,age=99
```

So that number in the third position represents the ID of the entry in the particular form that contains all the data related to the specific value on that line. To put it another way, if you want to get all the data about Cory from the profile form, then you need to query for ID 16 in the profile form. If you want to get all the data about Cory's activity on documentation, then you need to query for ID 55 in the activity log form.

Compare the result set above, with the description of the underlying data in the forms that produced it. Hopefully you will quickly see where all the information in the array keys comes from.

The fourth value in the results array is the handle of the form element that the data was entered into. The fifth value is the id number of that value within that form element. In most cases, this is simply a zero, and there is only one value. But in some cases, for instance checkboxes, or multiple selectboxes, where users can pick more than one response for a form element, there would be multiple values. The hypothetical activity characteristics element in this example is like that. Suppose it is a series of checkboxes. That leads to a variable number of answers and so there are varying numbers of values related to that form element in the final results array.

There are five pieces of metadata for each entry in the result set: creation_uid, mod_uid, creation_datetime, mod_datetime, creator_email.

They are accessible using the display functions (see below), just like any regular form element.

creation_uid is the ID of the user who created the entry (the owner of the entry). This never changes.

mod_uid is the ID of the user who last modified the entry.

creation_datetime is the creation date of the entry.

mod_datetime is the most recent modification date of the entry.

creator_email is the e-mail address associated with the account of the user who created the entry.

Functions for Gathering IDs from Entries in a Dataset, and Sorting Data

`internalRecordIds($entry, $formhandleOrId, $id)`

This important function takes the same $entry and $id parameters as the display function (see below). The second parameter is the title of the form, or the ID number of the form. It will return an array of all the IDs of the entries in that particular form which are part of the requested entry. For example, considering the sample data set above, this line:

```
$entry = $data[0];
$ids = internalRecordIds($entry, "Profile Form");
```

Will return an array like this: [0] => 22.

You can then use the id or ids returned to drill down on a specific entry in a subsequent getData call, or to construct a link to that entry.

The $formhandleOrId parameter is optional, and if omitted then an array will be returned where the keys are *all* the form titles in the dataset, and the value for each will be an array of the ids of the included entries from that form.

The $formhandleOrId parameter can also be an array of handles, or form Ids, if you want to get the ids for entries in multiple forms at once. In that case, the keys will be all the form handles you specified, and the value for each will be an array of the ids of the included entries from that form. Specifying $formhandleOrId as an array only makes sense if you're dealing with a Framework and there's more than one form making up the data in the dataset.

`resultSortRelevance ($data, $handles, $terms, $weights)`

This function is used to sort a dataset based on the prevalence of certain search terms. Pass in the dataset, along with an array of handles of elements in the dataset, and an array of search terms. Optionally, you can pass in a weighting array with numbers corresponding to each handle, indicating their relative importance (so hits in some handles will give higher rankings to that result versus others).

The dataset is sorted by counting the number of occurrences of each search term in each specified element, and multiplying the number of occurrences by the optional weighting (1 is used if no weighting is specified). The entry with the highest score in the dataset takes first position and so on.

Example:

```
// get a list of products
$productList = 3; // Framework ID for the Product List Framework
$productsForm = 7; // form ID for the products form
$data = getData($productList, $productsForm);

// Parse search terms from the user.  This function does not use the terms
// in any database queries, so it is *not* a SQL injection risk to pass
// the user specified text straight to the function in this case.
$terms = explode(",", $_POST['searchterms']); // breakup into array

// specify the fields to search, using handles
$handles[] = "productname";
$handles[] = "productdescription";

// consider hits in the product name three times as valuable as hits
// in the description
$weights[] = 3;
$weights[] = 1;

$data = resultSortRelevance($data, $handles, $terms, $weights);
```

Data Parsing and Display Functions

There are several functions available for formatting and displaying data returned from the getData function. The examples below refer to the sample dataset above extensively, so it pays to be familiar with it.

`display($entry, $handle, $id, $localid)`

Returns the value(s) of an element handle within a given entry.

$entry – either an entire results set, or a single entry from a result set (see below). Required.

$handle – either the element handle of the field you want to access or the ID of that element. Required.

$id – if an entire results set is passed in with $entry, then this is the array address of the entry in the results set you want to access. Optional, but necessary if an entire results set is passed through $entry.

$localid – this parameter is used to access the values for just one specific entry within a form that makes up the entire entry being accessed (see below). Optional.

Examples:

The following are both valid, and equivalent:

```
$staffFramework = 24;
$profileForm = 33;
$results = getData("", $profileForm);

print display($results, 'name', 0); // prints the name from master result 0

$entry = $results[0];
print display($entry, 'name'); // prints the name from master result 0
```

The localid is used to display only values related to a certain entry in a certain form. For instance, in the hypothetical result set above, the following code would display the activity characteristics related to Geoff's first activity log entry:

```
$results = getData($staffFramework, $profileForm);
$entry = $results[0]
$values = display($entry, 'activity_characteristics', '', 49);
```

ie: the $values array would be: [0]=>'hard' [1]=>'boring'

Contrast with:

```
$entry = $results[0]
$values = display($entry, 'activity_characteristics');
```

Without the filter for internal id 49, $values would be: [0]=>'hard' [1]=>'boring' [2]=>'hard' [3]=>'literary'

Note also that the display function can return an array, if there is more than one value corresponding to the element name specified. In the first example above, passing 'name' results in the single name being returned, as a string. In the second example, $values becomes an array since there is more than one value for activity characteristics. Of course, this is a factor of the number of values assigned to a particular entry in a particular form element, not a factor of the form element itself. So although this example is very similar to the previous example, $values would not be an array:

```
$entry = $results[1]
$values = display($entry, 'activity_characteristics', '', 38);
```

$values would equal 'challenging'. This is because in the master result entry number 1 (as opposed to number 0), internal id 38 contains only one value for the activity characteristics form element.

displayPara($entry, $handle, $id)

This function is used to take the contents of a textarea box and formats it as a series of HTML paragraphs. ie: the following text:

```
This is a paragraph
This is another paragraph
This is yet another paragraph

This is a fourth paragraph
```

Would be returned like this:

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
<p>This is yet another paragraph</p>
<p>This is a fourth paragraph</p>
```

Note that blank lines in the textarea box are ignored.

The parameter syntax is exactly the same as for the display function; an entire result set plus master ID, or a single entry can be passed in.

There can be more than one instance of the requested form element in the entry requested, for instance suppose there is another field in the activity log form in the hypothetical results array above, a field called 'activity description'. Suppose 'activity description' is a text area box. We would expect that master id 0 would have two values generated from this box, one for internal id 49 and one for 55. Similarly, master id 1 would have two values, one for internal id 31 and 38. In that case, displayPara would return an array containing HTML formatted text like above, where each value in the array corresponds to one of the text areas in the master entry.

displayBR($entry, $handle, $id)

This function returns the contents of a text area box and formats it as HTML with simply <br> tags between each line. ie: the following text:

```
This is a paragraph
This is another paragraph
This is yet another paragraph

This is a fourth paragraph
```

Would be returned like this:

```
This is a paragraph<br>
This is another paragraph<br>
This is yet another paragraph<br>
This is a fourth paragraph<br>
```

Note that blank lines in the textarea box are ignored.

The syntax is exactly the same as for the display function; an entire result set plus master ID, or a single entry can be passed.

There can be more than one instance of the requested form element in the entry requested, for instance suppose there is another field in the activity log form in the hypothetical results array above, a field called 'activity description'. Suppose 'activity description' is a text area box. We would expect that master id 0 would have two values generated from this box, one for internal id 49 and one for 55. Similarly, master id 1 would have two values, one for internal id 31 and 38. In that case, displayPara would return an array containing HTML formatted text like above, where each value in the array corresponds to one of the text areas in the master entry.

displayList($entry, $handle, $type, $id, $localid)

Like the other "display" functions, this one supports the same parameters, with the addition of the $type parameter, which can be either "bulleted" or "numbered". Default is bulleted. Note that this function supports the $localid parameter while displayBR and displayPara do not (yet).

This function returns the contents of a text area box and formats it as an HTML list. ie:

```
This is a paragraph
This is another paragraph

This is yet another paragraph
```

Would be returned like this:

```
<ul>
<li>This is a paragraph</li>
<li>This is another paragraph</li>
<li>This is yet another paragraph</li>
</ul>
```

Note that blank lines in the textarea box are ignored.

There can be more than one instance of the requested form element in the entry requested, for instance suppose there is another field in the activity log form in the hypothetical results array above, a field called 'activity description'. Suppose 'activity description' is a text area box. We would expect that master id 0 would have two values generated from this box, one for internal id 49 and one for 55. Similarly, master id 1 would have two values, one for internal id 31 and 38. In that case, displayPara would return an array containing HTML formatted text like above, where each value in the array corresponds to one of the text areas in the master entry.

`displayTogether($entry, $handle, $sep, $id, $localid)`

This function is very similar to the other "junior" display functions, except it takes the value passed to $sep and uses that as "glue" between all the values that are returned. It is intended to be used with form elements that allow multiple selections, such as checkboxes and multi-select boxes, rather than textboxes. It will take all the values returned by the standard display function and merge them into a single string with $sep in between each one.

Low Level Functions

Within the Pageworks or your own custom PHP code, you may need to perform some other tasks related to forms and data. To assist, there are some other miscellaneous functions available, including the important ones listed below. Any function in the Formulize include/functions.php file is actually available within a Pageworks page, or in PHP if you include that file. Most of those functions are only useful when called internally from displayForm or displayEntries, but some have general uses, such as:

`getCurrentURL()`

This function simply returns the current URL complete with all parameters.

`printSmart($value, $chars)`

$value – a text string. Required.

$chars – the number of characters of the string you want displayed. After this number of characters, the rest of the string is replaced with "…". Optional. Default is 35.

This function returns the newly formatted string, it does not print the result to the screen.

Further Documentation

For more details about how the insides of Formulize work, check out the PDF called *Inside Formulize: A Developer's Guide*.