Web-based

Technical Guide

This is a pre-release document. It has not yet undergone a thorough edit or content review and is subject to frequent, unreleased revision. Some parts of this document are still under development.

Last updated February 11, 2009

Web-based Architecture 2
The Browser Presentation Blueprint 3
Annotated #1 Script. 7
Annotated #2 Script. 9
Annotated #3 Script Template. 13
Error Message Handling in PHP Scripts 15

Customizing FINANCIER 17
Adding a Page to the FINANCIER Menu 18
Adding a Field to an Existing Page 20
Adding an Institutional Page 25

WEB-BASED ARCHITECTURE

The Web Architecture topics in the FINANCIER Help Library provide the information you'll need as a programmer responsible for maintaining FINANCIER at an institution. Use of this information assumes strong PHP and PL/SQL programming skills.

These topics document WolffPack's browser framework and the scripts involved in form processing, with practical guidelines for adding institutional forms to the system, and a description of the utility programs and scripts that can be employed.

WolffPack strongly recommends limiting institutional customization of FINANCIER, as all functionality is built to uphold best practices and to provide maximum flexibility for setting parameters to reflect your business rules. And, adhering to the base system greatly simplifies the implementation of enhancements, maintenance updates and annual Regulations releases. Nevertheless, we recognize that many institutions will need to add fields and pages to support institutional functions.

Browser Presentation Structure

Browser Presentation Blueprint (page 3)

Structure of a #1 script (page 7)

Structure of a #2 script (page 9)

Structure of a #3 script (page 13)

Error Message Handling in PHP Scripts (page 15)

Page Customization

Adding a Page to the Menu (page 18)

Adding a Field to a Page (page 20)

Building a Page using WolffPack Conventions (page 25)

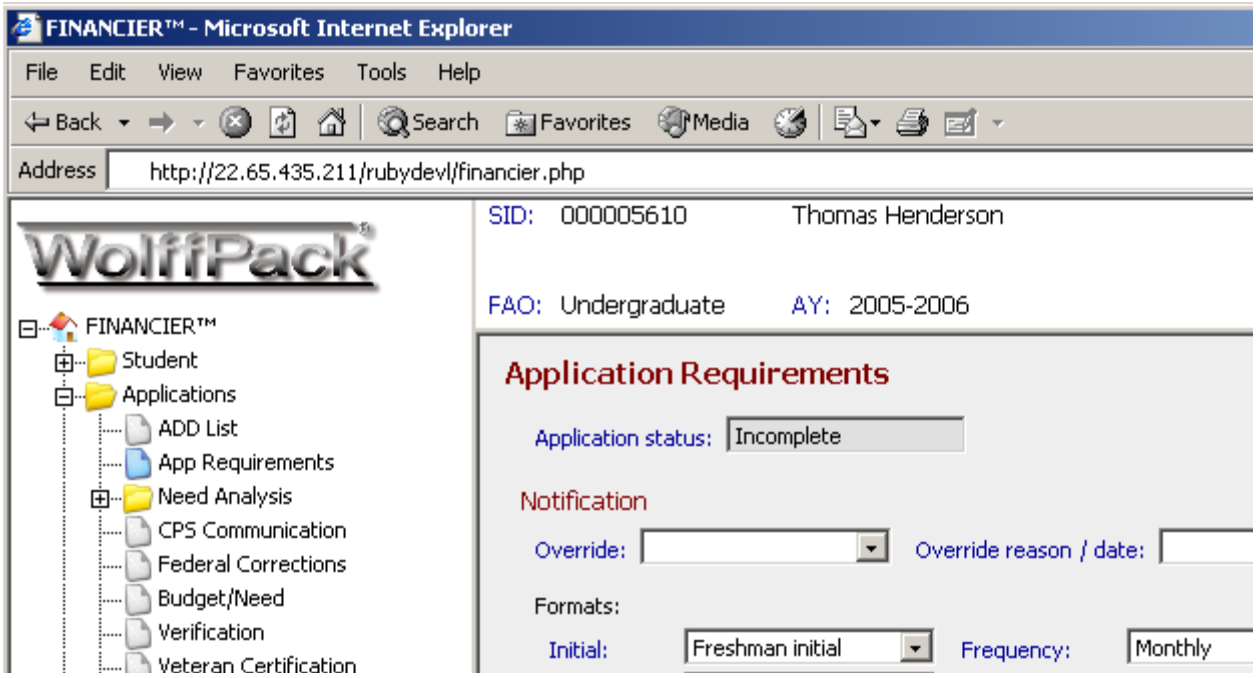
WEB-BASED ARCHITECTURE

The Browser Presentation Blueprint

FINANCIER's page presentation combines consistent appearance and navigation designed for utility and convenience, while providing the flexibility to allow for clients' choice of style sheets.

At the visible level, the structure consists of four frames. On the left, headed by a logo panel, the navigation area contains a tree structure providing access to the functional pages. At the top right, the context area displays record key information, with buttons to allow for data selection and navigation functions. The application page, with display/input forms and processing buttons, occupies the main portion of the screen on the lower right.

The structure, within an Internet Explorer browser window, looks like this:



Both the navigation frame on the left and the application area are scrollable. Application forms can be defined as any length.

Basic Structure

At a basic level, the four frames are defined as two, left and right. The left contains a fixed-size top area for the logo and a variable-size bottom area for the navigation structure display. The right also contains a top and bottom, the top being for the context and the bottom for the application display.

While the programmer generally doesn't need to think about the frames when coding, the names of the parts are:

- WPleft – the left portion

- WPc – left upper (logo)
- WPd – left lower (navigation)

• WPright – the right portion

- WPa – right upper (context and navigation buttons)
- WPb – right lower (application page and page processing buttons)

Frameset Components and Left Side Frames

The following pages and scripts define the overall frameset and make up the functionality of the logo and navigation tree. All of these pages are located in the default (top-level) directory of the web server.

File or Script	Purpose	Routes to (next)
index.htm	Presents “banner” page	logon.php
logon.php	Presents form for logging on to FINANCIER	validatelogon.php
validatelogon.php	Accesses SECURITY table to verify userid and password	fail: logon.php *or* pass: financier.php
financier.php	Creates initial frameset of WPleft and WPright	financier_index.php (WPleft) *and* financier_init.php (WPright)
financier_index.php	Creates initial frameset of WPc and WPd (in WPleft)	financier_top.php (WPc) *and* financier_menu.php (WPd)
financier_init.php	Invokes and displays initial application page	
financier_top.php	Contains logo	
financier_menu.php	Invokes and displays menu structure (via JavaScript function call)	

These pages and scripts are universal. That is, they hold for the entire browser session. They display the available system functions for the user.

The navigation structure groups the system’s functional pages into folders of related functions. For example, the user may choose Award Summary under the

Award folder. This choice invokes specific scripts for that function and causes the selected page and context to display in the right side frames.

Right Side Frames

What the user actually sees after choosing an item from the navigation tree depends directly on the functionality involved, but the basic structure used to serve the information is consistent and predictable. In the context (upper portion), student-based functions will display the FAO, student ID and name, while fund-based functions will show the FAO, fund ID and name. If the function selected is aid-year-specific, the aid year is displayed as well. Buttons appearing here are those that pertain to the entity in context (Event, Notepad) or serve general navigation purposes (Select, Help).

The application page that occupies the space beneath the context frame can be a typical form, with text literals, entry fields, dropdown lists – or anything else. If it can be displayed in a browser, it can be displayed here. A typical FINANCIER page is a form that consists of one or more "formlets." Formlet is WolffPack's term for a set of related fields from one table or view that make up a portion of a form. Most pages consist of a single formlet, because a typical page displays data from a single table or view in the database. Examples of pages that include multiple formlets are the Award Summary and FAO pages.

At the top right of the application page, based on the user's security clearance, are the buttons that allow updates, deletions and other functions specific to the page. Of course, display-only functions have no need for the Add, Update and Delete buttons, so these will never appear on such pages no matter what security clearance the user has.

Logically the tasks involved in serving the application information break into three steps:

1. Show the appropriate context and standard buttons.
2. Show the functional page filled with data from the database, with appropriate action buttons.
3. If necessary, update the database with the user's input on the form.

Each step corresponds to a PHP script (program). The #1 script (named `scriptname1.php`) handles the context and buttons. The #2 script (`scriptname2.php`) displays the form/page. The #3 script (`scriptname3.php`) performs the database update.

Obviously, the #2 and #3 scripts are specific to the page's function. The #1 script, however, since it controls what is displayed in the context area, can be applied (with a little modification) to all the functions that require the same record identification information – typically those functions grouped in a folder (such as Student, Application, Award, Fund) on the navigation menu.

The "conduit" script by which the #1 and #2 programs are invoked is `route.php`, which ultimately receives the name of the target script (minus the digit 1 or 2) and issues the necessary HTML to invoke both the #1 and #2 scripts. There is no guarantee that the web server will run the #1 script first, nor is the

order relevant. However the #3 (update) script is, and must be, invoked from the #2 script, as the update depends upon data displayed by the #2 script.

Context and Session Variables

Context is simply the current values of a number of key fields, such as the student ID, aid year, FAO, and so on. These fields are needed to retrieve and update data, and must be saved across the user/browser interaction. In PHP, the concept of "session variables" enables the storing of persistent context data. These variables remain available for the browser session. Once the browser is closed, they disappear.

WolffPack's session variables begin by convention with `SV_`. These variables represent context data, and include critical information communicated between the scripts, such as error messages.

You can view the current content of any session variable in FINANCIER on the Utilities>Session Vars page.

For more detail

Structure of a #1 script (page 7)

Structure of a #2 script (page 9)

Structure of a #3 script (page 13)

Error Message Handling in PHP Scripts (page 15)

WEB-BASED ARCHITECTURE

The #1 script controls the context display and navigation buttons in the top right frame in the browser presentation of FINANCIER.

Annotated #1 Script

Following is the #1 script for the Local Information page in FINANCIER, with notes that explain some of the common characteristics of the #1 script. You can adapt this #l script when adding a page for institutional data to FINANCIER.

```
<?php
session_start();  (1)
?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <META content="text/html; charset=windows-1252" http-equiv="Content-Type">
  <STYLE>.but{border:1px buttonface solid; background-color: white; margin-
top:2px;margin-left:2px;}</STYLE>
<?php
// Include file used to supply standard PHP functions used by #1 scripts
include ("../inc/inc_for_script1.php");  (2)

// JavaScript function used to supply page specific help
echo '<SCRIPT>' . $eol;
echo 'function help() {' . $eol;
echo '  contexthelp(' .$quote . 'demographic'  .$quote . ');' . $eol;  (3)
echo '}' . $eol;
echo '</SCRIPT>' . $eol;
?>
</HEAD>
<BODY class="navigation2">
<?php                                                                  (4)
include ("../inc/inc_context_open.php");   // Include file used to supply standard
button opening logic

/* Navigation buttons */
$gen_event = 'Y';  // (generate event button)
/* Display options */
$display_student = 'Y'; // (display student specific detail)
$display_aidyear = 'Y'; // (display AidYear information)

/* Context table */                                                    (5)
include ("../inc/inc_context_local.php");  // Include file used to supply context
info (specific to 'folder')

include ("../inc/inc_context_close.php");  // Include file used to supply standard
button close logic  (4)

?>
</BODY>
</HTML>
```

Notes:

1. The `session_start();` provides access to the session variables from within the PHP script.
2. The PHP Include statement invokes a set of standard utilities stored in the "include" file `inc_for_script1.php`. Include functions can be nested; for example, `inc_for_script1.php` employs references to `inc_js` and `inc_css`, which contain client-side JavaScript utilities and cascading style sheets respectively.
3. The page-specific JavaScript function `contexthelp()` supports Page Help.

   `contexthelp()` is defined in the the `inc_js` file, along with a default Help function, `help()`. `contexthelp()`, when coded in the #1 script, overrides the default. The name of the PAGEHELP topic, as defined in the FINANCIER Dictionary, must be passed as a parameter to the function `contexthelp()`. For example, PAGEHELP.local is the Dictionary entry for the Local Information Page Help topic, so "local" is the contexthelp parameter.
4. The Include file `inc_context_open.php` and its complement, `inc_context_close.php,` are open table and close table scripts to invoke the page-specific buttons (Update, Delete, etc.).
5. The `inc_context_local.php` script contains the specific context requirements for the folder, in this case the Local pages. There is a context Include file for each subsystem, `inc_context_foldername,` named to specify the folder in which the page resides.

   The Include file has the following lines to display the student ID, name and FAO:

```
  echo '        <TD align=right width=10%>Student: </TD><TD align=left width=15%>
<span class=context align=left>' . $sv_st_sid . '</span>
</TD><TD colspan=1><span class=context>' . $sv_st_name . '</span></TD>';
  echo '        <TD align=right>FAO/AY:</TD><TD align=left>
<span class=context>' . $sv_fao . ' </span></TD>';
```

See also

Structure of a #2 script (page 9)

Structure of a #3 script (page 13)

Error Message Handling in PHP Scripts (page 15)

WEB-BASED ARCHITECTURE

The display of the application functional page is the responsibility of the #2 script.

Annotated #2 Script

Following is the #2 script for the Local Information page in FINANCIER, with notes that explain the generalized data retrieval and form presentation process. You can adapt this #2 script when adding a page for institutional data to FINANCIER.

```
<?php
session_start();

echo '<HTML>';
echo '<HEAD>';
// Include file used to supply standard PHP functions used by #2 scripts (1)
include ("../inc/inc_for_script2.php");
echo '</HEAD>';
echo '<BODY language="JavaScript" onbeforeunload="return checkFormStatus(document.forms[0])"
OnLoad="document.recordindx.f1.focus()" onFocus="return checkModal()">';
                                                                        (2)

// Include for student record construction (Interface)
include ("../inc/inc_fn_buildstu.php");

// Error checking include for #2 program
include ("../inc/inc_error_check.php"); (3)

$high_offset = 0;
$high_tab = 0;

echo '<SPAN  style="position:absolute; left:' . $sv_screen_indent . 'px;  top:' . $high_offset . 'px"
echo '<form method="' . $sv_get_post_method . '" name="recordindx" action="sample_localinfo3.php">';

                                                                        (4)
// Invoke student interface (due to demographic data being on page)
build_sturec_demog();

// In this sample, two 'formlets' are used. The first accesses the  STUDENT table and the second
//   access the LOCAL_DATA table.  The PHP function getdata_buildform reads the FORM_DEFN table
//  for the passed formlet ($formlet)  and appropriate database table and/or view ($table_or_view)
//  and builds the HTML for the formlet.
//
/* STUDENT table */
$formlet = "w_localinfo_student";
$table_or_view = "STUDENT";
$key_cols = "fao:fin_key_s";
check_key_cols($key_cols, $missing_key, $missing_key_data, $missing_key_data_str, $formlet);
$rtn_array = getdata_buildform($formlet,$table_or_view,$key_cols,0,0);
$high_offset = $rtn_array[0];
$high_tab = $rtn_array[1];

/* LOCAL_DATA table */
$high_offset = $high_offset + 10;
$formlet = "w_localinfo_local_data";
$table_or_view = "LOCAL_DATA";
$key_cols = "fao:aid_year:fin_key_s";
check_key_cols($key_cols, $missing_key, $missing_key_data, $missing_key_data_str, $formlet);
$rtn_array = getdata_buildform($formlet,$table_or_view,$key_cols,$high_offset,$high_tab);
$high_offset = $rtn_array[0];
$high_tab = $rtn_array[1];
```

```
echo '<INPUT type="hidden" name="db_function" value ="';
echo $db_function . '" method="' . $sv_get_post_method . '">' . $eol;
echo '</form>';

echo '</SPAN>' . $eol;


include ("../inc/inc_buttonstart2.php");  // Include file used to supply standard button opening
logic
if ($missing_key == 'Y') {
  missing_key_alert($missing_key_data, $missing_key_data_str);
} else {
  include ("../inc/inc_button_update2.php");  // Include file used to supply standard update but-
ton logic
  include ("../inc/inc_button_delete2.php");  // Include file used to supply standard delete but-
ton logic
}
include ("../inc/inc_buttonend2.php");  // Include file used to supply standard button close
logic

// Call the function within inc_fn_error to display any error that
// getdata_buildform may have encountered.
display_errors_if_any();

echo '</BODY>';
echo '</HTML>';
?>
```

(5) (6) (7)

Notes:

1. The Include file `inc_for_script2.php` supplies standard database and utility function access. It also contains the standard "button" javascript function for handling the processing of the buttons. If you need to override the standard button function, your own "button" javascript function can be provided in your script. ***Important:*** this function, if you define it yourself, must appear *after* the Include statement for `inc_for_script2.php`.

   `inc_for_script2.php` employs references to `inc_js` and `inc_css`, which contain client-side JavaScript utilities and cascading style sheets respectively; and to `inc_fn_db_utils.php, inc_fn_utils.php and inc_fn_get_build.php` for database access, error handling, field masking and form building routines.

2. In the <BODY> tag, the onLoad action causes the specified field to be the initial focus of the cursor. The desired field will almost always be the first on the form, named by the tablename followed by 0 to designate the first field, such as `tablename_0`. The fields on a formlet are processed via a PHP array, starting with the index at 0.

   The onBeforeUnload action will check, when a form is being exited, to see if any changes were made on the form and not saved to the database. If so, a confirmation dialog will appear and the user will be able to cancel the exiting and save the changes or confirm that the changes should be discarded.

3. When cycling back through the #2 script from the #3 (update) script, it's necessary to check for errors encountered in the #3 script. Any such errors are displayed prior to re-displaying the page's formlets; otherwise (when returning

from the #3 script with no errors), the relevant session variables are initialized. The code to perform the error check and display the errors or initialize the variables is in `inc_error_check.php`.

4. The name of the #3 script must be included in the <form> tag as the object of the `action=` qualifier.

   The `getdata_buildform` function (`inc_fn_get_build.php`) retrieves the form (formlet) to be used from the FORM_DEFN table, and retrieves and maps the data from the table or view to the form (formlet). It uses absolute positioning to situate the labels and fields on the form.

   This function expects that the calling #2 script will render the beginning and ending tags of the form (<form>...</form>). Parameters are:

   - **pm_form** (required) is the name of the form in the FORM_DEFN table. (Passed in this sample script as `$formlet`.)
   - **pm_table_view** (optional) is the table or view from the database that will be used to populate the form. If empty, deafult is to use the first part of the dataname of the first column on the form. (Passed in this sample script as `$table_or_view`.)
   - **pm_where** (optional) is a colon-separated list of the columns to be used in the where clause. If empty, looks for a label on the form in the format `SQL:colname1:colname2:` etc., and takes those column names for use. If neither the parameter nor the SQL label exist, it's assumed there is no where clause. (Passed in this sample script as `$key_cols`.)
   - **pm_top_offset** (optional) is the offset from the top of the frame, where rendering should begin; defaults to zero. It is used to allow multiple formlets to be displayed consecutively using absolute positioning.
   - **pm_tab_offset** (optional) is the offset from 0 where the tabindex should begin; defaults to zero. It is used to allow multiple formlets to be displayed consecutively using absolute positioning.
   - **pm_db_data** is a true/false indicator; defaults to true, meaning this formlet is intended to read data from the database. The "false" setting would pertain in situations where it is necessary to create a formlet from the virtual data definitions in the Dictionary and other such "definitional" tables, where no data is retrieved before the fields are displayed.
   - **pm_name_offset** is used only when multiple formlets reference the same table or view in the same form; defaults to zero. Use an arbitrary number here, such as 100, greater than the number of fields on the previous formlet, to add that number to the html_names in this formlet. Otherwise, there would be a collision of the html_name and the js_define across formlets.

   The function returns an array of **high_offset** (highest top_px vertical offset from this formlet) and **high_tab** (highest tabindex used by this formlet), the information needed to position the next formlet, if any, on the form.

5. The `db_function` is a hidden field representing the value of a button pressed in the WPa (context) frame. It is defined on the form so that the #3 script will automatically receive its value when the form variables are passed along by the web server.

6. To supply the button generating code, the Include files

`inc_buttonstart2` and `inc_buttonend2` must enclose any other button code Include files that are appropriate to the page.

This button code does not use absolute positioning, so the buttons will be presented at the top right of the form. For this reason you must be careful not to display any non-absolute positioned HTML above this code, or the buttons will be displayed further down the form.

7. Finally, check for errors encountered by this #2 script and display them, via the `display_errors_if_any()` function.

See also

Structure of a #1 script (page 7)

Structure of a #3 script (page 13)

Error Message Handling in PHP Scripts (page 15)

WEB-BASED ARCHITECTURE

The #3 script performs the database update, insert or delete requested by the user from a FINANCIER application page. An update is done if a row with the given key existed previously, otherwise an Insert statement is issued. A delete is done if the Delete button is pressed.

Annotated #3 Script Template

Following is the #3 script for the Local Information page in FINANCIER, with notes that explain some of the common characteristics of the #3 script. You can adapt this #3 script when adding a page for institutional data to FINANCIER.

```
<?php
session_start();

// Include file used to supply standard PHP functions used by #3 scripts
include ("../inc/inc_for_script3.php");

// $sv_rows_found = 0 if this is an insert, else > 0 (1, we hope)
// $sv_column_pass = an explodable list of columns for sql statement construction
// $sv_where_clause = the where clause used in the number 2 program.

// This page will only update data from the LOCAL_DATA table.  While data is displayed
(via a formlet)
//   from the STUDENT table, it is not updateable on the page.
$table_or_view = 'LOCAL_DATA';
$table_on_form = 'LOCAL_DATA';
if ($db_function == 'UPDATE') {
  if ($sv_rows_found["$table_or_view"] == 0) {
    $new_row = true;
    row_insert($table_on_form, $table_or_view);
  } else {
    row_update($table_on_form, $table_or_view);
  }
} else {
  if ($db_function == 'DELETE') {
    row_delete($table_or_view);
  }
}

// Check for existence of errors, and set the session variable accordingly,
// so that the #2 script can be informed of the error condition.
$sv_err_exec_ident = exec_ident($err_count);

// Set up refresh of data display by re-invoking the #2 script
echo  '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">';
echo  '<HTML>';
echo  '<HEAD>';
echo  '    <META content="text/html; charset=windows-1252" http-equiv="Content-Type">';
echo  '    <meta http-equiv="REFRESH" content="0; url=sample_localinfo2.php">';
echo  '</HEAD>';
echo  '<BODY>';
?>
</BODY>
```

Notes:

1. Most of the intelligence involved in the database update, insert, or delete logic resides in the `inc_for_script3.php` file specified in this template.
2. Parameter definitions:
   - `$table_on_form` is the database table specified in building the form. This information is used internally in the automated parts of the processing. `$table_or_view` is the actual database table to be updated. For production processing it is the same as the table on the form, but this parameter allows for testing on copies of tables.
   - A third parameter that is not shown in the example, `$name_offset` relates to the pm_name_offset parameter to `getdata_buildform` that is supplied in the #2 script when multiple formlets reference the same table. Use the same number here that you used in the #2 script for the same formlet. In general, you will put the Update or Insert function calls in the same order as the `getdata_buildform` calls in the #2 script.

   Note that the Delete function only needs the actual database table as a parameter.
3. The last ten lines or so of the #3 script allow it to pass control back to the #2 script. Place the name of the #2 script to invoke in the `url=` portion of this code. Without this, the #3 script will simply display an empty frame and the user would need to press the Back button or enter a URL in the address line of the browser. With this code, the #2 script will automatically be requested upon completion of the #3 script.

   The #3 script creates an array of the actual values from the form in a session variable. That way, the user's input can be saved and re-displayed in the event of an error occurring in the #3 script. The `exec_ident`, a pointer to the error messages that may have been produced, is also stored in a session variable.

   The update logic will set session variables automatically for maintained data fields that relate to session variables (drives off cross-reference in SESSION_VARS table). ***Important*****:** If any session variable data is maintained on a formlet, the routing logic cannot simply present the #2 script again, since the session variables displayed as context data will be incorrect in the #1 script. An example of this would be student name. This situation would require an If statement to reload both the entire WPright frame (both the #1 and #2 scripts).

See also

Structure of a #1 script (page 7)

Structure of a #2 script (page 9)

Error Message Handling in PHP Scripts (page 15)

WEB-BASED ARCHITECTURE

Error Message Handling in PHP Scripts

The capture, recording, and display of errors encountered in the course of retrieving, displaying and updating data on the FINANCIER functional pages – by the running of the #1, #2 and #3 scripts – are handled with a set of standard procedures and utilities.

Standard PHP Script Error Messages

If you write only PHP scripts, and if those scripts simply take advantage of the automated processes of displaying and updating data, then you can simply follow the script examples to ensure that errors are displayed to the user in the system's standard way. Refer to the two calls to `display_errors_if_any` in the #2 script and the call to `exec_ident` at the end of the #3 script. Those calls are the minimum requirement for using the built-in error handling for PHP scripts.

Messages in More Complex PHP Scripts

At times, it is necessary to write PHP scripts that require coding that is beyond the capabilities of the standard set of functions and utilities. Developers of these scripts will need to understand the syntax for calling the error message storage routine.

The following snippet of code shows a direct call to the `store_error` procedure (in `inc_fn_utils.php`). The example illustrates handling an error occurring in a database (SQL) statement.

```
$conn->db_stmt($stmt);
if ($conn->sql_errcode <> 0) {
  store_an_error($gbconn, 99999, 'My descriptive text',
       'My module', 'My module location',
       'My table name' , $conn->sql_errcode, $conn->sql_errmsg);
  }
```

The parameters to the `store_error` procedure, in required order, are:

- The "handle" of the database connection object – the variable name used when the "new dbobj" is defined. This is usually established within the `getdata_buildform` function used by the #2 script. You may set up a connection object yourself if necessary.
- The assigned error number, which associates the error with the Message Definition table (WP_MSG_DEFN) to allow access to detailed explanation and corrective action information when errors are displayed.
- The run-time message text. This allows the developer to give specific information for this particular occurrence of the message. Typical information might include identifying key fields and values to help pinpoint the offending data.
- The module name. Here you normally specify the file name of the PHP script or the name of the PL/SQL module involved.

- The module location. Here you normally specify the function within the PHP script (or the PL/SQL function or procedure) where the message was generated. This helps pinpoint the location where the message was generated during debugging.
- The table or view name. If this is a message resulting from a SQL statement, specify the table or view name involved. You may leave this parameter empty or leave the parameter out of the call altogether if a table or view name does not apply to this message.
- The SQL error code. If this is a message resulting from a SQL statement, specify the `$connection->sql_errcode` here; otherwise, leave the parameter out of the call.
- The SQL error message. If this is a message resulting from a SQL statement, specify the `$connection->sql_errmsg` here; otherwise, leave the parameter out of the call.

Enter an empty string for any parameter that does not apply in a particular message occurrence.

The database connection established in a script in persistent until the script relinquishes control to the browser. This means that the same connection is used during that period regardless of the number of times a “new dbobj” is requested. The advantage of this is that the developer does not need to be concerned with the timing and location of the establishment of a connection. So, if you have created a dbobj in your script, use it. If you do not have one, and you need to call the `store_error` procedure, go ahead and create a “new dbobj” before calling `store_error`. The message will be tagged with the same `exec_ident` as any other message from the same timeframe.

For more detail

Structure of a #1 script (page 7)

Structure of a #2 script (page 9)

Structure of a #3 script (page 13)

CUSTOMIZING FINANCIER

Any institutionally designed and developed web page can be hooked into the FINANCIER navigational structure; this is the simplest means for extending the system, and for many purposes may prove the most efficacious. You can also add fields to an existing page, and make use of WolffPack's generalized presentation scripts and form definition capability to add a page that takes advantage of the FINANCIER blueprint.

The base system has a menu entry, Local Information, intended as a locus for institutional pages. Under Local Information in the base system you'll find several sample pages which are used as the basis for the following procedures.

Adding a Page to the Menu (page 18)

Adding a Field to a Page (page 20)

Building a Page using WolffPack Conventions (page 25)

CUSTOMIZING FINANCIER

Adding a Page to the FINANCIER Menu

Any web page can be linked to the FINANCIER menu. For example, the base system contains a navigational link to the home page for FAA Access to CPS Online, providing access via the menu entry Local Information>Central Processor.

Setting up a menu link involves using the Page Maintenance function to register the page in FINANCIER, describe its web location and enter the parameters that determine its place on the menu.

>> To add a pre-existing page to the FINANCIER menu:

1. From the menu, select Utilities>Page Listing.

   The display lists all pages defined in FINANCIER.

2. Click the Add button for the Page Maintenance detail window.
3. In the Page Maintenance window, enter values to set up the navigational hierarchy.
   - In the Page field, specify the name to be displayed on the menu.
   - In the URL field, enter the page's address, relative to the FINANCIER installation. If the page file is in the directory for a Subsystem folder (such as Student or Local), specify the file name -- **samplepage.htm,** for example. If the page does not reside within the FINANCIER directory structure, enter the web address as you would enter it in the browser address line: **http://www.homepage.edu/samplepage.htm**
   - Set the Folder field to **Not a folder**.
   - In the Order field, indicate the order in which the page should be listed within its folder (01 for first, 02 for second, etc.).
   - Activate the page: set the Active field.
   - As the Subsystem, select the top-level folder under which the page belongs.
   - In the New Window field, select **New window** if the page should appear in a separate window. Leave this field blank if the page should appear in the FINANCIER page display area (WPright).
   - In the Parent field, specify the immediate parent folder.
4. Define the appropriate user access to the page. (See the procedure on defining security on page 180.)
5. To see the page or folder listed in the navigation tree, refresh the browser (or, depending on the security defined, log on as an authorized user).

Refreshing the browser rebuilds the JavaScript navigation tree (the FINANCIER menu) by reloading `financier.php`.

>> To add a folder to the menu:

1. From the menu, select Utilities>Page Listing.
2. Click the Add button for the Page Maintenance detail window.

3. In the Page Maintenance window, enter values to set up the navigational hierarchy.
   - In the Page field, specify the name to be displayed on the menu.
   - Flag the object as a folder (Folder field).
   - Indicate its order beneath the parent (Order field).
   - Activate the folder (Active field).
   - In the Parent field, specify the immediate parent folder.
4. Define the appropriate user access to the page. (See the procedure on defining security on page 180.)
5. To see the page or folder listed in the navigation tree, refresh the browser (or, depending on the security defined, log on as an authorized user).

Related Functions

>> To add a field to a FINANCIER page (page 20)

>> To build a page using WolffPack Conventions (page 25)

CUSTOMIZING FINANCIER

Adding a Field to an Existing Page

For system maintenance reasons, the recommended means for adding institution-specific attributes into FINANCIER is to add the column(s)/field(s) to an institutional database table and institutional page. The base system provides a locus for institutional pages, under Local Information on the FINANCIER menu, and a template Local Information page, designed to look and function like the base system pages.

The procedure involves modifying a table and a "formlet." A formlet is a structural component of a page that contains data from a single database table or view. The formlet(s) that compose a page are listed in the page's #2 script as values of the `$formlet` parameter(s).

Example: Adding an Attribute Field to the Local Information Page

To illustrate the procedure, we'll use the example of adding an attribute with explicitly defined values to the "State Grant" portion of the Local Information page. The field LD_INST_FLAG (data type varchar2) will be added to the LOCAL_DATA table and to the formlet `w_localinfo_local_data`.

Formlet names are mnemonic, based on the name of the page and/or table ID, so you can probably locate the correct formlet simply by name. To be sure, however, you can bring up the page in FINANCIER. Then, anywhere in the background of the page display, right-click the mouse and select View Source. When the source is displayed in Notepad (or your designated editor), search for the word "formlet". You should find a comment containing the formlet name at the beginning of the code for displaying each formlet.

Another way to find out what formlets are on a page is to the get the name of the #2 script for the page by referring to the URL specified in the page definition (go to Utilities>Page Maintenance, then click on the page name). The #2 script name consists of the URL entry followed by suffix 2 and file extension.php. For the Local Information page, the URL is "localinfo," so the #2 script is `localinfo2.php.` You can then check the #2 script for values of `$formlet.`

>> To add a field to a page, on an existing formlet:

1. Add the attribute as a column in the appropriate database table.

   For example, to add the column LD_INST_FLAG *to the* LOCAL_DATA *table, you would execute the following SQL:*

   ```
   ALTER TABLE LOCAL_DATA ADD (
     LD_INST_FLAG VARCHAR2(1) )
   ```

2. Add a Dictionary definition for the column. If the field has a specific value set, add the valid values. (See the procedure "Maintaining Dictionary Entries" on page 174)

3. Retrieve the formlet to which the field is to be added.

- From the menu, select Utilities>Formlet Listing.
- Click the name of the formlet.

The Formlet Maintenance page is displayed, listing the components of the formlet – fields, labels, selection lists, etc. – with their location coordinates.

For the formlet `w_localinfo_local_data`, *you'll see entries for the formlet label* State Grant*, the field labels* Eligible: *and* Amount: *and the database columns that correspond to the Eligible and Amount fields. The entries to be entered include the definitions for the field label* Inst flag: *and the field for* LOCAL_DATA.LD_INST_FLAG.

4. Click the Add button to define a label for the new field.

   The Formlet Object Maintenance window is displayed.

   - Specify the location (Top and Left pixels) and size (Height and Width pixels) of the label.

   The Top and Left pixel settings are relative to the beginning of the formlet. In our example, we'll locate the Inst. Flag: *label beneath the* Eligible: *label, which is situated 60 pixels from the top of the formlet and 20 from the left and measures 20 in height. Coordinates for the new label are set to Top Pixel of 90 and Left Pixel of 20, to align it with the one above while providing some space between them.*

   Standard setting for the Height Pixel of a field label is 20. Approximate width for a field label of 10 characters in length would be 20.

   - Select a CSS Class value to control the label's appearance.

   The CSS class refers to the style sheet characteristics to be employed. The value for a field label is **Simple Text Label.**

   - Specify the Object Type.

   The value for a field label is **Label.**

   - Enter the Label Text, consisting of an HTML <anchor> tag.

   The HTML anchor tag produces the label. It contains the literals that constitute the label text and JavaScript to invoke the Field Help function. For example,

```
<a href="javascript:field-
help('LOCAL_DATA.LD_INST_FLAG')" onMou-
seOver="window.status='Field Help'; return true"
onMouseOut="window.status=' ';return
true"class=label_1">Inst Flag:</a>
```

   - When information is complete click Update and close the window.

5. Click the Add button to define the new update field.

   The Formlet Object Maintenance window pops up.

   - Specify the location (Top and Left pixels) and size (Height and Width pixels) of the field.

   To situate the field next to its label in our example, coordinates are set to Top Pixel of 90 and Left Pixel of 100.

   Standard setting for the Height Pixel of an input/display field is 19. Width of 160 will accommodate value translations.

- Select CSS Class value to control the field's appearance.

The CSS class refers to the style sheet characteristics to be employed. The value for a field is **0 - Default.**

- Specify the Object Type and Tab Order.

The Object type for an input field with valid values is **drop down list.** *The tab order is adjusted according to the pattern in which you want the cursor to move from field to field.*

- Enter the TABLE.COLUMN ID as the Field Name and Data Name and specify the Data Type to accord with the database column definition.

For our example, LOCAL_DATA.LD_INST_FLAG *and varchar2.*

- Set the HTML Name to identify the table, with a unique numeric suffix.

The HTML name is needed for update processing and is passed between scripts. The two existing fields on the formlet are LOCAL_DATA_1 *and* LOCAL_DATA_2*; our new field is set to* LOCAL_DATA_3*.*

- Enter the JS Onchange parameters.

The Onchange javascript is used for data validation. The format for the parameter entry is: `define('HTMLName','datatype value','LabelLiteral',null,null,1)`

Or, for our example:

```
define('LOCAL_DATA_3','string','Inst
Flag:',null,null,1)
```

- When information is complete click Update and close the window.

Other Options for Adding Fields to a Page

For system maintenance reasons, we recommend using an institutional page whenever possible for customization. In the example above, none of the components would be affected by a WolffPack maintenance release.

You could use the same procedure to add column(s)/field(s) to a base FINANCIER table and page, which may in some cases seem preferable for functional reasons. However, if you choose this option, you will need to reapply your modifications whenever a maintenance upgrade affects the modified table or formlet.

A third possibility is to add the field(s) to an institutional database table and to a base FINANCIER page. This option would involve adding the fields in a separate formlet at the bottom of the page. You would be modifying (or adding) an institutional table, creating a new institutional formlet and updating the page's #2 script. You would need to reapply your script modifications whenever the affected script was updated.

>> To add a formlet to a page:

1. To add the database column(s), follow Steps 1 and 2 of the procedure for adding an institution-specific field (page 20).
2. To create the formlet, refer to the procedure for defining a formlet (page 25).

3. Modify the #2 script.

   - After the last formlet process, add code to display the new formlet. In the following example, the new formlet, which is named `w_localinfo_local_data_2`, involves data from the LOCAL_DATA table:

```
/* LOCAL_DATA table */
$high_offset = $high_offset + 10;
$formlet = "w_localinfo_local_data_2";
$table_or_view = "LOCAL_DATA";
$key_cols = "fao:aid_year:fin_key_s";
check_key_cols($key_cols, $missing_key, $missing_key_data,
$missing_key_data_str, $formlet);
$rtn_array = getdata_buildform($formlet, $table_or_view, $key_cols,
   $high_offset, $high_tab);
$high_offset = $rtn_array[0];
$high_tab = $rtn_array[1];
```

   - Modify the `$table_or_view =` statement to refer to the table associated with the formlet being added:

```
$table_or_view = "table_name";
```

   - Modify the `$formlet =` statement to refer to the name of the formlet being added:

```
$formlet = "your_formlet_name";
```

4. If the formlet is intended to update as well as display data, modify the #3 script:

   - After the last formlet process, add code to enable data updating. The `w_localinfo_local_data_2` formlet in the following example updates the LOCAL_DATA table:

```
$table_or_view = 'LOCAL_DATA';
$table_on_form = 'LOCAL_DATA';
if ($db_function == 'UPDATE') {
  if ($sv_rows_found["$table_or_view"] == 0) {
    $new_row = true;
    row_insert($table_on_form, $table_or_view);
  } else {
    row_update($table_on_form, $table_or_view);
  }
} else {
  if ($db_function == 'DELETE') {
    row_delete($table_or_view);
  }
}
```

   - Modify the `$table_or_view =` statement to refer to the table associated with the formlet being added:

```
$table_or_view = "table_name";
```

   - Modify the `$table_on_form =` statement. Typically this is the same table name. (This variable allows you to use a copy of a table for development/testing purposes.)

```
$table_on_form = "your_table_name";
```

Related Functions

>> To add a page to the FINANCIER menu (page 18)

>> To build a page using WolffPack Conventions (page 25)

CUSTOMIZING FINANCIER

Adding an Institutional Page

To add an institutional page, you can use the formlet maintenance capability to produce web forms in the FINANCIER style, and the page presentation script templates to create the page's processing scripts.

The following procedures assume you are starting with the Local Information page scripts to produce a page to display and update information from an institutional table using an institutional formlet. We'll use the delivered Local Data page and w_localinfo_local_data formlet to provide examples.

>> To define a new formlet:

Before creating a formlet, plan the content and layout, including the page title, data fields and field labels. Add new fields to the database and Dictionary.

1. Go to the Utilities>Formlet Listing page. Click the Add button.

   The Formlet Add window is presented.

2. Enter the formlet name, with top and left pixels to determine the position of the first object on the formlet (via absolute positioning).

   The first object can be a page title, section title, column heading, data field label or data field. On a formlet that is to be at the top of the page (or the only formlet on the page), the first field will probably be the page title. The typical entries for top and left pixel for a page title are 9 (top) and 3 (left).

   The top pixel of the first field on any formlet is always defined relative to zero. If there is more than one formlet on a page, the form-building process (`getdata_buildform`) positions them for display according to the variable `$high_offset` that is passed when `getdata_buildform` is called in the #2 script.

3. Click the Update button in the Formlet Add window.

   The Formlet Listing is redisplayed. The new formlet name is included in the listing (in alphabetic order).

4. Click the formlet name. When the Formlet Maintenance window pops up, click the "update" link to define the first component on the formlet.

   The Formlet Object Maintenance detail window is presented, with the location defined by the top and left pixel coordinates you assigned.

5. If this is the first or only formlet to be on the page, define the first component to be the page title.

   For a page title, supply

   - Absolute positioning coordinates for Top and Left pixels (if not entered when you created the formlet); the offset from the top and left of the formlet.
   - The Height and Width pixels to establish the size and length. Typical height value for the title on a FINANCIER page is 13; width depends on length of the text. For comparison purposes, width for the title "Local Information" is 219.

- CSS Class value of **3 - Form heading text** to control the title's appearance.
- Object Type of **label**.
- Data Type of **varchar2**.
- Label Text, consisting of an HTML <anchor> tag containing the literals that constitute the title and JavaScript to invoke Page Help; for example `<a href="javascript:pagehelp()" class="label_3">Local Data</a>`

When the title characteristics have been entered, click the Update button and close the Formlet Object Maintenance window.

The Formlet Maintenance window is redisplayed, with the object information that you entered.

6. For each field to be included on the formlet, define a field label and a data field.

   To create each new component,

   - Click the Add button on the Formlet Maintenance window
   - Enter the component's characteristics
   - Click update and close the window

   If the formlet is updatable, it must contain the (invisible, unlabeled) field UPD_CHK from the table associated with the formlet, which helps manage updating of the database.

   For the UPD_CHK field, by convention the first object on the formlet beneath the page title, supply

   - Absolute positioning coordinates for Top and Left pixels, defining the offset from the top and left of the formlet. On the w_localinfo_local_data formlet, UPD_CHK is the first object on the formlet, located around 10 (top) and 10 (left).
   - The Height and Width pixels to establish the size and length. Typical height value for UPD_CHK on a FINANCIER page is 10; width is 10.
   - CSS Class value of **0 - Default**.
   - Object Type of **simple text field**.
   - Field Name and Data Name, both consisting of the TABLE.COLUMN identifier. These are used to inform the #2 script which table/view and column to process.
   - Data Type is **number** (corresponding to the data type in the field's database definition).
   - HTML Name, a unique identifier for form processing. WolffPack convention is *tablename_seq*, where seq is a sequence number. Use sequence number zero for the UPD_CHK field; for example, LOCAL_DATA_0.
   - JSOnchange, for generating a JavaScript onchange event script. Format is `define ('HTMLname','type','label',mask,min,max)`, using the HTML name assigned, a *type* value of string, num or email, and the field label text as the *label* literal. *mask*, *min*, *max* are optional parameters, to specify an edit mask, if edit mask validation to be performed by JavaScript, and minimun and maximum length of input. For example: `define`

`('LOCAL_DATA_0','num','LOCAL_DATA.UPD_CHK',null,null,2)`. (For numeric fields such as UPD_CHK, the edit mask parameter is null; the database performs the validation.)

For a field label, supply

- Absolute positioning coordinates for Top and Left pixels, defining the offset from the top and left of the formlet. For the first data field on a page (depending on the size of the formlet and how many fields are to be included), the label might be located at 60 (top) and 20 (left). The coordinates for subsequent fields and labels can be estimated based on the location of adjacent fields.
- The Height and Width pixels to establish the size and length. Typical height value for the label on a FINANCIER page is 20; width depends on length of the label text. For comparison purposes, width for the label “Name:” is 44.
- CSS Class value of **1 - Simple text label** to control the title’s appearance.
- Object Type of **label**.
- Data Type of **varchar2**.
- Label Text, consisting of an HTML <anchor> tag containing the literals that constitute the field label and JavaScript to invoke Field Help; for example, for the label Eligible the entry is `<a href="javascript:fieldhelp(‘LOCAL_DATA.LD_ST_GRANT_ELIG’)” onMouseOver=”window.status=’Field Help’; return true” onMouseOut=”window.status=’ ‘; return true” class=”label_1”>Eligible:</a>`

For an updatable field, supply

- Absolute positioning coordinates for Top and Left pixels, defining the offset from the top and left of the formlet. For the first data field on a page, if the label is located at 60 (top) and 20 (left), the field itself might be located at 60 (top) and 100 (left). The coordinates for subsequent fields can be estimated based on the location of adjacent fields.
- The Height and Width pixels to establish the size and length. Typical height value for a data field on a FINANCIER page is 19; width depends on maximum length of values or value translations. For comparison purposes, width of a 30-character field is 132.
- CSS Class value of **0 - Default** to control the field’s appearance.
- Object Type of **drop down list** if the field has Dictionary-defined values, or **simple text field** if input is freeform.
- Tab order; a number to indicate the order in which the cursor should proceed to this field.
- Field Name and Data Name, both consisting of the TABLE.COLUMN identifier. These are used to inform the #2 script which table/view and column to process.
- Data Type corresponding to the data type in the field’s database definition.
- Edit Mask if needed to format input; use Oracle formats.

- HTML Name, a unique identifier for form processing. WolffPack convention is *tablename_seq*, where seq is a sequence number. For example, LOCAL_DATA_2 is the HTML name assigned to the second field on the w_localinfo_local_data format. Use sequence number zero for the UPD_CHK field, use the
- JSOnchange, for generating a JavaScript onchange event script. Format is `define ('HTMLname','type','label',mask,min,max)`, using the HTML name assigned, a *type* value of string, num or email, and the field label text as the *label* literal. *mask*, *min*, *max* are optional parameters, to specify an edit mask, if edit mask validation to be performed by JavaScript, and minimun and maximum length of input. For example: `define('LOCAL_DATA_1','string','State Grant Eligible:',null,null,1)` or `define('LOCAL_DATA_2','num','Amount:',null,null,6)` (Note that for numeric fields, the edit mask parameter is null; the database performs the validation.)

7. When all components have been defined, review the formlet layout: press the Preview button (the gear icon). Revise positional settings as needed.

To display a formlet as a FINANCIER page, you need to define the page and create the display/update scripts.

>> To add a page to FINANCIER:

1. Create the formlet(s) for the page, using the “Define a new formlet” procedure (page 25).
2. Copy the presentation script templates `/local/localinfo1.php, localinfo2.php` and `localinfo3.php` and rename them for the new page.
3. Add a PAGEHELP entry in the Dictionary–for example, **PAGEHELP.newpage**–with an Extended Description to document the new page.
4. Edit the #1 script for the new page.
   - Change the `contexthelp` parameter, replacing the Page Help identifier ‘local’ with the ‘*newpage*’ identifier from Step 2.
   - Set the appropriate context buttons. The Help, Select and Notepad buttons are displayed by default; if you don’t want one of these on the page, insert into the script `$gen_help = 'N', $gen_select = 'N'` or `$gen_notepad = 'N'.` If you don’t want the Event button to display, remove the statement `$gen_event = 'Y'` from the script.

   The context area will be governed by the specifications in `inc_context_local.php` to include the FAO, aid year and student name, ID, status and tag.
5. Edit the #2 script for the new page.
   - If you’ll be displaying information from the STUDENT table, retain the Include statement for `/inc/inc_fn_buildstu.php.` Otherwise, you can remove this statement.

- In the <FORM> tag, modify the `action=` qualifier to point to your #3 script.
- For each formlet to be displayed on the page, duplicate or modify the code for the formlet specifications, including at least

```
$formlet = "formletID";
```

```
$table_or_view = "table/viewID";
```

(Use a table when possible; while views may be more powerful, there may be Oracle issues attending the updating of rows through a view.)

```
$key_cols = "keycolumn1:keycolumn2:keycolumn3";
```

(Specify physical column IDs in a colon-separated list.)

- Check the `getdata_buildform` parameters.
- Check the button function Include files (`/inc/inc_buttonstart2.php,` etc.); add or remove as appropriate. You may need to override the default JavaScript executed by the button files.

6. If the data can be updated, edit the #3 script for the new page.
   - Modify the `$table_or_view` and `$table_on_form` variables to specify the table/view name.
   - Modify the <META> tag to point to the #2 script. (This reinvokes the display script to re-read the data and present the form as updated.)
7. Add the page to the FINANCIER menu. Follow the steps in the procedure for editing the menu (page 19).

Related Functions

>> To add a page to the FINANCIER menu (page 18)

>> To add a field to a FINANCIER page (page 20)