The PHP framework

A programmers guide

www.tfyh.org

Spring 2021

Table of Contents

Foreword

In 2017 I started to do some PHP programming and found out that this is a great option to support any form of business administration. I started to build a club administration tool which I introduced in my rowing club when they asked me to run the membership administration. It was well adapted by the club mambers since they now could manage their data themselves. By and by more functions were added such as booking boat trailers or providing logbook data.

The other tasks came by. Another club, a youth group and an event booking application in Corona times. Yet another club, a funding thing.

In order to be able to run that all I harmonized some of the PHP code and the design of all these applications. That code is explained here for the purpose of being able to reuse it.

Bonn, Spring 2021

Martin Glade

Licence consideration

All I do is published under the GNU public license V2.

System prerequisites

Files will run on any PHP interpreter but need a MySQL data base for data storage to run with.

A word on the contents

The harmonized code come in section depending on what it intends to do. Each section is then a folder on the PHP server, so that all filles can be addressed from all other files by ../[section]/[file.php]. That is limited in scaling but very easy in handling.

tfyh - framework classes

Backup_handler

The backup_handler provides a means to create a backup of all tables of the connected database in the form of csv-tables, one file per table, zipped into one archive. It will locate all backups into the provided $log_dir/backup directory which shall be existing.

backup()

Creates a backup. Ten backups in a row are indexed, different archive files, and then start with the first index over again. At ech roll over, the existing first backup will be renamed to become a secondary backup file. Secondary backup files are also indexed 0 .. 9 and rolling over back to 0 after 9. Then, the secondary file with index 0 is overwritten.

The backup can be configured by the application configuration to be mailed to a predefined mailbox. To trigger the sending, set an application parameter within the Parameter table named "Backup_Mailbox" to an email address, e.g. john.doe@trueme.org. The archive will be sent as base64 encoded text file. In order to prevent such a mail attachement from unauthorized access it can be masked by xoring the base64 encoded archive using a key. This key by default is a 64 bit base64 sequence. You may use a different one by setting an application parameter within the Parameter table named "Backup_Mask".

unmask()

unmasks an existing backup in order to be able to read it. Provide the filename of the masked backup, the mask and the binary zip will be returned, ready to be unzipped then.

Config

A utility class to hold all application configuration. There are three layers of config data.

1. Application constants which are part of the code and configure the framework (code constants),
2. data base stored parameters which are typical for the app's function and (values within a config table)
3. administrative parameters which are typical for the tenant using the application (the setgtings_app and settings_db file)

The config class reads all and provides variable access to them for other classes. It is automatically instantiated by the Toolbox class shall and only be accessed via the toolbox.

The expected settings are currently:

- $app_name;
- $changelog_name;
- $parameter_table_name;
- $pdf_footer_text;
- $pdf_document_author;
- $pdf_margins;

get_cfg()

Provides all configuration as associative array $key => $value.

set_cfg()

Copies the provided associative array $key => $value into the memory configuration. This deletes all previously read settings. Shall only be used by the configuration change form.

Tfyh_cron_jobs

A static class container file for a daily jobs routine: backup and log cleansing.

This class shall be extended by an application specific Cron_jobs class which then can add application specific tasks such a record deletion for data privacy reasons.

run_daily_jobs()

performs the standard tasks. It may be triggered by whatever, checks whether it was already run this day and if not, starts the sequence. So you may trigger this with any specific user or api action like the login.

Form

This class provides a form segment for a web file.

Form definition file

The definition must be a CSV-file, all entries without line breaks, with the first line being always "tags;required;name;value;label;type;class;size;maxlength" and the following lines the respective values.

The form shall always displayed as a responsive grid. Use the "tags" definition section to provide the needed tags, which define the grid.

Here is an example:

```
tags;required;name;value;label;type;class;size;maxlength
<div class='w3-row'><div class='w3-col
l2'>;*;Bezeichnung;;Bezeichnung;text;;25;64
</div><div class='w3-col l2'>;*;OrtID;;Veranstaltungsort;"select
list:select:1";;20;
</div></div><div class='w3-row'><div class='w3-col
l3'>;*;AnzahlEinzel;;Anzahl Einzelplätze;"select
```

```
0=0;1=1;2=2;3=3;4=4;5=5;6=6;7=7;8=8;9=9;10=10;11=11;12=12";;5;5
</div><div class='w3-col l3';*;AnzahlPaar;;Anzahl Partnerplätze;"select
0=0;1=1;2=2;3=3;4=4;5=5;6=6;7=7;8=8;9=9;10=10;11=11;12=12";;5;5
</div></div><div class='w3-row'><div class='w3-col l1'><div
style='float:right';;submit;Platzkontingent jetzt
ändern;;submit;formbutton;;
</div></div></div>;;_no_input;;;;;;
<li><span class='helptext';;_help_text;;Felder mit einem * müssen einen
Eintrag enthalten.;;;;
</span></li>;;_help_text;;;;;;
```

The form definition file is located in the ../config/layouts folder and shall carry the same name as the php file which uses this form without the extension. In multistep forms, steps 2 through x forms must use the layout name [form]_[step], e.g. lyaout #1 is 'registration', then #2 is 'registration_2'.

Form input elements

Following input trypes do not have any configuration at all except their size and maxlength properties:

- Input type **text**
- Input type **email**
- Input type **date**

Following elements can be chosen for a form input and have a special handling then:

- Input type **select**: set type to "select value1=display1;value2=display2 ...". You can define the selection options statically in the forms layout file or use the following **dynamic selection** definition options:
  - set the $select_options variable. Pass the select options to this String programmatically as array, e. g. ["y=yes", "n=no", "d=dunno"]. Syntax is “select $options” for the form layout.
  - set a parameter within the parameter table.
    Syntax is "select use:titles_choice"
  - use a data base list (see Tfyh_list class). Either the list set like for mail distribution list selction
    Syntax is "select list:listset" or
  - "select list:listset:listid[+]" [the ‘+’ adds ‘-1=(leer)’ as first entry]. In both cases the user must have the necessary privileges to read the list, if not a single error option is displayed.
- Input type **radio**: set type to "radio value1=display1;value2=display2 ...". Radio buttons are displayed one above each other, separated by the
-tag. You can use "**radioh** value=display1;..." to align them one next to each other. In that case each option is cased into a DOM element.
- Input type **textarea**: set size=count of rows, maxlength=width in characters. If maxlength is not provided, the full available width will be used.
- **Mandatory** entry: set "required" to "*" to trigger the validity check to expect an entry.
- Name **_help_text**: if the name equals '_help_text' it will not be returned in

get_html(), but rather in get_help_html(). This name can be used more than once.

- Name **_no_input**: if the name equals '_no_input' only the label is used. This name can be used more than once. Used to add explanations in forms.
- Name **#Name, @Name**: if the name equals '#Name' or '@Name' all available subscription (#) or workflow (@) names are used and the field is repeated for each. Use "#Titel: #Beschreibung" as input label or "@Titel: @Beschreibung" respectively.

Form usage

Form can have a single page or more. The latter provides the opportunity of branching dialogues, e.g. the registration depending on the age of the person who registers. The init pseudo-class provides a random number when a form is called which then can be reused to follow he sequence of entering data step by step. This ensures that data are linked to the same activity before being written to the data base, even if the user has multiple tabs with the same form open in parallel.

The form will always use the action "?fseq=[randomNo][$form_index]", so that re-enters the same php-page after form completion. The $form_index reflects the step that was done when entering the values.

The Form class reads data provided in the http POST request and fills an array with those. Reading includes replacement of "`" by the Armenian apostrophe and ";" by the Greek question mark. Characters look similar, but have different code points so that they will not be interpreted in their SQL-function such by any data base. To prevent from cross side scripting, "<" is replaced by the math preceding character.

To display a Form, create a PHP File using the "$done" and "$todo" indices.

The "$done"-value provides the information of the executed workflow step before the current http request, i. e. that one that delivers the POSTed values and rules the application logic. The entered values shall be read and validated in the first PHP-code part. Construct a Form using the done worklow step and run the "read_entered" function to read the values. Compile all generic errors via the "check_validity" function. Apply workflow specific application logic afterwards. Collect all further errors.

Now, in a second code part, the form and texts are displayed using the "get_html" function. If there were errors, can repeat the step by reusing the same Form instance. "get_html" will provide you with that form showing all entered values and red-flagged erroneous fields. If the validation succeeded (or upon workflow start), construct a new, empty Form for data entry. You should rule the page display code part by the "todo"-parameter.

Form functions

preset_value(), preset_values()

Set a single value or a set of values within a form prior to its display. Usage is typically for data record change forms. You create the form, preset the values of the existing record and displays it for the change action.

Note: preset_values() also works for #Name / @Name fields and subscriptions / workflows respectively.

get_html(), get_help_html()

Provide the html code to display the form in a page or the help text in a page.

read_entered()

Use read_entered() to read all POST parameters which match a Form field definition into the respective input field. All form values are stored in the $_SESSION superglobal array using a $_SESSION["forms"][$fs_id]-array.

check_validity()

This function checks all form inputs after they have been read against their syntax validity, e. g. valid EMail format, mandatory fields filled in asf. If errors appear it returns an error String, else it returns "".

set_validity()

This allows to run a validity check outside the form class and inform the form object of an invalid entry to make sure it does not continue with the wirkflow but rather displays the red border of the invalid field. Set the respective error message in the calling page.

get_entered()

Simple getter of all data read so far. Just returns a copy of the $_SESSION["forms"] [$fs_id] array, kept for backwards compatibility of the code.

init.php

init is not a class, but rather a standard include to all php files read. in other words: all PHP-files include at the very beginning init.php.

init does the session control, initialzes the Toolbox including the Config and Users classes, triggers load throttling, creates the menu and the form sequence identifier or picks it from the GET parameters passed in the request, pushes all $_GET paramnaters into the $_SESSION["getps"] container for the appropriate form sequence and initializes the data base access.

init.php also steers the end of the web page call by providing the end_script() function which must be called at the end of all PHP-files to close the data base connection appropriately. It logs the activities so that erroneous abortion of a PHP-file execution can be traced back. It registers a shutdown function for graceful exit in such cases.

Mail_handler

The mail handler provides a functionality to send simple html formatted mails to application users. It uses the tenant specific configuration as stired in the settings_app file for parametrization. The parameters are

- $system_mail_sender: mail address for system generated mails, including plain name, e.g. 'No-reply<noreply@domain.com>'
- $mail_schriftwart: mail address for copy recipient of workflow generated mails

- $mail_webmaster:mail address for getting information from users
- $mail_mailer: mail address for system generated mails on behalf of users
- $mail_subject_acronym: Acronym to prefix the subject line, e.g. "[YApp]"
- $mail_subscript:mail signature for system generated mails ("Yours sincerely A. Bee")
- $mail_footer: mail footer for system generated mails ("see www.abc.org")
- $system_mail_address: mail return address for system generated mail

The mail handler cares for proper encoding and provides helpful functions to avoid spam rejection such as multipart support to add a plain text to the html message. It supports up to two attachements. Added is further the option to save mails as text files.

send_mail()

This is th most frequently used function, providing the format and send capability. It returns true on success, an error on failure. Sounds simple and is it.

store_mail(), get_html(), get_last_index()

Provide a capability to save mails as text file whith appropriate indexing and read them back for display as HTNL. Thee different file paths can be used depending on whether the mails were sent individually, to a distribution list, or from the system. In the current implementations this is not used, but mails are stored in the data base.

Menu

The menu class does two things: provide a menu to the user and check whether the user is allowed to request a specific page. For the latter it uses methods of the Tfyh_user class.

Menu template file

Construct the menu from its template file. A template file is a flat file of menu items, starting with a programmatic name, the role, workflows and subscriptions which are allowed to use it, the display name and the link which is called when selecting the menu item. Menu items will be displayed in the sequence of the file and only, if the current user is allowed to use them. Level 2 item names must start with a "_". A menu can have 1 or two levels, not more.

The role, subscription and workflow are a comma separated list. Any of these which is preceded by a ".” will allow the access to the menu item link, but not display the item.

An application must have two menu definitions, pmenu for the public or anonymous access and imenu for the internal or authorized access. Both sit in ../config/access. init.php chooses which one to use based on the $_SESSION["User"] being set or not.

Here's a pmenu for reference:

```
id;permission;headline;link
Start;.Anonym;Startseite;../public/index.php
_Start_Datenschutz;.Anonym;Datenschutz;../public/datenschutz.php
_Start_Impressum;.Anonym;Impressum;../public/impressum.php
Buchen;Anonym;Buchen;../public/filter.php
```

```
_Buchen_Buchen;.Anonym;Buchen ;../forms/buchen.php
_Buchen_Ticket;.Anonym;Buchungsquittung ausgeben ;../pages/pdf_ticket.php
Storno;Anonym;Buchung stornieren;../forms/storno.php
Kontrolle;Anonym;Buchung nachsehen;../forms/kontrolle.php
Umbuchen;Anonym;Umbuchen;../public/umbuchen.php
Termine;.Anonym;Termine;../public/termine.php
Login;.Anonym;Einloggen ;../forms/login.php
Logout;.Anonym;Abmelden ;../pages/logout.php
```

Role hierarchy

The menu definition file refers to roles, workflows ans subscriptions. Roles have a hierarchy which is defined in the role_hierarchy file in the same location. An example is:

```
Anonym=Anonym
Besuchen=Besuchen,Anonym
Begruessen=Begruessen,Besuchen,Anonym
*Anbieten=Anbieten,Begruessen,Besuchen,Anonym
*Verwalten=Verwalten,Anbieten,Begruessen,Besuchen,Anonym
```

So "Verwalten" can do everything which is specifically allowed for "Verwalten" plus all what is alloed for any role in the list right to the '=' character. The asterisk points out that this is a privileged role. That means: in the list of access rights users having this role will be listed by name.

Workflows and subscriptions

Workflows and subscriptions are bit masks of 32 bits each bit being a flag for whether this workflow is allowed or not for a specific user. What a bit means is defined in the "../config/access/workflows" and the "../config/access/suscriptions" files, e.g.:

```
ID;Name;Titel;Beschreibung;Flag
1;Datenverwendung;Datenverwendung einschränken;Modifikation der
Widersprüche zur Veröffentlichung von Daten.;1
2;FahrzeugGenehmigen;Fahrzeug genehmigen;Befugnis eine Reservierung für ein
Fahrzeug des Vereins zu genehmigen.;2
3;TrainingDokumentieren;Training dokumentieren;Möglichkeit, freiwillig die
eigenen Trainingsleistung online zu dokumentieren.;4
```

The menu definition refers to a workflow by @[Bitmask] and to a subscription by #[bitmask]. The difference is that a workflow is meant to be assigned by an adminstrator to a user while subscriptions are set and removed in user self service.

Functions

get_menu()

Returns the html-code for the menu. Uses the $_SESSION["User"] variable to select the allowed menu items.

is_allowed_menu_item()

Checks the path to a requested file and returns true, if this page is accessible for the session user, else false. Checks all: role, workflows, subscriptions.

is_allowed_role_change()

Checks whether a user is allowed to login with this different role. This is the case, if that role is included in the role list for the users role within the role hierarchy. That function is usually only used tfor testing purposes and called when logging in with the "as" parameter. See the login page description.

PDF, PDF_adapted

Both classes are simple wrappers for the public TCPDF framework which is used to create PDF documents. PDF_adapted just extends formally TCPDF to be able to apply a page footer which is set in the configuration.

PDF provides the function to create a PDF from an html template. The template shall be put into the templates folder. It can contain fields which are resolved to their respective values using the data record indexed. Additionally fields cn be computed in the calling function for replacement within the template. An example for a template is:

```
<h4> {#Veranstaltungen.Bezeichnung#}, {#Ort#}, {#Zeit#},
{#gebuchteSitze#} Personen</h4>
<span>{#Buchungsliste#}</span>
<p>Storniert und deswegen für diese Veranstaltung <b>nicht mehr
gültige Codes: {#Storni#}</b></p>
```

Functions

convert_to_pdf()

Create a pdf document from the provided html String. In order to set a footer text, set the $this->footer_text variable first. Similar with the document author: to set one, set the $this->document_author field.

create_pdf()

Create a pdf based on the table data.

Pivot_table

A little helper class for the Tfyh_list class to creat a pivot table based on a provided list. There is just one function available: Pivot_table::get_html() to return an html formatted pivot table of the passed list.

Socket

The class to handle all data exchange with the data base. Build for a mySQL data base. The socket has no application logic except the change log. In the change log table all data modifications are logged. The tfyh_cronjobs care for the log cleansing.

In tfyh native applications all tables have a primary key named ID and being a unique, autoincremented integer value. Efacloud has a different key management, For that purpose a key can also be given as a record with multiple fields and values which must

all match.

Generic functions

The three generic functions are

- open_socket(),
- close() and
- query().

The query function takes an arbitrary SQL command and shall be avoided for safety reasons. It is not logged. But all queries which are performed are listed in a log file "queries.txt"

Standard modifications

The three standard modifications are insert, update and delete.

insert_into()

Insert a data record into the table with the given name. Does not check any key or value, but lets the data base decide on what can be inserted and what not. Returns either the "ID" of the inserted record on success or a String with warnings and error messages.

update_record(), update_record_matched()

Update comes as a set of two different functions, but update_record is a convenience short hand for the other. It gets the first record of which the key is matching and updates all provided values, including the empty ones. It returns an error statement in case of failure, else an empty String.

delete_record(), delete_record_matched()

Very similar to update except that the first matching data record is deleted.

Find records

To retrieve multiple records from the data base find_records comes as a set of functions, all being somehow a shorthand for find_records_sorted_matched().

find_records_sorted_matched()

Find all records as indexed array of records, each as associative array of key => value matching the provided key array and condition. Sort them in the requested order. Returns false, if the value is not found or any other error occurred.

The condition combines $key and $value. Use it to the SQL type operand, e. g. "!=" for not equal. Set to "" to get every record. You can use a condition for each matching field, if so wished, by listing them comma separated, e.g. >,= for two fields if which the first shall be greater, the second equal to the respective values. If more matching values are provided than conditions (e. g. 3 values, but only two operators), the last condition is taken for all extra matching fields.

The result can be sorted for multiple fields by providing a comma separated list of fields. Precede the field name by a '#' to sort also text as numbers, e.g. "#EntryId". The sort order is ascending or descending, but always the same for all sort fields.

find_records(), find_records_matched(), find_records_sorted()

These are all short hand convenience calls for finding multiple records.

Find record

To retrieve a single record from the data base find_record comes as a set of functions, all being somehow a shorthand for find_records_sorted_matched() with a maximum number of records being 1.

This set, the record returned is always the first found. No check is done whether there are further records matching that condition. It is in the responsibility of the calling function to ensure unambiguity of the answer.

find_record_matched(), find_record(), get_record_matched()

All short hand convenience calls to the find_record_matched. The get_record assumes that the provided value matches the “ID” data field.

Full table export and import

Use get_table_as_csv() or get_table_as_array() to retrieve a full table, all columns, all rows.

Import a csv file into a table or delete table records (provide single column csv with IDs only). The csv-file must use the ';' separator and '"' text delimiters. It must contain a headline with column names that are literally identical to the mySQL internal column names. All data records must be of the same length as the header line, not more, not less. If a data record does not comply, it will not be imported.

The first column must be the records 'ID'. If this is not the case, no data will be imported at all. For data records with an existing 'ID' all provided record fields will be replaced, i. e. data will be deleted, if the respective field is empty. For data records with an empty 'ID' the 'ID' will be auto generated by the mySQL data base. In this case, and if the provided 'ID' is not yet existing, a new table record is inserted into the table. All changes will be logged, as if they had been made manually.

A full table import needs a singel key field to work. The name of this ‘ID’ field can optionally be provided, default is ‘ID’.

Get data base structure information

You can retrieve information the following on the table structure with a set of functions wrapping the necessary SQL calls:

- get_db_name()
- get_column_names()
- get_column_types()
- get_indexes()

- get_table_names()

Modify the data base

You can also manipulate the data base structure.

- create_table(): that drops the table first, if existing
- add_columns()
- set_unique()
- set_autoincrement()

Tfyh_audit

A non-static class container file for an audit routine which is at least once called by the daily jobs routine. It may be triggered by other activities e.g. software update. The tasks are:

- Check and correct web server directory access settings using the settings_tfyh directives
- Check users and access rights.
- Check backup files count and size.

The result is logged and written to the audit log which can be useful for debugging and supoport.

Tfyh_cronjobs

A static class container file for a daily jobs routine. It may be triggered by whatever, checks whether it was already run this day and if not, and if not run starts the sequence. The tasks are:

- run a backup
- Cleanse the change log and activity logs
- Collect the user access statistics

You shall extend this class to a Cronjobs class and run additional daily jobs such as data deletion in the extension.

Tfyh_list

This class provides a list segment for a web file. The segment displays a part of a data base table as it is within the data base. No manipulation, just filtering and sorting applies. It provides a link to download the list as zipped csv-file.

The list definition must be a CSV-file, all entries without line breaks, with the first line being always ‘id;permission;name;select;from;where;options’ and the following lines

the respective values. The name is the column name and can be preceded by a '#' character to enforce sorting as unsigned integer (e. g. #EntryId). The values in select, from, where and options are combined to create the needed SQL-statement to retrieve the list elements from the data base.

Options are:

- sort=[-]column[.[-]column]: order by the respective column in ascending or descending (-) order
- filter=column.value: filter the column for the given value, always using the LIKE operator with '*' before and after the value
- link=[column name]:[URL] link the column to the given url e.g. 'link=ID:../forms/change_user.php?id='. The field value will be appended to the URL in the link provided.
- For the column which reflects the user tables user ID always the "link=[field name of user ID]:nutzer_profil.php?nr=" is used as default without being explicitly specified.

Lookup fields in list definitions

Instead of column names list definitions may contain lookup fields. That is an Id which references to another table and there to a column, using an inner join.

An example is "BoatId>Boats.Name@Id" in a list for a boat trip table "Trips". That gives the name of the boat instead of its Id Using an inner join of type "INNER JOIN `Boats` ON `Boats`.`Id`=`Trips`.`BoatId`".

Variable list definition

The list definition may use variables, which is in particular useful for filter definitions. They will be replaced by values within the definition during Tfyh_list construction.

So you may e.g. choose to define a list with "where" being "(NAME LIKE {name})" and pass the argument array ["{name}" => "John%"] to the constructor. The replacement value (e.g. "John%") MUST NOT contain a ';' for security reasons. If so, the replacement will be "{invalid parameter with semicolon}" instead of the given value.

Calculated additional columns (compounds)

List definitions may contain additional "compounds" columns, i. e. Strings which are compiled from table entries. They are a String with placeholders $1, $2 asf. for the entries as they are in the definition, so a definition could look like:

```
1;member;Persons with birthday; \
  ID,Person=$2 $3 (born: $5),firstname,lastname,gender,birthday;persons; \
  1;sort=lastname.firstname
```

Note: placeholders start with $1 for the first column in the definition and skip the compound columns in the count.

The list is always displayed as a table grid. It will show the default sorting, if no sorting option is provided.

Construction

Upon construction of a list always the complete list definition file is read with all the

lists defined. You can choose the one to use by its name or ID, or choose ID=0 to get the list of lists displayed rather than a content of a specific list.

The list constructor allows to provide arguments as array which will be used as variables in the list definition. E. g. using s definition with the filter "name={name}" and providing as argument ["name" => "John"] will return a list with only those records with the name "John".

parse_options()

Parsing the options is usually a function of the constructor, parsing the options of the selected list. But when it comes to the list set display, these need also to be parsed to be transported in the list call.

Simple getters

There is a set of simple list parameter getters implemented: is_valid(), get_table_name(), get_list_name(), get_list_id(), get_set_permission(), get_permission(), get_all_list_definitions().

Get the list

There are different options to get the list, depending on the use case.

Tfyh_logger

The logger class provides functions to log what happens during execution. Three log types exist:

- Actions: They will be collected in three files, the "dones.txt", "warns.txt" and "fails.txt". They are not aggregated for statistical purposes, but meant to be used for tracing down errors. It can be provided as a list.
- Activities: A single activities.txt file with arbitrary type activities which will be counted per day for statistical purposes. Statistics can be provided as a list.
- Mass transactions: While the format is like with Activities, the purpose is transparency on who did what mass transaction, not statistical aggregation. It can be provided as a list.

Methods for actions

log()

log actions, warnings and failures

list_and_cleanse_entries()

Return all logged actions which are younger than $maxAgeSeconds as list "\n" separated. Remove those which are older than a certain age if requested.

Methods for activities

log_activity()

simple file append of the provided message plus time stamp.

collect_and_cleanse_activities()

Reads the activities log and creates an array with the count of activities per type, except the current day. It deletes the collected activities from the activities log and appends the count per type with the date of yesterday to the daily count log.

get_activities_html()

Return the activities per day for the last $count_of_days as html table.

Methods for mass transactions

log_mass_transaction()

log actions, warnings and failures

list_and_cleanse_mass_transactions()

Return all logged mass transactions which are younger than $maxAgeSeconds as list "\n" separated. Remove those which are older than a certain age if requested.

Tfyh_user

This class provides all generic function on users. It shall be extended by a Users class to provide application specific functions such as get_user_profile.

It also handles the rules of who is allowed what. So it resolves the role hierarchy and the menu, subscriptions and workflow allowances. The access rules are provided in three config files:

1. Role hierarchy.
   Roles are defined, typically 4-6, where the hierarchy tells which role includes what other roles. There must always be one anonymous role for users who are not logged in. A preceding asterisk in the role definition indicates the role is a privileged one. For privileged roles the names who have that access right are disclosed, for the others only the count af assignments
2. Workflows (optional).
   There is one integer bitmask of workflows. So up to 32 different workflows can be defined freely. To refer to a workflow use the @-character plus the bit mask value, e.g. @64.
3. Subscriptions (optional).
   There is one integer bitmask of subscriptions. So up to 32 different subscriptions can be defined freely. To refer to a subscription use the #-character plus the bit mask value, e.g. #128.

It provides functions monitor allowances and resolve attributes, see below.

It provides an option to get user related attributes of three separate lists in which a user can be assigned as many list attributes as is wanted. An example is the list of functions. Any function may be assigned to any user, a record is one of these assignments.

Access control

is_hidden_item(), is_allowed_item()

check whether a menu item is hidden (simple check for the preceding ".") or whether it is allowed for the current user (by all: role, workflow, subscription).

get_all_accesses()

return a HTML encoded list of role, workflow and subscription users. For provoleged roles all names, for the remainder the count of users with this access right.

get_user_services()

a HTML list of the services (I. e. workflows or subscriptions) a user is granted or subscribed to.

Other functions

get_user_attributes()

return all attributes of a given type, currently: Funktionen, Ehrungen, Spinde. This is very specific for brg-intern, so if needed at different places it will need some further abstraction.

check_new_user_name_for_duplicates()

Check whether a user name (first and last name) is or was already used in the user archive and the user table.

get_action_links()

get the html-links which are available for a current user out of the set of action links configured based on the access rights of the current user.

Token_Handler

A utility class to create one-time tokens for user identification. It creates more or less random tokens, stores them in a token file together with a time stamp and can cleanse the lot. It is used for the api identification, but should no more be used nowadays.

Toolbox

The last in the list is actually the most used. There is no file of the application which does not instantiate the toolbox and with it its dependent classes Users and Config.

The toolbox reads all application configuration and user settings, and provides functions for session support, data validity control, file handling and zipping, csv parsing and generation, load throttling and some miscellaneous stuff.

functions for session support

start_session(), generate_token(), display_error()

Start a session or display an error, if failing. Generate a random character sequence for arbitrary purposes.

create_login_token(), decode_login_token()

Create and decode a login token which can be used as login without password for a limited period of time. Useful for e. g. Feedback gathering.

Data validity checks and formatting

check_and_format_date()

Dates may be ISO type YYYY-MM-DD (e. g. 2021-03-30) or DD.MM.YYYY (30.03.2021). It is checked whether the String is a valid date and returned in the ISO way.

form_errors_to_html()

provide a nice red character color and a preceding “Fehler: “ to a String

strip_mail_prefix()

remove a mail prefix used for duplicated mails as accounts such as “2.max.mustermann@tfyh.org” => “max.mustermann@tfyh.org”.

create_GUIDv4()

return a version 4 GUID (36 characters).

age_in_years()

return the current age in years based on the birthdate.

CheckIBAN()

check an IBAN validity.

check_password()

check a password against the minimum security rules. 8..32 characters, at least three character types.

swap_lchars()

password obfuscation.

File handling

list_files_of_branch()

Parse a file system branch and return all relative path names of files.

unzip(), zip_files(), zip()

Archiving support. The last (zip()) zips a String into an archive.

return_file_to_user(), return_string_as_zip(), return_files_as_zip()

Return information to the user (as download). This is not providing a link, but rather providing information based on the current user privileges which are not open to public. The information is a file, a zipped single file or a zip-archive containing multiple files. With the last function a zip file will be created in the file branch which the user visits.
All these functions do not return, but exit the script. The files returned stay in the directory where they were fetched from. If zip-archives are created, they get a 0600 file access mask not to be accessible by the public.

get_dir_contents()

Return the contents of a directory as html table.

CSV support

read_csv_line()

Read a single csv line assuming a separator character ‘;’ and a text delimiter ‘”’.

encode_entry_csv()

Encode a single value to a csv entry assuming a separator character ‘;’ and a text delimiter ‘”’. Build a line by encoding the entries and appending them one by one adding a ‘;’ in between them.

read_csv_array()

Read a simplified csv-file into an array of rows, each row becoming a named array with the names being the first line entries. CSV-format must be with text delimiter = " and separator = ;. There must not be any space character left or right of the delimiter. First line entries must not contain line breaks. Lines ending with unquoted " \" will be joined with the following line. The file is preferrably encoded in UTF-8, but ISO-8859-1 should also work due to automatic encoding detection.

Load throttling

load_throttle()

Measure the frequency of web page inits, api sessions and errors. Meant to prevent from machine attacks. A set 3000 of init timestamps resides in the /log/inits or /log/api_txs folder. When this function is called, the current pointer is read, and the timestamp to which it's pointing is also read. If such timestamp is existing, it is checked, whether it is older than $event_monitor_period. If so, it is replaced by the current timestamp, the pointer is stored back to the file system and true returned. If not, an error page is displayed. Same procedure with errors: then the load throttle records the error and blocks the site in case more than 100 errors have been received in the monitor period.

Miscellaneous

Return a timestamp for a booking, based on separate date and time; get the encoded parameters of a request as plain associative array; "Encrypt" a base64 String by xoring it with a key. Decryption is the same as encryption.

Tfyh configuration and resources

Before we continue with the forms it shall be advised how to configure an application using this framework. Here are three layers of configuration:

The framework settings

They are defined in the /config/settings_tfyh file.

Their purpose is to configure the application. These settings are part of the code like form layouts and shall not be editable by the user. They are read on startup af a PHP request and accessible as public variable $toolbox→config→tfyh-settings. They are stored as a two level array, e.g. $this→settings_tfyh["config"]["app_name"].

Framework setting must never be changed by the application.

The tenant settings

They are defined in the /config/settings_app and /config/settings_db files.

Their purpose is to configure an app according to a tenants' properties such as footer texts, email addresses and so forth. The settings_db is separate and does only contain the access settings for the data base. It can only be edited at the applicatioon installation, whereas the other tenant settings are configurable via the application configuration menu.

Part of the tenant settings are also the colors, which are stored in the resources area, as app_colors.txt file. When changing the colors, the application uses this file and the app_no_colors style sheet to generate the tenant specific style sheet for the colouring.

Tenant setting shall only be changed by the respective forms, never else.

The application run time configuration

This is a configuration table in the data base and is meant for permanent storage of application parameters which may change due to program execution such as the highest membership index or similar. It is for historical reasons also partly used for tenant specific and even application specific configuration, but should not be used for it in the future.

Application run time configuration can change at any time. If a parmater is used, it must be read on the fly.

Other configuration

All configuration sits within the /config/ folder. There are the following directories:

- access: contains the public and internal menu definition and the role_hierarchy file, the workflow and subscriptions bitmask definition and, if applicable a menu (i.e. access control) definition for any server API.
- layouts: contains all form layouts
- lists: contain all list set definitions
- snippets: contain the definition of the html snippets displayed at the start of a page, between menu and body and as footer. Configure the snippets to change the page title, the logo displayed and the footer text, if requested.

This directory also contains the settings_app, settings_db, and settings_tfyh files.

Styles and resources

Two style sheets define all styles: a generic w3_style.css, in major parts copied simplified and adapted from w3schools.com, and the app-style.css which contains those parts of the style sheet which use colors and font definitions. The difference is made to distinguish the generic from the tenannt specific definition.

The style sheet provides all styles needed for the standard left hand menu and responsive form design. Note that the configuration snippets, the start and end snippets of the menu class and all form layouts use these styles. So changing the style sheets to change the layout concept will incur also a code change in these parts. It is not recommended.

They sit in the /resources/ folder. Please put here al other style elements. Like logo and images for the dateiablage.php file manager.

Tfyh user and session management, forms

A key function for all applications to benefit from this framework must be data entering. So forms play a major role. The framework shall help to concentrate on the business logic which must be written in plain PHP, supported by some framework helpers.

It starts with the user identification and authorisation (see the Menu class) and continues with a multistep form support.

Tfyh user and session

Tfyh uses two $_SESSION array sections to manage users and input data:

- $_SESSION[“User”]: the record of the current user. May or may not be known in the data base.
- $_SESSION[“forms”]: per form sequence this contains a subarray $_SESSION[“forms”][$fseq] holding all parameters ever posted by this sequence of multistep form input.
- $_SESSION[“getps”]: per form sequence this contains a subarray $_SESSION[“getps”][$fseq] holding all parameters ever conveyed via the GET-parameters in the URL by this sequence of multistep activity.

The arrays are gathered within the init and Form classes, so that you will not really have to care about the details. Your form layout will rule the fields to be entered.

The settings_tfyh file

This file provides an option to configure the fraework for the specific application. It is plain text, and has section of settings. The syntax is simple: one line = one setting, empty lines and lines starting with # are ignored, lines ending with ‘ \’ are joined with the following line. The setting has two names separated by a dot reflecting a two level settings array.

The values are literals: String must be quoted (“) and inner quotes and backslashes escaped (\”, \\), arrays must be in square paranthesis ([]).

Here’s an example:

```
# all settings for usage of the common tfyh modules
# ther must not be extra blanks or trailing characters.

# module users.php
users.action_links=["Verwalter: <a href='..\/pages\/nutzer_profil.php?
id={#ID}'>anzeigen<\/a>","Verwalter: <a
href='..\/forms\/nutzer_aendern.php?id={#ID}'> \u00e4ndern<\/a>,"]
users.user_table_name="Mitgliederliste"
users.user_id_field_name="Mitgliedsnummer"
```

```
users.user_archive_table_name=""
users.user_firstname_field_name="Vorname"
users.user_lastname_field_name="Nachname"
users.user_subscriptions=false
users.user_workflows=false

# module config.php
config.app_name="Get your data in - app"
config.changelog_name="Changelog"
config.parameter_table_name="Parameter"
config.pdf_footer_text=""
config.pdf_document_author="tfyh.orge"
#left, top, right, header, footer
config.pdf_margins=[20,15,15,10,10]

# module init.php
init.max_inits_per_hour=3000
init.max_errors_per_hour=100
init.max_concurrent_sessions=25
init.max_session_duration=600

# module api, client handler.php
api.hideout_key=1234567890
api.token_key=1234567891

# module socket.php, record version history
history.Mitgliederliste="Historie"
maxversions.Mitgliederliste=25
history.Projekte="Historie"
maxversions.Projekte=25
```

Tfyh – framework forms

For some basic activities which are needed in all applications, a set of forms is provided covering login & password reset, tenant configuration, table import, mail sending and reading, as well as file upload.

All forms follow the same logic and structure:

1. They are build using one or more layout files, as many as the form has steps to provide
2. The form action always calls to the very same form, indicating the step that was done
3. A specific $_Session field gathers everything which was ever posted into this form sequence
4. A business logic which decides on the provided input what to do next
5. A form display part

A typical form file

Here is how a typical form file would look like:

1. Start with an init.php call to do the authorisation

2. Continue with the interpretation of GET Parameters. Typically you use get parameters to select the record to be modified on modification forms or similar. Since it is gathered in a form sequence specific array this will also keep the context if the user has multiple browser tabs opened in parallel.
3. If this is a subsequent call by the form being filled and sent, rebuild the form in memory and interpret all post parameters using a call to the appropriate Form class.
4. Do all the business logic, e. g. store data, decide on the next step, send mails etc.
5. Display the page based on what was decided to be the next step of the form in 3. (If it is the first call in a sequence then step 3 was skipped. No options to select.) The Form calls builds the form based on the layout. You may use stored data base values to preset the forms inputs. If this is the last step it may not display any form but just a completion message.

Go to the login.php to have a look. And see in the Form calls for the layout definition.

configparameter_aendern.php

Use this form to change the tenant config. The form layout defines what parameters will be used for change. The result will be stored in the config/settings_app file as base64 encoded ext.

dateiablage.php

Use this form to upload files into the upload directory and manage all files there. It provides all functionality to upload, create directories, download and delete file within the uploads section. A sort of min file manager.

farben_aendern.php

Use this form to adapt the theme colours. Note that this will always overwrite the current app-style.css using the provided colours and the app-style-no_colors.css as template. The color set itself is stored in the app-colors.txt file which also will be overwritten. A change can not be undone.

login.php and reset_password.php

The user login form and the password reset form. User login uses either an E-Mail address of the user or its userID. The name of the table carrying the information of users is configured in the users' section of the settings_tfyh.

If the password is left empty, the provided E-Mail address corresponds to a registered user and this user has no password set, a 6 digit token is sent to this mail address to provide access.

If the user has set a password and has forgotten it, the reset_password.php provides a form which will delete the password sending a deletion token. Then the user can login again with another token and set a new password.

By that means these forms provide a complete although very basic access control management.

mail_versenden.php and mail_nachlesen.php

These two forms allow for users to send mails to predefined distribution lists.

The distribution lists are set via a set of tfyh_list lists, so you can use subscriptions or similar criteria to select users.

You may parametrize a list in "mail_versenden.php" by calling it as "mail_versenden.php?listparameter=whatever" and using the String "{listparameter}" as parameter in the list definition, see Tfyh_list section for details on list parametrization.

All mails sent are stored in the data base in a Mails table. They can be read by those who can send using the mail_nachlesen.php – differentiated by the distribution list.

tabelle_importieren.php

Use this form to import a csv formatted table into the data base. Use the form layout to define which tables may be imported.

The csv table must be ';' separated with '"' text delimiters. The first line is the header line, column names must match exactly the data base column names (case sensitive). Not all columns must be provided. There is one primary key per table. This key column must be included in the columns list.

Records are inserted, if the value for the primary key is empty. Records are updated, if the primary key is provided together with at least one other column. Records are deleted, if the primary key is provided and no other field.

The import will first dry run and show what will be done, then the user has to confirm the action.

Tfyh – framework pages

Few generic pages are part of the framework to complete the functionality of access control and logging.

alle_berechtigungen.php

Use this page to display a compilation on all access rights assigned within the applications. Privileged role owners are displayed one by one, providing the names. For all other roles, workflows and subscriptions the number of users with the access right assigned is given.

show_actions.php, show_changes.php

Display the log-contents for either logged activities (together with a statistic for the last 14 days) or data changes.

show_lists.php

Display a selected list. The user can sort and filter the list and download its contents as csv formatted table (zipped).

logout.php, construction.php, error.php

Standard pages after the user logged out, hit a menuy link which is still to be built or caused any transactional or session error (most common: session expired).

To display an error use the $toolbox→display_error function which redirects then to the error.php page.

maintenance.php

This is a mini pages which can be displayed upon maintenance. Just edit the init.php to provide an information on the time the site is planned to be up and running again to show this page instead of any other web page of the application. It is the only common page located in the /public/ folder.

Acknowledgements

The set was created using the eclipse based Zend IDE.

This document was created using oracles openOffice word processor.

Many of the design hints I borrowed from w3schools.com.

I would never have come that far in PHP knowledge without the Google search engine, stackoverflows explanations and the php.net tutorials.