Classes design v2-0

work in progress

Current classes

Not all files in classes/ dir, but just the relevant classes.

AuthSaml

Methods that check if user is authenticated with SAML, is an admin (as defined in SAML config), returns authenticated user data.

This stuff should be moved to the new user class.

AuthVoucher

Just a couple of methods that check if a voucher stored by id is actually a valid voucher, returns an array of voucher properties.

These methods should be moved to a voucher class.

DB_Input_Checks

Methods that validate data used in queries (that a string representing an e-mail is in valid e-mail format, etc).

Should be contained in an IOvalidation class ?

DB

Prepares and executes queries, loads db-relevant config data, initiates DB connection.

Two exception handlers (both extend **Exception**, but do not add anything) - **DbException** and **DbConnectExc**... - but handles **PDOexceptions**

Functions

-- **getTrackingCode** (some transaction class should have this instead?)

-- **getVouchers** simply returns an array of vouchers that are valid. Should there be a vouchers class that holds all objects of type Voucher?

-- checks if user is SAML authenticated. This belongs in a user class?

-- some mail processing methods

-- gets some files properties from DB table files. A class that holds an array File objects can do that

-- gets vouchers that were created by a user. Having a Vouchers object in a User class and implementing this method in class User can be more intuitive.

-- gets files by voucher id. Can also be moved to a transaction class, voucher class, file class?

-- gets transaction details by querying the DB

... (a lot more functions that can be sorted by what objects they perform stuff on and hence be moved into classes that init those objects)

In general, there's a lot of functions that do queries on a "need-to-know" basis and kinda omitting the whole object-oriented model. Instead the base classes can fetch data they need on initialization (interact with the DB) and the UI scripts can call methods of these objects to get what they have access t.

That way the UI scripts won't see the "raw" user data in the DB, but instead in a more secure way (as described in the Pine Security audit).

Mail

Prepares and sends e-mails.

-- Functions.php contain some mail related functions (like adding recipients to mail). If a Mail object is initialized when "send files" form is loaded and not on the "send" click event (does it actually work that way?), these could be moved to Mail.php

TeraSender, Zipper

Support classes that are not part of core functionality and should not be tinkered with unless one knows how they work.

The classes design for version 2.0

Goal with the design

- Upper-layer classes (available to the interface scripts): Functions (generic functions that do not interact with DB), IOvalidation, Mail, Zipper, Terasender, Config ...
- Mid-level (base classes that interact with data): File, Files (to hold an array of File objects), User, Transaction, Transactions, Voucher, Vouchers, ...
- Low-level/DB interaction: DB class that uses PDO (which is the current DB.php)

An important thing is that we start droping hand-made singleton implementations (**getInstance** methods) in favor of mostly static classes, this reduces memory footprint and makes related code about 15% faster (benchmarked).

Classes in new design

Most class names have been choosen to represent best what is their purpose and if possible to avoid conflict with existing name so that code refactoring can be done while keeping old code bits and still be working.

Core exceptions

File : classes/Exceptions.class.php

Defines core exceptions :

- **LoggingException** which takes a message code to display error to the user and additionnal details to log, linking the 2 with an error code that can be displayed to the user, used in an error report form ...
- **DetailedException** (extends **LoggingException**) which logs the stack trace and given additionnal details (any other mixed args after the message code)

All exceptions, anywhere in the code should extend one of those, preferably **DetailedException**.

Config

File : classes/Config.class.php

Takes care of config file loading, configuration parameter read.

Only exposed methods is **get**, to get the value of a given parameter (or a set of parameters that have the same prefix by using wildcard *).

At first it should still have the **getInstance** method and implement **arrayaccess** to suport old config access style and should log when old access is done in debug mode with calling script and line so that we can track down remnants of old code.

Configuration defaults can be set in **includes/ConfigDefaults.php** as a **$default** array indexed by

parameter name.

Parameters post-processors can be set in **includes/ConfigProcessors.php** as a **$processor** array indexed by parameter name and pointing processor name (processors are methods from **Config** class named like **processorProcessornamewithfirstletteruppercased**).

Usage :

```
$foo = Config::get('foo');

$db_parameters = Config::get('db_*'); // Array of parameters whose keys begin with 'db_', returned keys are stripped from the prefix
```

Along with the **Config** class those exceptions (all extending **DetailedException**) are defined :

- **ConfigFileMissingException**
- **ConfigBadParameterException**
- **ConfigCannotSetException** temporarly used until old access style code is put down
- **ConfigUnknownProcessorException** used to report that given post-processor does not exist

DBI

File : classes/DBI.class.php

Database interface handler.

Why renaming this one : the idea behind classes name is that they indicates what they represent and allows to manipulate, since this class represent an interface to the database and not the database itself this is more like a **DatabaseInterface** class than a **Database** or **DB** class, but this is a long name, so **DBI**.

It exposes the same methods as a **PDO** instance (through use of magic **__callStatic**), so one can use it as :

```
$statement = DBI::prepare('SELECT * FROM foo where bar = :bar');

$statement->execute(array(':bar' => $bar));
```

Along with the **DBI** class those exceptions (all extending **DetailedException** or a child of it) are defined :

- **DBIConnexionException**
- **DBIConnexionMissingParameterException**
- **DBIUsageException**

DBObject

File : classes/DBObject.class.php

Base database stored object class, every stored object class must extend that one.

It provides easing methods to cache objects (to avoid loading them several times, thus sparing memory and cpu), converters from and to database compliant data, methods to insert or update database records ...

It also provides a way to define database fields to be used in install/update scripts.

Exposed methods (inherited by children) are :

- **fromId** to get object in cache from its id and load it if not in cache
- **fromData** to get object in cache from its id and fill it from already loaded data it if not in cache
- **fillFromDBData** to fill object properties from database returned data, handling typecasting on the fly
- **toDBData** to get a database compliant data set from the object, handling typecasting on the fly
- **getDBTable** to get the object's table name, handling prefix if needed **(object classes must use this instead of hardcoded table name)**
- **updateRecord** to update an existing record (selected from its id) from database compliant data
- **insertRecord** to insert a new record from database compliant data

Along with the **DBObject** class those exceptions (all extending **DetailedException** or a child of it) are defined :

- **PropertyAccessException** used in child classes' magic getters/setters to tells that accessed property does not exist

Transaction

File : classes/Transaction.class.php

Represent the act of sending one or more file to one or more recipient, extends **DBObject**.

It defines its stored fields in the $dataMap static property.

It exposes setters and getters to access main transaction properties.

It exposes various methods to interract with a single transaction :

- **addRecipient**
- **removeRecipient**

- **addFile**
- **removeFile**
- ...

It also provides methods to get transactions sets :

- **fromId** (inherited) to get a transaction from its primary key (cache)
- **fromData** (inherited) to fill a transaction object from an already loaded table row (cache)
- **fromVoucher** to get a transaction from an upload voucher (cache)
- **create** to create a new transaction at upload start
- **getExpired** to list expired transactions for removal

Along with the **Transaction** class those exceptions (all extending **DetailedException** or a child of it) are defined :

- **TransactionNotFoundException**

Recipient

File : classes/Recipient.class.php

Represent a recipient (from transaction or maybe invitation), provides methods to create one from transaction/invitation and email and to delete it.

File

File : classes/File.class.php

Represent an uploaded file, provides transaction accessor (magic getter), creator from transaction, filename and size, methods to add and read chunk from offset ...

Invitation

File : classes/Invitation.class.php

Represent an invitation, provides send method which uses Mail class to send invitations, may use **Recipients** class (?).

User

File : classes/User.class.php

Provides access to user preferences (lang, upload and invitation options).

May hold current user, offering a way for **Auth*** classes to set it along with several properties (IdP, name ...).

AuditLog

File : classes/AuditLog.class.php

Represent a audit log entry, provides ways to select them from user id and maybe type for display (admin, transaction files downloads ...).

StatsLog

File : classes/StatsLog.class.php

Represent a stats log entry, provides ways to select them.

IOValidation

File : classes/IOValidation.class.php

Holds various data validators (which throws dedicated exceptions) as static methods.

Avaliable validators are :

Along with the **IOValidation** class those exceptions (all extending **DetailedException** or a child of it) are defined :

Utilities

File : classes/Utilities.class.php

Holds various utilities as static methods :

- **generateUID** to generate upload voucher, download voucher, invitation voucher
- **formatDate** to format php timestamps as per configuration
- **sizeToBytes** to turn ini file size notation into number of bytes

Along with the **Utilities** class those exceptions (all extending **DetailedException** or a child of it) are defined :

- **BadSizeFormatException**

Language

Some class that loads the right language file and translates language tokens (refactoring LanguageSelection should do it)