

RECOMANDATIONS TECHNIQUES

PLATE-FORME AFIDE

Bases de données, scripts PHP
et documentation



Sommaire

1 Utilisation de ces recommandations

1.1 But du document

1.2 Mises à jour

1.3 Justifications

2 La base de donnée

2.1 Les contraintes

2.2 Les Fonctions PL/PGSQL

2.2.1 Les méthodes d'implémentation

2.2.2 Le processus de production

3 Les scripts php

3.1 Regroupement des fichiers communs au back-office et au front-office

3.2 Exemple de structure de regroupement

3.3 Généralisation des entetes

4 La documentation

4.1 Fichiers pris en compte

4.2 Formats des commentaires

1.1. But du document

Ce document a pour but de décrire, de justifier et de formaliser les évolutions techniques de la plate-forme AFIDE.

En effet, l'hétérogénéité du codage et des méthodes utilisées commence à être ressentie comme un obstacle aux nouveaux développements, et ne permet pas une utilisation optimale des capacités de production, tant au niveau technique qu'au niveau humain.

Cependant, ce document n'est pas à voir comme un 'règlement' ou des 'lois' de programmation. Il a pour vocation de récapituler et de formaliser des choix faits par l'ensemble de l'équipe de production, et n'a de sens que s'il évolue et est modifiée dès que le besoin s'en fait ressentir. Si cette documentation reste statique ou si elle n'est pas approuvée par l'équipe de production, elle perd tout son sens et n'a plus de raison d'être.

1.2.Mises à jour

[illegible]

1.3. Justifications

Les points suivants ont donc été relevés par l'équipe de production:

- L'absence de contraintes sur la base de donnée ne permet pas de stocker les données dans la base de façon intègre et cohérente.
- le serveur de base de donnée Postgres est peu utilisé, contrairement au serveur de PHP: une meilleure repartition de la charge entre les 2 serveurs est donc souhaitée.
- Les scripts php deviennent de moins en moins clairs et compréhensibles au fur et à mesure des développements et leurs tailles excèdent largement les recommandations de la communauté PHP.
- Aucun système ne permet actuellement de savoir de façon satisfaisante où sont utilisées les fonctions ou encore de déterminer avec exactitudes les relations entre les différents fichiers de scripts.
- L'absence de directives de développement et de documentation ne permet pas d'exploiter correctement le code précédemment écrit.
- L'hétérogénéité des méthodes de stockages des données dans la base implique des méthodes de lectures des données elles aussi hétérogènes. Cela entraîne notamment des problèmes lors de l'exportation et l'importation des données de la plate-forme.

2. La base de données

Les bases de données utilisées actuellement par les plateformes sont situées sur des serveurs postgres de version 7.1.3 et les tables n'ont aucunes contraintes.

Les bases sont interrogées avec des requetes SQL écrites dans les scripts PHP, c'est à dire coté client.

Il n'y a aucune convention sur le format des données lors de leur enregistrement et leur lecture.

Comme cela aurait du etre fait depuis longtemps, et suite à de précédentes tentatives infructueuses, des contraintes d'intégrités vont etre misent en place. Elles permettons par la suite d'avoir des données cohérentes et devraient donc éviter de nombreux bugs.

Afin de repondre au besoin de repartition de charge, de clarification et de structuration du code, il a été retenu de mettre en place des fonctions PL/PGSQL, c'est à dire d'interroger la base de données non plus coté client mais coté serveur.

Cependant, la version actuelle du serveur ne permet pas d'exploiter pleinement le PL/PGSQL, en limitant par exemple à un resultat unique le retour d'une fonction.

La plupart des requetes de la plate-forme retournant plusieurs resultats l'interret du PL/PGSQL sur la version actuelle du serveur devient donc très limité, voir nul.

Seule une version 7.4 (ou supérieure) de PL/PGSQL permet de passer outre ces limitations.

La migration de l'ensemble des serveurs actuels vers la version 7.4 ne pouvant se faire dans l'immédiat suite à des contraintes techniques, il a été retenu de mettre en place un tel serveur sur une machine disctincte, et d'adapter les developements en vu de la migration, prevue au mars 2005.

Compte tenu de la configuration des bases de données (format de stockage des données) et en vue d'une uniformisation des methodes de stockage il a été retenu :

- De ne pas effectuer de traitements sur les données à leur

enregistrement,

- D'effectuer ces traitements, s'il sont nécessaires, à la lecture des enregistrements.
- D'utiliser le format int4 pour stocker les entrées, afin d'éviter toute erreur de conversion.
- D'utiliser le format float4 pour les chiffres à virgules pour les mêmes raisons que précédemment.
- D'utiliser le format varchar pour stocker les données de type texte.

2.1.Les contraintes

Compte tenu des travaux déjà effectués sur ce sujet, il à été entendu de les finaliser et de les mettre en place le plus rapidement possible.

La base de donnée évoluant avec les développements et étant constamment sollicité, il est difficile, voir impossible, de mettre en place les contraintes sans bloquer la production et provoquer des erreurs sur l'ensemble de la plate-forme.

La mise en place de la plate-forme demo-client1, de part sa dissociation du serveur de bases de données, permettrait cela, mais viendrait perturber les développements pour lesquels elle a vocation.

Ainsi, les contraintes seront donc misent en place non pas sur les tables de démo-client1, mais sur des tables doublons.

Les requetes se feront toujours sur les tables originales, et entrainerons les memes actions sur les tables doublons, grace à la mise en place de triggers.

La majorité des erreurs liées aux contraintes devraient donc etre relevées et ce devrait éviter de nombreux bugs lors de la mise en place « réelle » des contraintes.

Les tables doublons porteront les memes noms que les originales, mais seront terminées par « _CT » pour les distinguer.

Une fois cette periode de tests terminée, le remplacement des tables originales par les tables avec contraintes sera effectué, et permettra ainsi une nouvelle phase de tests sur démo-client1.

En l'absence de contraintes techniques sur versions des serveurs de bases de données, les contraintes pourront etre misent en place sur l'ensemble des serveurs dès cette dernière étape effectuée.

2.2. Les Fonctions PL/PGSQL

Comme expliqué précédemment, les fonctions PL/PGSQL ne peuvent être mises en place directement sur les serveurs postgres des plates-formes, et seront donc mises en place sur le serveur HAGAKURE de la plate-forme demo-client1.

Il convient donc de fixer d'une part:

- le processus d'utilisation de cette plate-forme au sein du processus de production,
- les méthodes et directives de codage de ces fonctions afin de permettre leur réutilisation.

2.2.1. Les méthodes d'implémentation.

Actuellement, les requetes SQL sont implémentées directement dans les scripts php.

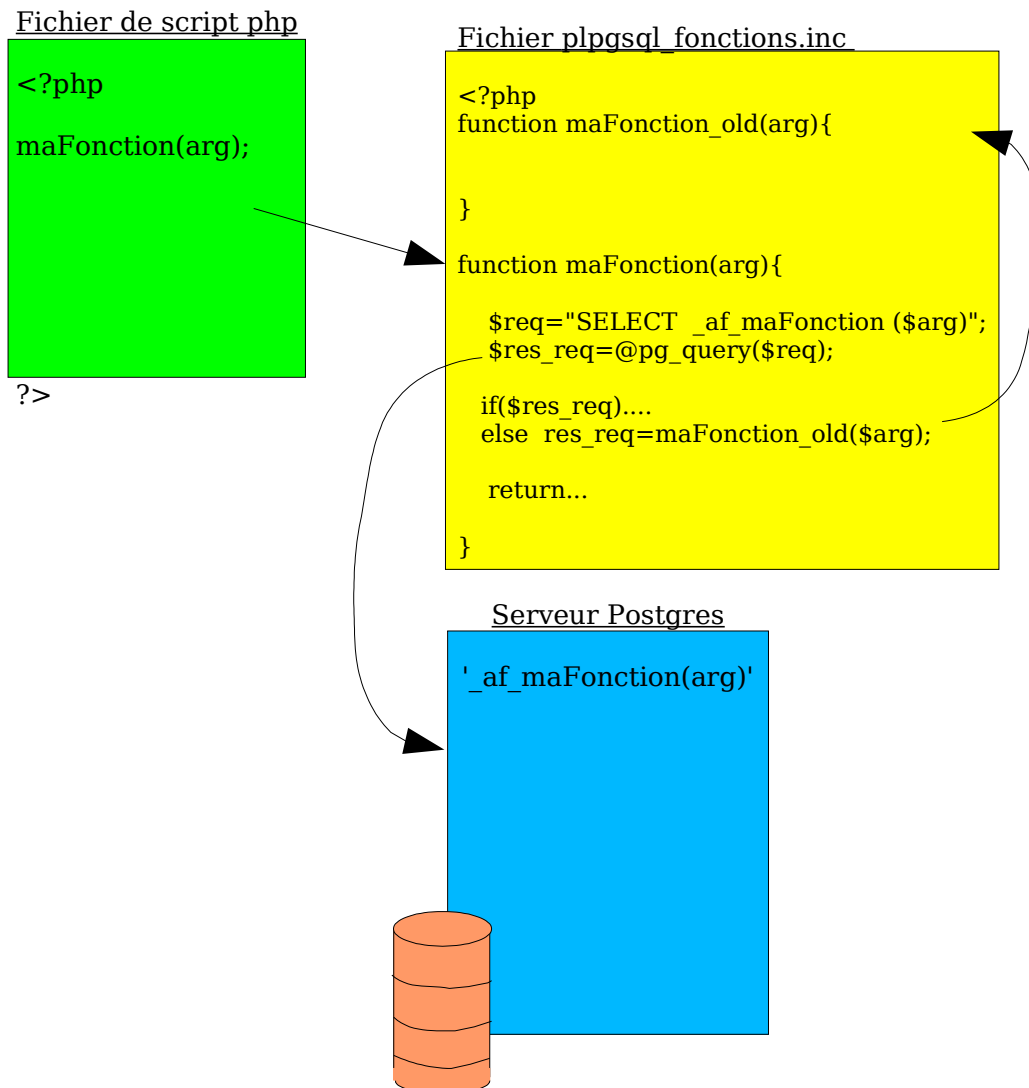
Afin de permettre la mise en place d'une documentation et une meilleure reutilisation des fonctions existantes et pour que les scripts fonctionnent sur la totalité des serveurs de bases de données, il convient de ne pas les remplacer directement par des appels aux fonctions PL/PGSQL, mais de passer par des fonctions php intermédiaires.

Ainsi, si l'on désire interroger la base de données dans un script php, il conviendra de le faire par l'appel à une fonction PHP qui devra répondre aux caractéristiques suivantes:

- la fonction php devra être implémentée dans un fichier nommé `plpgsql_fonctions.inc`. Ce fichier devra être placé dans le même répertoire que les scripts php.
- la fonction php devra porter le même nom que la fonction PL/PGSQL implémentée dans la base, sans `_af_` au début..
- La fonction php devra faire appel à la fonction PL/PGSQL et comporter un test sur sa bonne exécution. En cas d'erreur, c'est à dire dans le cas où la fonction PL/PGSQL n'est pas présente sur le serveur de base de données, la fonction php devra faire appel à une fonction de substitution, écrite en php, et qui permettra de faire le traitement requis, mais du côté client. Cette fonction devra être implémentée dans le fichier `plpgsql_fonctions.inc` et avoir le même nom, suffixé par `_old`.
- La fonction php utilisée dans les scripts devra retourner directement le résultat attendu s'il s'agit d'un résultat unique, ou l'identifiant résultat s'il s'agit de résultats multiples (plusieurs colonnes, plusieurs lignes,...) .

L'interrogation de la base de données se fera donc par l'intermédiaire de 3 fonctions, 2 en php, 1 en PL/PGSQL.

Par exemple, un script php pourrait faire appel une fonction nommée 'maFonction' implementée dans le fichier plpgsql_fonctions.inc du repertoire 'Parcours'. Cette fonction ferait donc appel à la fonction PL/PGSQL nommée '_af_maFonction', et si celle -ci ne peu s'executer, elle appellera la fonction php 'maFonction_old', dans le meme fichier.



2.2.2 Le processus de production

Dans un premier temps, il faut considérer que `démo-client` ne contient pas les toutes dernières versions des scripts php.

Il faut donc, avant de procéder à tout développement sur cette plate-forme, vérifier que les scripts à modifier sont bien à jour, sans risquer de perdre les développements précédents.

Afin d'éviter de malencontreuses manipulations, l'automatisation de cette procédure est elle aussi souhaitée.

Il a donc été convenu d'adapter un script de synchronisation pour qu'il réponde à ces besoins, ce qui se traduira, par la mise en place dans l'arborescence du serveur de production, d'un fichier contenant la liste des fichiers et répertoires à ne pas remplacer sur `démo-client1`.

Une fois les développements effectués et testés sur `démo-client1`, il conviendra de les reporter manuellement sur les serveurs de postproduction et de production, afin qu'ils soient pris en compte pour la prochaine synchronisation vers la totalité des plates-formes.

Ainsi, les fonctions PL/PGSQL ne seront pas encore présentes sur les serveurs de bases de données des plates-formes autres que `démo-client1`, mais les scripts php seront fonctionnels et prêts à leur mise en place.

3 Les scripts php

Toujours dans l'optique de clarifier les scripts et d'augmenter leur reutilisabilité, des travaux ont été effectués pour permettre d'éviter la présence de doublons entre le back-office et le front-office.

3.1 Regroupement des fichiers communs au back-office et au front-office

Si des fichiers viennent à servir dans les deux interfaces, il a été convenu:

- de les stocker dans l'arborescence du back-office,
- de conserver le classement actuel des scripts par thèmes dans des répertoires. Il n'est pas obligatoire que les arborescences des répertoires entre bo et fo soient identiques.
- de les placer dans le repertoire des scripts l'utilisant dans un sous répertoire nommé 'commun', afin de les différencier des autres scripts, qui restent spécifiques au bo.

Bien que l'exécution de scripts inter back-offices n'impose aucune contrainte, l'exécution de scripts du back-office à partir du front-office demande d'en respecter certaines.

En effet, l'accès aux fichiers du back-office est restreint par un accès sécurisé (.htaccess demandant un login et un mot de passe), et il convient donc de pouvoir exécuter ces scripts sans avoir à s'identifier et tout en maintenant cet accès restreint.

Présentement, une structure de fichiers a été mise en place sur plusieurs parties de la plate-forme et semble convenir à la plupart des cas:

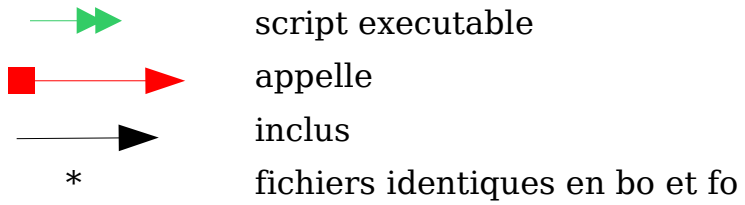
- les fichiers communs (répertoires 'commun' du bo) ne doivent être que des fichiers de traitements, ou des fichiers effectuant des affichages spécifiques et fonctionnels.
- 2 fichiers particuliers doivent être mis en place en bo et en front office, et doivent avoir les mêmes noms:
 - le premier fichier, dont le nom devra être en relation avec la fonctionnalité concernée (mafonctionnalité.php), aura un contenu différent selon qu'il soit en bo ou en fo. Les noms devront

etres cependant identiques. Il s'agit en quelque sorte d'un fichier template dont les fonctions seront:

- d'inclure les fichiers necessaires à l'execution des scripts (pas d'inclusion dans les fichiers communs),
 - d'afficher des informations communes à toutes les étapes de la fonctionnalité (entetes, pieds de pages,...)
 - d'inclure les fichiers de traitements contenant des affichages selon les différentes étapes de la fonctionnalité (gestion de l'ordre et du cheminement dans les scripts).
 - D'autres fichiers de template peuvent etres ajoutés en fonctions des besoins. Ainsi, un 3 eme nommé fenetre_mafonctionnalité.php est parfois présent en bo et fo pour tenir le meme rôle mais concernant uniquement l'affichage dans une fenetre popup.
- le second fichier, nommé porte_bo.php, devra etre présent en bo et en fo. Ces 2 fichiers donc, devront restés strictements identiques pour simplifier les mises à jour. Il servira à:
- inclure des fichiers sans affichages (et toujours sans inclusions),
 - appeler le fichier template pour continuer à evoluer dans la fonctionnalité.

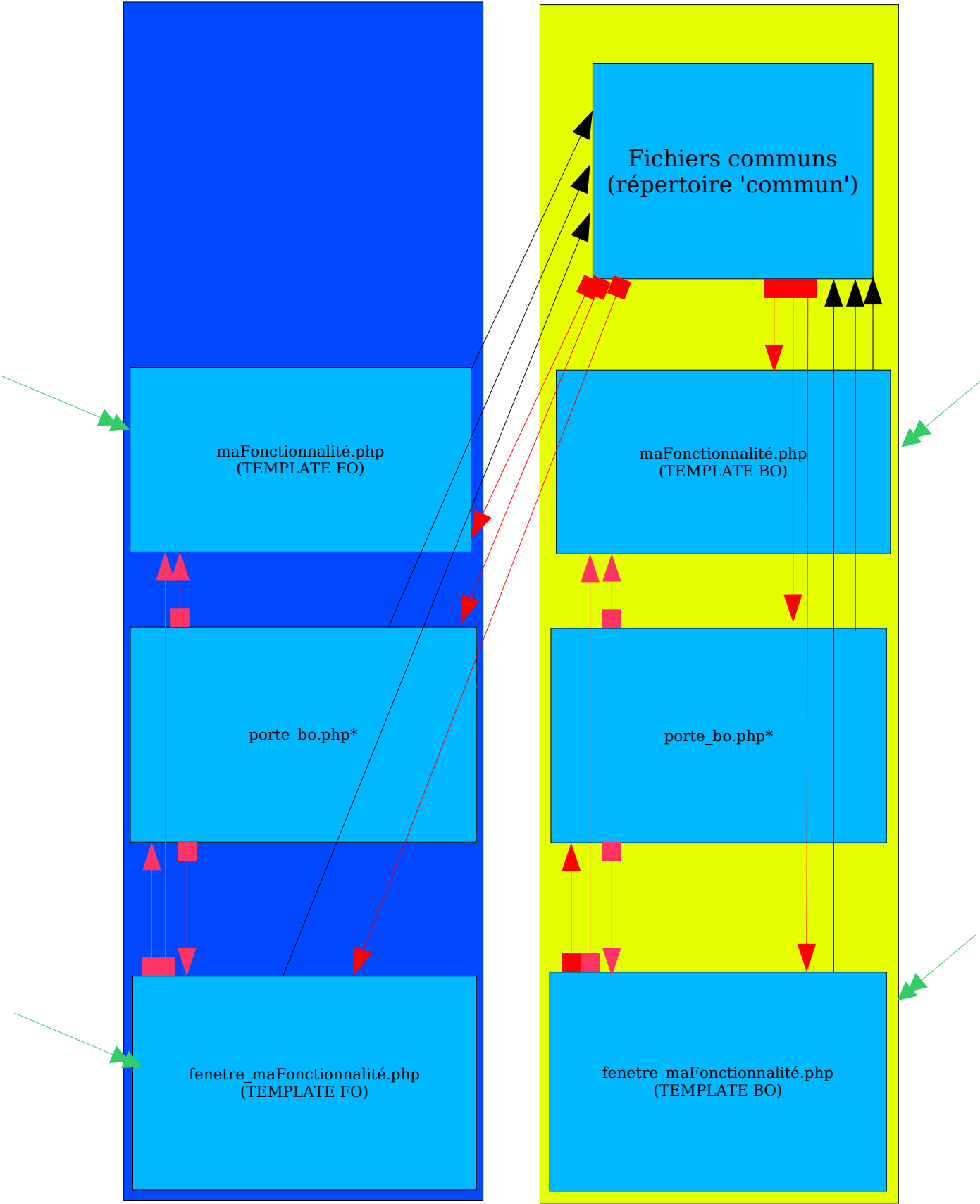
3.2 Exemple de structure de regroupement

Légende:



FRONT-OFFICE

BACK-OFFICE



3.4 Généralisation des entetes

La plupart des scripts php font appel à des fichiers nommés entete.php, enteteBack.php, etc...

Ces entetes sont généralement dynamiques, et dépendent de variables devant etre instanciées avant leur inclusion.

Il conviendra donc d'exploiter au maximum ces entetes, et de reporter toutes les fonctionnalités dans toutes les entetes qui peuvent en avoir besoin. A ces fin, une documentation de qualité permettra une utilisation optimale de leurs fonctionnalités.

Un certain nombre sont déjà implémentées:

- \$styleCSS permet de spécifier une feuille de style (CSS) pour l'ensemble de la page. Il a été convenu de placer ce fichier dans le repertoire des scripts (comme le fichier plpgsql_fonctions.php)
- \$javascriptFILE permet de spécifier un fichier php contenant uniquement des fonctions ou du code javascript. Il ne doit pas contenir de balises <SCRIPT>, celles ci étant déjà ajoutée par les entetes.

Ces fonctionnalités permettent d'allegier et de simplifier les scripts php, et aussi de respecter les normes html en vigueur : feuilles de style et scripts inclus dans les balises <head></head>, balises <body> spécifiques si pas de feuilles de style, etc...

Concernant les feuilles de style, la rédaction de chartes graphiques est prévue. Leur emploi systématique n'est cependant pas une priorité tant que les chartes n'ont pas été établies. Cependant, leur utilisation n'est pas à proscrire car elle permet d'une part, de tester leur comportement sous les différents navigateurs en vue de la redaction des chartes, et d'autre part de s'y familiariser. Il faut cependant prendre en compte que la plupart de ces feuilles de style seront modifiées une fois les chartes établies.

4 La documentation

Le systeme de documentation retenu par l'équipe de production est l'utilisation de PHPDoc (<http://www.phpdoc.de>).

Il s'agit d'un systeme Open Source de documentation automatique, inspiré de JavaDoc, sous licence PHP.

Ce systeme est donc utilisable à des fins commerciale et présente donc des possibilités d'adaptation et d'utilisation ouvertes.

Outre le fait de pouvoir disposer d'inventaires et d'explications des fonctions et des fichiers PHP en vue de leur réutilisation, la PHPDoc permet de disposer, en autres, des relations d'inclusion entre les fichiers.

Il est donc ainsi possible de savoir quels fichiers utilisent un fichier donné, ainsi que la liste de ceux dont il a besoin.

La PHPDoc a donc été mise en place sur la plate-forme de production, et est accessible à partir du back-office (Gestion des ressources administratives --> Utilitaires --> Documentation).

Il est ainsi possible de générer et de consulter cette documentation pour le back-office et le front-office.

Une documentation sur la totalité des scripts d'une plate-forme (bo et fo) n'est actuellement pas permise par PHPDoc, mais pourrait etre envisagée si le besoin s'en faisait ressentir, en modifiant les scripts de PHPDoc.

Cette documentation est générée par une analyse des commentaires des scripts php, et la qualité de son contenu est donc totalement lié à la qualité des commentaires de ces scripts.

L'équipe de production est seule responsable de l'utilisabilité de cette documentation, et a priori, seule destinatrice de son contenu. Il est cependant possible et envisageable que cette documentation viennent à etre transmettre aux eventuels acheteurs de la plate-forme.

Il est donc recommandé de rédiger systematiquement des commentaires pour chaque fichier php ainsi que pour chaque fonction. Il conviendra aussi dans un premier temps de tester l'interprétation des commentaires afin de bien appréhender la syntaxe des commentaires et ainsi permettre de disposer d'une documentation de qualité.

3.1 Fichiers pris en compte

La PHPDoc permet de paramétrer les extensions des fichiers à analyser. Elle est actuellement configurée pour prendre en compte les fichiers de type .php, .php3 et .inc.

Il est à noter que les scripts php contenant uniquement des fonctions devraient être suffixés par l'extension .inc, et les autres par .php, l'extension .php3 étant amenée à disparaître.

Les commentaires des fonctions javascripts sont eux aussi pris en compte s'ils sont écrits dans des fichiers de type .php ou .inc. Il est cependant recommandé de suffixer ces fichiers par .php afin d'éviter toute confusion.

3.2 Format des commentaires

En l'absence de documentation précise sur les formats à utiliser et sur les méthodes d'analyse des scripts par la PHPDoc, il est donc fortement souhaitable que cette partie de ce document évolue aux fur et à mesure des constatations faites par l'équipe de production.

Les recommandations qui vont suivre seront donc établies à partir d'observations et de tests, et il est donc souhaitable les tester et de les vérifier par soi même, et de reporter les éventuelles modifications dans présent document.

Ainsi:

Un commentaire interprétable par la PHPDoc doit être du format suivant :

```
/**
```

Courte description sur 1 SEULE ligne

Longue description sur PLUSIEURS LIGNES

```
@param type description param1
```

```
@param type description param2
```

```
@param type description param3
```

```
@return type description (pas géré par phpdoc)
```

```
*/
```

Tout commentaire commençant par '/*' et terminant par '*/' sera donc traité comme un commentaire PHPDoc. Il conviendra donc de réserver cette syntaxe aux seuls commentaires PHPDoc, et donc d'utiliser '//' ou '/* */' pour les autres commentaires.

Ces commentaires doivent être placés avant les fonctions ainsi qu'au début de chaque fichier.