

Table des matières

1	Principes	2
1.1	La représentation des données	2
1.2	Les méthodes de manipulation de cette donnée	2
2	Accès aux données en mode objet	2
2.1	Complétion automatique et découverte des API	3
2.2	Accès à des données simples de premier niveau	3
2.3	Sous-objets	4
2.4	Tableaux	4
3	Recherche de données en base via « find »	4
4	Accès aux données en mode requête via une agrégation	5
4.1	Le principe de l'agrégation	5
4.2	API d'exécution	5
4.3	Exemple d'utilisation d'une agrégation	5
5	Génération de formulaires	6
6	API du modèle Objet d'accès aux données	6
6.1	Informations	6
6.2	Lecture des données	8
6.3	Génération de formulaires et d'affichage HTML	8
6.4	Ecriture des données	11
7	Exemples	12
7.1	Formulaire simple d'accès à des données de premier niveau	12
7.2	Récupération d'une liste de champs et affichage du formulaire	13
7.3	Création d'un nouvel enregistrement en programmation pure	13
7.4	Chargement d'un objet précis et affichage du formulaire correspondant	13
7.5	Suppression d'une donnée d'un tableau	14
7.6	Utilisation de la classe d'agrégation pour l'extraction de données selon une pile fournie :	14

	Application ARIA3	Annexe 4 CCTP Date : 05/06/2026 Version : 7
	API d'accès aux données	

1 Principes

Le principe général qui sous-tend l'application ARIA3 est de permettre de manipuler n'importe quelle forme de donnée, quel que soit son niveau hiérarchique, et ce sans difficultés ou contraintes techniques importantes. Ce principe est à mettre en parallèle aux développements de cette application sur base d'une base de données documentaire (MongoDB), qui permet de stocker des données organisées de manière hiérarchique.

Dans le cadre de ARIA3, la très large majorité des données est organisée sous forme hiérarchique, avec une relation 1-1 ou 1-n avec les nœuds parents. L'approche d'une application universelle d'accès aux données hiérarchique est donc bien adaptée.

Le système, pour fonctionner, s'appuie sur deux piliers essentiels :

1.1 La représentation des données

Chaque donnée est décrite de manière très précise. Cette description est sauvegardée dans une collection MongoDB dédiée au stockage de ce modèle.

La référence des modèles de données pour l'ensemble de l'application est contenue dans un fichier de type « Mindmap » au format Freemind qui décrit précisément chacune des collections, les champs qu'elle contient et comment ils sont structurés (sous-objets, tableaux...).

Chaque collection possède un certain nombre d'attributs (décrits dans l'annexe 3) définissant des paramètres généraux relatifs à la collection (nom...).

Chaque collection possède un nombre donné de champs. Chaque champ est décrit de manière exhaustive : format, contraintes d'utilisation (valeurs limites...).

1.2 Les méthodes de manipulation de cette donnée

Pour chaque format de donnée existant dans l'application, un modèle est créé. Il permet, en fonction des contraintes décrites pour chaque champ du modèle, de définir les règles d'affichage et de manipulation de la donnée :

- quelle valeur doit apparaître et sous quelle forme ;
- quels champs de formulaire et quelles méthodes JavaScript doivent être utilisées pour gérer le fonctionnement de chaque champ.

Ces méthodes permettent d'accéder facilement et rapidement à chaque champ existant dans une collection, de modifier son contenu, de stocker et gérer les données en base (modification, effacement, insertion).

Les méthodes disponibles facilitent voire automatisent la mise en place d'un « CRUD » pour l'accès aux données, malgré leur forme hiérarchique.

2 Accès aux données en mode objet

Le modèle objet mis en place permet une instanciation et un accès aux données direct et facile.

Certaines méthodes permettent un accès direct à la donnée stockée, d'autres permettent de manipuler la donnée visible.

	Application ARIA3	Annexe 4 CCTP Date : 05/06/2026 Version : 7
	API d'accès aux données	

En effet selon le format du champ, la donnée présentée à l'utilisateur n'est pas forcément la même que celle enregistrée en base. Exemples et contre-exemples :

Un nom d'utilisateur : présentée en formulaire sous la forme d'un champ texte, l'utilisateur manipule en général directement la vraie donnée.

Une date : en raison de la manipulation particulière qui est faite, les dates sont très rarement gérées en base dans le format dans lequel elles sont affichées.

- exemple d'un format « sql » classique : « 2013-03-31 12 :35 :56 »
- la même date sera présentée à l'utilisateur de cette manière : « 31/3/2013 12h35m56s »

2.1 Complétion automatique et découverte des API d'accès aux données

Le modèle de données de ARIA3 est décrit de manière abstraite via un fichier descripteur. Ce modèle est matérialisé par exemple lors de la sauvegarde d'un objet, à l'exécution, mais n'est pas décrit matériellement dans les objets métier.

Concrètement ce mode de fonctionnement ne permet donc pas, normalement, de disposer d'une complétion automatique du code pour explorer les champs de données natifs.

Un système d'aide à la saisie a été mis en place par l'intermédiaire d'un fichier d'annotations, que chaque développeur peut récupérer via une instance de ARIA3, et qui apporte, via les fonctionnalités de son environnement de développement, une complétion automatique.

2.2 Accès à des données simples de premier niveau

Les propriétés de premier niveau sont accessibles simplement après chargement d'un objet.

Il faut dans un premier temps instancier et charger un objet de données basé sur une collection existante, par exemple la collection « mon_objet » :

```
$objet=new mon_objet();
```

Puis charger la donnée :

```
$objet->load($identifiant_mongo_db);
```

Note : l'identifiant utilisé pour le chargement est nécessairement un identifiant type Mongold.

Enfin l'accès au champ lui-même se fait selon les deux manières suivantes :

La valeur réelle en base :

```
$valeur = $objet->mon_champ->get();
```

La valeur qui doit être affichée à l'utilisateur :

```
$valeur = $objet-> mon_champ->getValue();
```

Ces deux valeurs sont parfois identiques, pour les champs texte, les champs contenant des valeurs numériques par exemple. Elles sont en revanche différentes quand le champ contient une référence, par exemple dans le cas d'un champ « commune » : la valeur stockée sera un code, la valeur affichée à l'utilisateur étant un nom de commune.

2.3 Sous-objets

Le cas d'un sous-objet est un peu plus indirect, mais similaire. Pour le modèle de données suivant par exemple :

Mon_objet -> mon_sous_objet -> mon_champ : texte

La première étape est similaire :

```
$objet=new mon_objet();
$objet->load($identifiant_mongo_db);
```

Mais il faut maintenant accéder au contenu de l'objet, cela se fait avec la méthode « getLigne ». Après cette méthode le reste est similaire :

```
$valeur = $objet->mon_sous_objet->getLigne()->mon_champ->getRealValue();
```

2.4 Tableaux

Enfin le cas des tableaux est extrêmement similaire aux objets, à la différence qu'il est nécessaire d'indiquer à quelle ligne on veut accéder :

```
$listeLignes = $obj->mon_sous_objet->listeLignes();
// ... Boucle sur les lignes et obtention d'un numéro de ligne $numligne :
$valeur = $objet->mon_sous_objet->getLigne($numligne)->mon_champ->get();
```

Comme pour les identifiants de collections, les numéros de lignes sont nécessairement des identifiants Mongold.

L'unicité de ces identifiants est très importante dans le contexte de gestion de tableaux dans l'application, car il permet d'identifier une entrée de manière sure et unique au sein de l'application.

3 Recherche de données en base via « find »

La méthode « find » est la méthode standard de recherche de données, son utilisation est identique à la fonction « find » décrite dans la documentation « MongoDB ». Dans ARIA3, l'implémentation de la méthode « find » permet de retourner des objets standards et de boucler sur chaque objet de l'échantillon.

Le principe de base est le suivant :

```
// Instancier un nouveau objet
$objet = new mon_objet();
// Recherche selon un critère donné
// Le tableau fourni en paramètre est la requête de recherche, le find retourne le nombre de résultats trouvés
$nbResultats = $objet->find(["monchamp" => "mavaleur"]);
do {
    echo $objet->monchamp->get(); // Affichage d'une valeur
} while($objet->next());
```

4 Accès au données en mode requête via une agrégation

4.1 Le principe de l'agrégation

L'agrégation est un principe spécifique à MongoDB qui consiste à créer une pile d'exécution, chaque élément de la pile réalisant un traitement sur les données transmises par l'élément exécuté précédemment.

Des méthodes d'exécution d'agrégations sont implémentées dans ARIA3, selon le principe d'exécution elles ne retournent par des objets instanciés mais des tableaux bruts de données.

Dans certains cas il est nécessaire de collecter et de travailler sur des données dans un format brut, afin d'accélérer les traitements significativement, car le mode objet présenté précédemment, s'il permet de manipuler des données avec certaines facilités, reste très lent pour des traitements de masse.

L'agrégation est l'équivalent d'une requête SQL dans les bases relationnelles, elle permet au final d'obtenir, selon certaines contraintes et critères, des tableaux de données contenant des valeurs prétraitées : sommes, synthèses...

L'agrégation permet par exemple d'extraire très rapidement des listes de données 1-n, par exemple causes, ou types de phénomènes, de les mettre en table, de réaliser des synthèses dessus (somme, totaux), et de fournir le résultat en tableau.

4.2 API d'exécution

L'application possède une classe spécifique facilitant l'exécution de fonctions d'agrégation : « AggregateurMongo ». Cette classe simplifie les opérations en :

- Fournissant une initialisation directe de l'objet d'agrégation
- Permettant de fournir sous forme de paramètres simplifiés différents types d'éléments de pile : filtre, éclatements de données, groupes, projection de champs
- Et en facilitant la navigation dans la données sous forme d'un itérateur

4.3 Exemple d'utilisation d'une agrégation

Dans l'exemple ci-dessous on prépare une pile d'exécution basée sur plusieurs opérateurs d'agrégation standards : match, unwind, project (cf. documentation officielle mongod), puis on exécute l'agrégation et on récupère le résultat sous la forme d'un tableau.

```
$pile=array();
$pile[]=array('$match'=>array('_id'=> MongoDBDatatypes::Id($evtid),$chemin.'._id'=>$lineid));
foreach($arrCheminsTableaux as $curCheminTableau)
  $pile[]=array('$unwind'=>'$'.$curCheminTableau);
$pile[]=array('$match'=>array('_id'=> MongoDBDatatypes::Id($evtid),$chemin.'._id'=>$lineid));
$pile[]=array('$match'=>array($chemin.'._id'=>$lineid));
$pile[]=array('$project'=>array("ref"=>'$'.$chemin.'._reference_id'));
$resultat= AggregateurMongo::quickExecute($curCollection, $pile);
echo $resultat[0]["ref"]; // exemple d'utilisation
```

5 Génération de formulaires

La création de formulaires est facilitée dans le cadre de l'application ARIA3.

Les données sont décrites dans le modèle des données. Pour chaque collection un objet existe. Il est alors possible, une fois instancié, d'obtenir :

- la liste des champs visibles ;
- la liste des champs dans un contexte donné.

Puis pour chaque champ :

- la valeur réelle enregistrée en base ;
- la valeur affichée ;
- le formulaire d'édition de cette donnée.

6 API du modèle Objet d'accès aux données

6.1 Informations

6.1.1 « champsAffichage(\$contexte) » (collection, champs)

Public function champsAffichage(\$contexte)

6.1.1.1 Paramètres

Chaine de caractère : contexte dans lequel on veut connaître les champs affichés. Trois contextes possibles : « detail », « form », « liste ».

6.1.1.2 Retourne

Tableau des champs visibles dans le contexte, avec leur ordre d'affichage :

- la clef du tableau est le code du champ ;
- la valeur du tableau est l'ordre d'affichage.

6.1.1.3 Raccourcis

Valables uniquement pour les collections :

- public function champsAffichageListe()
- public function champsAffichageDetail()
- public function champsAffichageForm()

6.1.2 « champsCaches() » (collection, champs)

6.1.2.1 Paramètres

Aucun

6.1.2.2 Retourne

Tableau des champs invisibles dans le contexte, avec leur ordre d'affichage :

- la clef du tableau est le code du champ ;
- le tableau contient systématiquement la valeur 1.

	Application ARIA3	Annexe 4 CCTP Date : 05/06/2026 Version : 7
	API d'accès aux données	

6.1.3 [« isField\(&\\$field\) » \(collection, champs\)](#)

public function isField(&\$field)

6.1.3.1 Paramètres

Le champ que l'on souhaite tester

6.1.3.2 Retourne

Booléen indiquant si le champ transmis est un objet de type « champ » (field) permettant l'appel des méthodes de champs.

6.1.4 [« isArray\(\\$codechamp\) » \(collection, champs\)](#)

public function isArray(\$codechamp)

6.1.4.1 Paramètres

Le code du champ que l'on souhaite tester

6.1.4.2 Retourne

Booléen indiquant si le champ transmis est un tableau

6.1.5 [« isObject\(\\$codechamp\) » \(collection, champs\)](#)

public function isObject(\$codechamp)

6.1.5.1 Paramètres

Le code du champ que l'on souhaite tester

6.1.5.2 Retourne

Booléen indiquant si le champ transmis est un objet

6.1.6 [« isScalaire\(\\$typechamp\) » \(collection\)](#)

public static function isScalaire(\$typechamp)

6.1.6.1 Paramètres

Le format du champ que l'on souhaite tester

6.1.6.2 Retourne

Booléen indiquant si le champ transmis est une valeur simple : ni objet, ni tableau

6.1.7 [« isCollection\(\) » \(collection, champ\)](#)

public function isCollection()

6.1.7.1 Paramètres

Aucun

6.1.7.2 Retourne

Booléen indiquant si le champ transmis est une collection

6.1.8 [« getPath \(\) » \(collection, champ\)](#)

public function getPath()

6.1.8.1 Paramètres

Aucun

	Application ARIA3	Annexe 4 CCTP Date : 05/06/2026 Version : 7
	API d'accès aux données	

6.1.8.2 *Retourne*

Texte indiquant le chemin absolu définissant la position de l'objet dans l'arborescence des données

6.2 *Lecture des données*

6.2.1 « getId() » (collection)

public function getId()

6.2.1.1 *Paramètres*

Aucun

6.2.1.2 *Retourne*

Retourne la valeur de l'identifiant MongoDB pour l'enregistrement actuellement chargé dans la collection.

6.2.2 « getLigne(\$ligne) » (champs)

6.2.2.1 *Paramètres*

Si le parent est un objet, aucun paramètre

Si le parent est un tableau : le numéro (mongoid) de la ligne est à transmettre en paramètre

6.2.2.2 *Retourne*

Un objet contenant les données situées sur la ligne. Le parent peut être objet ou tableau, la forme de la donnée retournée est la même.

6.2.3 « getRealValue() » (champs)

6.2.3.1 *Paramètres*

Aucun

6.2.3.2 *Retourne*

La vraie valeur interne du champ. Cf. « setRealValue() »

6.2.4 « getValue() » (champs)

6.2.4.1 *Paramètres*

Aucun

6.2.4.2 *Retourne*

La valeur utilisateur du champ. Cf. « setValue() »

6.3 *Génération de formulaires et d'affichage HTML*

6.3.1 « getAffichage() » (champs)

6.3.1.1 *Paramètres*

Aucun

6.3.1.2 *Retourne*

Code HTML contenant le libellé et la valeur « utilisateur » du champ, encadrés par un balisage HTML de mise en situation.

	Application ARIA3	Annexe 4 CCTP Date : 05/06/2026 Version : 7
	API d'accès aux données	

Cette méthode est similaire à la méthode « getFormulaire() » mais pour la visualisation de la donnée avec son libellé. Elle est utilisée pour générer rapidement des formulaires où le champ doit être affiché avec son libellé.

6.3.2 [« getFormulaire\(\) » \(champs\)](#)

6.3.2.1 Paramètres

Aucun

6.3.2.2 Retourne

Code HTML contenant le libellé et le formulaire « utilisateur » du champ, encadrés par un balisage HTML de mise en situation.

Cette méthode est similaire à la méthode « getAffichage() » mais pour l'édition de la donnée.

Elle est utilisée pour générer rapidement des formulaires où le champ doit être affiché avec son libellé.

6.3.3 [« formGetCollectionName \(\) » \(collection\)](#)

public function formGetCollectionName()

6.3.3.1 Paramètres

Aucun

6.3.3.2 Retourne

Fournit un champ caché de formulaire permettant d'identifier la collection manipulée dans le cadre du formulaire affiché.

L'utilisation de ce champ de formulaire est requise pour pouvoir utiliser la méthode « saveForm() » qui automatise la sauvegarde de données du formulaire.

6.3.4 [« champsValeurCompleet \(\) » \(collection\)](#)

public function champsValeurCompleet(\$curchamp)

6.3.4.1 Paramètres

Aucun

6.3.4.2 Retourne

Code HTML contenant la valeur « utilisateur » du champ encadré par le balisage HTML permettant de le mettre en situation.

6.3.5 [« champsCachesFormulaire \(\\$curchamp\) » \(collection\)](#)

public function champsCachesFormulaire(\$curchamp)

6.3.5.1 Paramètres

Nom du champ

6.3.5.2 Retourne

Code HTML du champ caché du champ fourni en paramètre

6.3.6 [« champsFormulaireCompleet \(\\$curchamp\) » \(collection\)](#)

public function champsFormulaireCompleet(\$curchamp)

	Application ARIA3	Annexe 4 CCTP Date : 05/06/2026 Version : 7
	API d'accès aux données	

6.3.6.1 Paramètres

Code du champ

6.3.6.2 Retourne

Code HTML contenant le formulaire permettant d'éditer la valeur « utilisateur » du champ encadré par le balisage HTML permettant de le mettre en situation.

6.3.7 [« champsFormulaire \(\\$curchamp\) » \(collection\)](#)

public function champsFormulaire(\$curchamp)

6.3.7.1 Paramètres

Nom du champ souhaité

6.3.7.2 Retourne

Le formulaire d'édition de la donnée, seul sans balisage HTML de mise en situation.

6.3.8 [« champsValeur \(\\$curchamp\) » \(collection\)](#)

public function champsValeur(\$curchamp)

6.3.8.1 Paramètres

Code du champ

6.3.8.2 Retourne

Code HTML contenant la valeur « utilisateur » du champ.

6.3.9 [« champsLibelle \(\\$curchamp\) » \(collection\)](#)

public function champsLibelle(\$curchamp)

6.3.9.1 Paramètres

Code du champ

6.3.9.2 Retourne

Code HTML contenant la valeur « utilisateur » du libellé du champ

6.3.10 [« getLibelle\(\) » \(champs\)](#)

6.3.10.1 Paramètres

Aucun

6.3.10.2 Retourne

Code HTML du libellé du champ seul, avec mise en situation

6.3.11 [« getChamp\(\) » \(champs\)](#)

6.3.11.1 Paramètres

Aucun

6.3.11.2 Retourne

Code HTML du champ seul sous forme d'un formulaire, avec mise en situation

	Application ARIA3	Annexe 4 CCTP Date : 05/06/2026 Version : 7
	API d'accès aux données	

6.3.12 [« getField\(\) » \(champs\)](#)

6.3.12.1 Paramètres

Aucun

6.3.12.2 Retourne

Code HTML du champ seul sous forme d'un formulaire, sans code HTML de mise en situation

6.3.13 [« getChampHidden\(\\$params\) » \(champs\)](#)

6.3.13.1 Paramètres

Paramètres optionnels qui seront intégrés dans la balise HTML

6.3.13.2 Retourne

Code HTML du champ caché, pour insertion dans un formulaire

6.4 Ecriture des données

6.4.1 [« setRealValue\(\\$valeur\) » \(champs\)](#)

6.4.1.1 Paramètres

La valeur à sauvegarder. Celle-ci doit être cohérente avec le champ

6.4.1.2 Retourne

La vraie valeur interne du champ

6.4.2 [« setValue\(\\$valeur\) » \(champs\)](#)

6.4.2.1 Paramètres

La valeur utilisateur à inscrire dans le champ. Lors de l'enregistrement cette valeur est, si besoin, transformée dans la valeur « interne » du champ.

6.4.2.2 Retourne

La valeur utilisateur du champ

6.4.3 [« parse\(\\$valeur\) » \(champs\)](#)

6.4.3.1 Paramètres

Une valeur à vérifier

6.4.3.2 Retourne

La valeur mise en forme pour s'accorder aux contraintes techniques du champ. Exemple : pour un objet « mongoid », il est possible d'appeler la méthode « parse » avec une valeur d'identifiant mongoid valide mais sous différents formats : texte, objet, objet de classe mongoid. De manière à s'assurer d'une consistance des données sauveées, la méthode « parse » analyse la donnée entrée et fournit en retour une valeur consolidée, ou « false » si le format de la donnée n'était pas bon.

Idem pour les champs « date » : la date en entrée peut avoir plusieurs formats : utilisateur, sql, mongodate... La méthode détecte le format et retourne le temps unix, ou false.

	Application ARIA3	Annexe 4 CCTP Date : 05/06/2026 Version : 7
	API d'accès aux données	

6.4.4 « [getChampEffacement\(\\$theLigne\)](#) » (champs)

public function getChampEffacement(\$theLigne)

6.4.4.1 Paramètres

Identifiant de la ligne à effacer

6.4.4.2 Retourne

Code HTML du champ caché permettant de marquer une ligne à l'effacement

6.4.5 « [save \(\)](#) » (collection)

public function save()

6.4.5.1 Paramètres

Aucun

6.4.5.2 Retourne

Appliqué sur une collection, sauve les valeurs présentes dans l'objet. La sauvegarde crée un nouvel enregistrement ou met à jour l'enregistrement si l'_id est déjà présent en base.

6.4.6 « [saveForm \(\)](#) » (collection)

public function saveForm()

6.4.6.1 Paramètres

Aucun

6.4.6.2 Retourne

Cette méthode permet d'automatiser la sauvegarde des données transmises par formulaire. Un certain nombre de champs sont requis pour que cela fonctionne :

- le nom de la collection : obtenu à l'aide de la méthode décrite précédemment, ce champ permet à l'objet de vérifier que les données se situent dans son contexte. Cela permet d'utiliser la méthode saveForm sur plusieurs objets en s'assurant qu'aucune donnée ne soit stockée dans les collections non concernées ;
- l'identifiant de l'enregistrement : champ de formulaire caché dont le name vaut « _id » et contient un identifiant mongoid.

Les champs supplémentaires contenus dans le formulaire sont « mappés » avec le modèle de données et sauvegardés en base après vérification de la validité de la donnée sauvegardée.

7 Exemples

7.1 Formulaire simple d'accès à des données de premier niveau

A partir d'une collection donnée, création des champs requis pour la création d'un nouvel enregistrement dans un formulaire :

```
<form class="datalist" action="testformedition.php" id="adm_membre2" method="post"
enctype="multipart/form-data">
<?php
    $membre=new adm_membre();
    echo $membre->formGetCollectionName();
```

```

        echo $membre->_id->getChampHidden();
        echo $membre->initiale->getChamp();
    ?>
    <input type="submit" name="cmdSauver" value="Sauver">
</form>

```

7.2 Récupération d'une liste de champs et affichage du formulaire

Appel de la méthode permettant de récupérer les champs visibles pour un objet puis affichage des formulaires pour cet objet.

```

$champsprio=$composant->dependences->getLigne($keyligne)->priorite->champsAffichage();
$composant->save();
foreach($champsprio as $curchamp => $aff)
{
    if($aff)
    {
        echo $composant->dependences->getLigne($keyligne)->priorite-
->getLigne($keyligne2)->$curchamp->getChamp()."<br/>";
    }
}

```

7.3 Création d'un nouvel enregistrement en programmation pure

Dans une collection « adm_composants » contenant une propriété « classe », un objet « positions » contenant quatre propriétés numériques : left, right, footer, header, et un tableau d'objet « vues » contenant une propriété textuelle : « code »

```

$composant=new adm_composants();
$composant->classe="test";
// Ecriture du paramètre au travers de la méthode getLigne sans paramètres
$composant->positions->getLigne()->left->setRealValue(1);
$composant->positions->getLigne()->header->setRealValue(2);
$composant->positions->getLigne()->right->setRealValue(3);
$composant->positions->getLigne()->footer->setRealValue(4);
// On crée une nouvelle ligne, on récupère son identifiant
$keyligne=$composant->vues->addLigne();
// Ecriture du paramètre au travers de la méthode getLigne avec paramètre
$composant->vues->getLigne($keyligne)->code->setRealValue("toto");
// Les données ont été inscrites dans l'objet, on sauve
$composant->save();

```

7.4 Chargement d'un objet précis et affichage du formulaire correspondant

Le formulaire résultant permet l'affichage et l'édition de la donnée.

```

$membre=new adm_membre();
$membre->saveForm();
$membre->load("50993e97f94bd4e42b013fa3");
?>

```

```
<form class="datalist" action="testformedition.php" id="adm_membre2" method="post"
enctype="multipart/form-data">
<?php echo $membre->formGetCollectionName(); ?>
<?php echo $membre->_id->getChampHidden(); ?>
<?php echo $membre->initiale->getChamp(); ?>
<input type="submit" name="cmdSauver" value="Sauver">
</form>
```

7.5 Suppression d'une donnée d'un tableau

La suppression d'une donnée d'un tableau se fait en ajoutant un champ de formulaire « _del » avec pour valeur « 1 » au niveau de la ligne de tableau à effacer.

```
echo $composant->vues->getChampEffacement($keyligne) ;
```

7.6 Utilisation de la classe d'agrégation pour l'extraction de données selon une pile fournie :

```
$pile=array(
    array('$match'=>$this->requeteMongo),
    array('$unwind'=>array('path'=>'$ressources',"preserveNullAndEmptyArrays"=>false)),
    array('$unwind'=>array('path'=>'$ressources.lexiques',"preserveNullAndEmptyArrays"=>false
)),
    array('$project'=>array('id_evt'=>1,'valeur'=>'$'.$chemin)),
    array('$match'=>array("valeur"=>MongoDatatypes::Regex("/^lex_equipements/i"))),
);
$result= AggregateurMongo::quickExecute($collection, $pile);
if(is_array($result) && count($result))
{
    foreach($result as $curValeur)
    {
        // Traitement des valeurs : $curValeur["valeur"];
    }
}
```