plugins

Contents

Package default Procedural Elements

afficher_formulator.php

- **Package** default

array function plugin_catalogue_catalogage_grilles_actions_grilles_afficher_formulator($parametres, 1, 2, 3, 4)
[line 13]

Function Parameters:

- *mixed* **$parametres**
- *[infos]* **1** => [type_element], [nom_champ], [nom_ss_champ], [idx_onglet], [idx_champ], [idx_ss_champ]
- *[ID_element]* **2**
- *[ID_operation]* **3**
- *[action]* **4**

plugin_catalogue_catalogage_grilles_actions_grilles_afficher_formulator()

ajouter_champ.php

- **Package** default

array function plugin_catalogue_catalogage_grilles_actions_grilles_ajouter_champ($parametres, 1, 2, 3, 4) *[line 13]*

Function Parameters:

- *mixed* **$parametres**
- *[idx_onglet]* **1**
- *[ID_operation]* **2**
- *[nom_champ]* **3**
- *[action]* **4**

plugin_catalogue_catalogage_grilles_actions_grilles_ajouter_champ()

ajouter_ss_champ.php

- **Package** default

array function plugin_catalogue_catalogage_grilles_actions_grilles_ajouter_ss_champ($parametres, 1, 2, 3, 4, 5)
[line 14]

Function Parameters:

- *mixed* **$parametres**
- *[infos]* **1** => [type_element], [nom_champ], [nom_ss_champ], [idx_onglet], [idx_champ], [idx_ss_champ], [ID_parent]
- *[ID_element]* **2**
- *[ID_operation]* **3**
- *[auto_plugin]* **4** // plugin à utiliser pour ajouter le ss-champ
- *[action]* **5**

plugin_catalogue_catalogage_grilles_actions_grilles_ajouter_ss_champ()

- **Package** default

Ce function plugin_catalogue_catalogage_grilles_actions_grilles_get_liste_champs($parametres, 1, 2) *[line 17]*

Function Parameters:

- *mixed* **$parametres**
- *[idx_onglet]* **1**
- *[plugin_onglet]* **2**

plugin_catalogue_catalogage_grilles_actions_grilles_get_liste_champs()

get_liste_ss_champs.php

- **Package** default

Ce function plugin_catalogue_catalogage_grilles_actions_grilles_get_liste_ss_champs($parametres, 1, 2) *[line 17]*

Function Parameters:

- *mixed* **$parametres**
- *[ID_element]* **1**
- *[plugin_champ]* **2** // défini dans le registre (switchers)

plugin_catalogue_catalogage_grilles_actions_grilles_get_liste_ss_champs()

monter_descendre_champ.php

- **Package** default

array function plugin_catalogue_catalogage_grilles_actions_grilles_monter_descendre_champ($parametres, 1, 2, 3, 4, 5) *[line 16]*

Function Parameters:

- *mixed* **$parametres**
- *[infos]* **1** => [type_element], [nom_champ], [nom_ss_champ], [idx_onglet], [idx_champ], [idx_ss_champ], [ID_parent]
- *[ID_element]* **2**
- *[ID_operation]* **3**
- *[action]* **4**
- *[sens]* **5** // "monter" ou "descendre"

plugin_catalogue_catalogage_grilles_actions_grilles_monter_descendre_ss_champ()

monter_descendre_ss_champ.php

- **Package** default

array function plugin_catalogue_catalogage_grilles_actions_grilles_monter_descendre_ss_champ($parametres, 1, 2, 3, 4, 5) *[line 16]*

Function Parameters:

- *mixed* **$parametres**
- *[infos]* **1** => [type_element], [nom_champ], [nom_ss_champ], [idx_onglet], [idx_champ], [idx_ss_champ], [ID_parent]
- *[ID_element]* **2**
- *[ID_operation]* **3**
- *[action]* **4**
- *[sens]* **5** // "monter" ou "descendre"

plugin_catalogue_catalogage_grilles_actions_grilles_monter_descendre_ss_champ()

- **Package** default

array function plugin_catalogue_catalogage_grilles_actions_grilles_simple_validation($parametres, 1, 2, 3, 4) *[line 14]*

Function Parameters:

- *mixed* **$parametres**
- *[infos]* **1** => [type_element], [nom_champ], [nom_ss_champ], [idx_onglet], [idx_champ], [idx_ss_champ]
- *[ID_element]* **2**
- *[ID_operation]* **3**
- *[action]* **4**

plugin_catalogue_catalogage_grilles_actions_grilles_simple_validation()

suppression_element.php

- **Package** default

array function plugin_catalogue_catalogage_grilles_actions_grilles_suppression_element($parametres, 1, 2, 3, 4)
[line 13]

Function Parameters:

- *mixed* **$parametres**
- *[infos]* **1** => [type_element], [nom_champ], [nom_ss_champ], [idx_onglet], [idx_champ], [idx_ss_champ], [ID_parent]
- *[ID_element]* **2**
- *[ID_operation]* **3**
- *[action]* **4**

plugin_catalogue_catalogage_grilles_actions_grilles_suppression_element()

validation_lien_explicite.php

- **Package** default

array function plugin_catalogue_catalogage_grilles_actions_grilles_validation_lien_explicite($parametres, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11) *[line 24]*

Function Parameters:

- *mixed* **$parametres**
- *[infos]* **1** => [type_element], [nom_champ], [nom_ss_champ], [idx_onglet], [idx_champ], [idx_ss_champ]
- *[ID_element]* **2**
- *[ID_operation]* **3**
- *[action]* **4**
- *[param_lien_explicite]* **5** : toutes les infos pour générer le lien explicite, à savoir :
- *[type]* **6** => le type d'objet lié (biblio, auteur...)
- *[plugin_formate]* **7** => le plugin utilisé pour mettre en forme le champ à partir de a notice liée
- *[plugin_get_lien_explicite]* **8** => plugin utilisé pour récupérér la notice liée, et la mettre en forme grâce au plugin précédent
- *[ss_champs_a_conserver]* **9** => Les ss champs qu'il ne faut pas écraser
- *[plugin_definition_champ]* **10** : Contient les infos (type, icones, événements) nécessaires pour générer les ss-champs d'un champ. Utilisé par le plugin ci-dessous'
- *[plugin_get_infos_ss_champ]* **11** : Permet de récupérer les infos (type, icones, événements...) des nouveaux ss champs à insérer. Il utilise le plugin ci-dessus qui contient les infos de tous les ss-champs d'un champ

plugin_catalogue_catalogage_grilles_actions_grilles_validation_lien_explicite()

validation_ss_champ_synthetique.php

- **Package** default

array function
plugin_catalogue_catalogage_grilles_actions_grilles_validation_ss_champ_synthetique($parametres, 1, 2, 3, 4, 5)
[line 18]

Function Parameters:

- *mixed* **$parametres**
- *[infos]* **1** => [type_element], [nom_champ], [nom_ss_champ], [idx_onglet], [idx_champ], [idx_ss_champ]
- *[ID_element]* **2**
- *[ID_operation]* **3**
- *[action]* **4**
- *[nom_ss_champ_lien]* **5** => nom ($a, $b...) du sous champ de lien à modifier

plugin_catalogue_catalogage_grilles_actions_grilles_validation_ss_champ_synthetique ()

- **Package** default

void function plugin_catalogue_catalogage_grilles_add_ID_grille($parametres) *[line 9]*
Function Parameters:

- *mixed* **$parametres**

catalogue_catalogage_grilles_add_ID_grille()

add_ID_grille_modification.php

- **Package** default

array function plugin_catalogue_catalogage_grilles_add_ID_grille_modification($parametres, 1, 2, 3) *[line 12]*

Function Parameters:

- *array* **$parametres**
- *[plugin_definition_grille]* **1** : le plugin qui va nous donner la définition de la grille (quels champs vont dans quels onglets, le type des champs, les icones, les événements...)
- *[type_objet]* **2** : type de l'objet
- *[ID_notice]* **3** : ID de la notice à modifier

plugin_catalogue_catalogage_grilles_add_ID_grille_modification()

- **Package** default

void function plugin_catalogue_catalogage_grilles_enregistrer_notice($parametres) *[line 15]*
Function Parameters:

- **$parametres**

plugin_catalogue_catalogage_grilles_enregistrer_notice()
[ID_operation] => ID de l'opération

[plugin_marcxml] => le plugin à utiliser pour convertir les données de la grille en marcxml
[plugin_notice_2_db] => le plugin qui crée ou maj la notice dans la DB. Récupère accès..., gère liens implicites

Ce plugin valide une notice saisie dans une grille. Il appelle une série de plugins pour convertir la grille en marcxml puis en extraire les données (accès, tris, liens explicites), mettre à jour les liens implicites et enregistrer le résultat dans la DB

get_infos_ss_champ.php

- **Package** default

array function plugin_catalogue_catalogage_grilles_get_infos_ss_champ($parametres, 1, 2) *[line 14]*

Function Parameters:

- *array* **$parametres**
- *[plugin_definition_champ]* **1** => le plugin qui va retourner les paramètres sur le champ
- *[nom_ss_champ]* **2** => le code du ss champ (a, b, c...)

plugin_catalogue_catalogage_grilles_get_infos_ss_champ()

Ce plugin retourne des informations sur un sous-champ, à savoir quel type (textbox, textarea...)... Il utilise un autre plugin de définition du champ (et qui contient la liste des sous-champs)

grille_marc_2_marcxml.php

- **Package** default

array function plugin_catalogue_catalogage_grilles_grille_marc_2_marcxml($parametres, 1) *[line 11]*

Function Parameters:

- *mixed* **$parametres**
- *[ID_operation]* **1**

plugin_catalogue_catalogage_grilles_grille_marc_2_marcxml()

- **Package** default

void function plugin_catalogue_catalogage_grilles_switcher($parametres) *[line 3]*

Function Parameters:

- **$parametres**

get_notice.php

- **Package** default

array function plugin_catalogue_import_export_get_notice($parametres, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11) *[line 29]*

Function Parameters:

- *mixed* **$parametres**
- *["taille_fichier"]* **1** => taille du fichier (si fichier)
- *["handle"]* **2** => handle du fichier (si fichier)
- *["chaine"]* **3** => chaine de caractères (si chaine)
- *["taille_chaine"]* **4** => longueur de la chaine (si chaine)
- *["last_car"]* **5** => Position du dernier caractère lu la fois précédente
- *["plugin_split"]* **6** Plugin utilisé pour retourner la notice suivante (registre)
- *["plugin_convert_xml"]* **7** Plugin utilisé pour convertir en XML (registre)
- *["plugin_get_infos_ddbl"]* **8** Plugin utilisé pour récupérer les infos de dédoublonnage (registre)
- *["plugin_ddbl"]* **9** Plugin utilisé pour dédoublonner (registre)
- *["plugin_affiche_notice"]* **10** Plugin utilisé pour afficher les 2 notices (registre)
- *["type"]* **11** Type d'objet (biblio...) (registre)

plugin_catalogue_import_export_get_notice()

get_notice_suivante_separateur.php

- **Package** default

["resultat"]["notice"]=>notice function plugin_catalogue_import_export_meta_format_get_notice_suivante_separateur($parametres, 1, 2, 3, 4, 5, 6, 7) *[line 23]*

Function Parameters:

- *mixed* **$parametres**
- *["separateur"]* **1** => séparateur
- *["hex_separateur"]* **2** => séparateur fourni sous forme hexadécimale (pb des caractères spéciaux dans le registre)
- *["taille_fichier"]* **3** => taille du fichier (si fichier)
- *["handle"]* **4** => handle du fichier (si fichier)
- *["chaine"]* **5** => chaine de caractères (si chaine)
- *["taille_chaine"]* **6** => longueur de la chaine (si chaine)
- *["last_car"]* **7** => dernier caractère

plugin_catalogue_import_export_meta_format_get_notice_suivante_separateur()

- **Package** default

array function plugin_catalogue_import_export_meta_format_marc2xml($parametres, 1, 2) *[line 16]*

Function Parameters:

- *mixed* **$parametres**
- *["notice"]* **1** => notice marc
- *["plugin_encodage"]* **2** => plugin utilisé pour décoder les caractères

plugin_catalogue_import_export_meta_format_marc2xml()

Ce plugin transforme une notice marc en marcxml

array function plugin_catalogue_import_export_meta_format_marc2xml__retourne_ss_champs($idx_champ, $zone_adresse, $zone_champs) *[line 102]*

Function Parameters:

- *mixed* **$idx_champ**
- *mixed* **$zone_adresse**
- *mixed* **$zone_champs**

plugin_catalogue_import_export_meta_format_marc2xml__retourne_ss_champs()

retourne les sous-champs contenus dans un champ marc (sous forme d'Array)

split_fichier.php

- **Package** default

array function plugin_catalogue_import_export_meta_format_split_fichier($parametres, 1, 2, 3, 4, 5, 6) *[line 20]*

Function Parameters:

- *mixed* **$parametres**
- *["meta_format"]* **1** => méta-format
- *["taille_fichier"]* **2** => taille du fichier (si fichier)
- *["handle"]* **3** => handle du fichier (si fichier)
- *["chaine"]* **4** => chaine de caractères (si chaine)
- *["taille_chaine"]* **5** => longueur de la chaine (si chaine)
- *["lien_format_plugin"]* **6** (ARRAY) => tableau associatif méta-format / PA pour le traier (paramètre stoché dans le registre)

catalogue_import_export_meta_format_split_fichier()

- **Package** default

void function plugin_catalogue_marcxml_crea_marcxml($parametres) *[line 3]*

Function Parameters:

- **$parametres**

crea_notice.php

- **Package** default

void function plugin_catalogue_marcxml_db_crea_notice($parametres, 1, 2, 3, 4, 5, 6) *[line 18]*

Function Parameters:

- *mixed* **$parametres**
- *["contenu"]* **1** => le contenu
- *["acces"]* **2** => les acces sous la forme ["nom_acces"] => "valeur acces"
- *["tri"]* **3** => les tris (idem)
- *["type"]* **4** => Type d'objet (biblio...)'
- *["ID"]* **5** *option* => ID de la notice (si maj)
- *["liens"]* **6** => les liens sous la forme ["ID_lien"] => ID de la notice liée, ["type_objet"] => type d'objet lié, ["type_lien"] => type de lien

  Ce plugin enregistre dans la DB les données fournies en paramètre (sous forme d'array)

plugin_catalogue_marcxml_db_crea_notice()

get_lien_explicite.php

- **Package** default

array function plugin_catalogue_marcxml_db_get_lien_explicite($parametres, 1, 2, 3, 4, 5) *[line 25]*

Function Parameters:

- *mixed* **$parametres**
- *type* **1** => type de la notice (biblio, auteur...)
- *ID* **2** => ID de la notice
- *OU* **3** notice => la notice elle même (objet DOMXml))
- *plugin_formate* **4** => le plugin qui va récupérer et mettre en forme les bons champs dans la notice
- *OU* **5** [nouveau_champ_str] => on peut passer le nouveau champ directement (a déjà été formaté par la fonction appelante)

DEPLACE EN /MARCXML/DB/

plugin_catalogue_marcxml_db_get_lien_explicite()

Ce plugin génère un champ de lien explicite à partir des références d'une notice (type et ID) OU à partir de la notice directement Le retour est sous forme de tableau (qui pourra ensuite être converti en XML ou utilisé en JS) Le plugin utilise un autre plugin pour récupérer les infos et les formater PAR CONTRE, il ne gère pas les sous-champs à conserver. ça doit être fait par la fonction appelante

get_liste_liens_explicites.php

- **Package** default

array function plugin_catalogue_marcxml_db_get_liste_liens_explicites($parametres, 1, 2, 3) *[line 21]*

Function Parameters:

- *array* **$parametres**
- *[notice]* **1** => la notice XML
- *[champs_liens_explicites]* **2** => liste des champs susceptibles de contenir un lien avec les infos nécessaires pour récupérer le lien
- *[* **3** """""][700,464,...][type | type_lien | ss_champ_jointure]

plugin_catalogue_marcxml_db_get_liste_liens_explicites()

Ce plugin retourne la liste des notices liées à la notice fournie en paramètre Pour chaque notice il indique le type d'objet, le type de lien et le n° de notice Il utilise en paramètre un tableau contenant la liste des champs susceptibles de contenir un lien (+ d'autres infos) De telle sorte qu'en jouant sur ce paramètre, on peut ne récupérer les notices liées à tels ou tels champs

maj_liens_implicites.php

- **Package** default

array function plugin_catalogue_marcxml_db_maj_liens_implicites($parametres, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13) *[line 29]*

Function Parameters:

- *array* **$parametres**
- *ID* **1** : ID de la notice
- *type* **2** : Type de l'objet
- *[plugin_get_champ_lie]* **3** : plugin utilisé pour récupérer un champ de lien précis (à partir du n° de notice liée)
- *[plugin_maj_lien_explicite]* **4** : plugin utilisé pour mettre à jour un champ donné d'une notice XML
- *[notice_old]* **5** *option* : ancienne notice avant modification (si non fourni, trouvé avec ID)
- *[notice_new]* **6** : nouvelle notice (ne peut être fourni sous forme d'ID)
- *[champs_liens_implicites]* **7** liste des types de liens implicites susceptibles d'être générés à partir de cette notice !! utilise les paramètres des liens EXPLICITES
- *[type_lien]* **8** => nom du champ qui sera généré dans la notice liée
- *[type_origine]* **9** => type d'objet lié (biblio, auteur...)
- *[plugin_formate]* **10** => le plugin qui va récupérer et mettre en forme les infos dans la notice pour générer le champ de lien
- *[plugin_notice_2_db]* **11** => Le plugin qui permet de regénérer une notice quand un champ de lien y a été modifié
- *[ss_champ_jointure]* **12** => sous-champ contenant le n° de notice liée (souvent $3)
- *[ss_champs_a_conserver]* **13** => ss-champs à ne pas écraser

plugin_catalogue_marcxml_db_maj_liens_implicites()

Ce plugin met à jour les liens implicites d'une notice donnée (fournie par ID ou directement en XML) Il y a forcément une notice déjà présente dans la base et une nouvelle notice qui arrive On commence par lister les types de liens implicites qui peuvent exister pour cette notice Pour chacun, on regarde si la nouvelle notice va apporter une modification par rapport à l'ancienne Si c'est le cas, on récupère toutes les notices liées, et pour chacune, on modife le champ de lien.

maj_lien_explicite.php

- **Package** default

array function plugin_catalogue_marcxml_db_maj_lien_explicite($parametres, 1, 2, 3, 4, 5) *[line 18]*

Function Parameters:

- *array* **$parametres**
- *[notice]* **1** => la notice à modifier
- *[champ]* **2** => le champ à remplacer (DomNode)
- *[champ_remplace]* **3** => le nouveau champ (chaine de la forme a:xxx|b:yyy|b:zzz) retourné par un get_datafield
- *[ss_champs_a_conserver]* **4** => les ss_champs à ne pas écraser dans le champ
- *[nom_champ]* **5** => le nom du champ (700, 464...)

plugin_catalogue_marcxml_db_maj_lien_explicite()

Ce plugin met à jour un champ de lien donné dans une notice. Il faut fournir la notice en XML et le champ de lien à remplacer sous forme de DomNode (généralement fourni par le plugin get_datafiels_node) On fournit également la liste des ss-champs à ne pas écraser ATTENTION notice et champ doivent appartenir au même DOM

notice_2_db.php

- **Package** default

void function plugin_catalogue_marcxml_db_notice_2_db($parametres, 1, 2, 3, 4, 5, 6) *[line 19]*

Function Parameters:

- *array* **$parametres**
- *[plugin_notice_2_infos]* **1** => le plugin pour récupérer accès, tris, liens...
- *[plugin_maj_liens_implicites]* **2** => pour maj les liens implicites
- *[plugin_crea_notice]* **3** => opérations SQL
- *[notice]* **4** => notice DOMXml
- *[ID_notice]* **5** ** option ** => si maj de notice
- *[type_objet]* **6** => type de notice (biblio, auteur...)

plugin_catalogue_marcxml_db_notice_2_db()

Ce plugin crée ou met à jour une notice dans la base de données La notice doit être fournie en XML Il faut également fournir les plusgins nécessaires pour gérer les accès, les liens et faire les opérations SQL à proprement parler

notice_2_infos.php

- **Package** default

array function plugin_catalogue_marcxml_db_notice_2_infos($parametres, 1, 2, 3, 4, 5, 6) *[line 20]*

Function Parameters:

- *array* **$parametres**
- *[plugin_acces]* **1** => plugin utilisé pour récupérer les accès
- *[plugin_tri]* **2** => pour récupérer les tris
- *[plugin_liens_explicites]* **3** => pour récupérer les liens
- *[type]* **4** => type d'objet (biblio, auteur...)
- *SOIT* **5** [notice] => la notice en XML
- *SOIT* **6** [ID_notice] => ID_notice (pour récupérer la notice)

plugin_catalogue_marcxml_db_notice_2_infos()

Ce plugin extrait les infos d'une notice afin de l'enregistrer dans la db (plugin crea_db) Il va utiliser plusieurs plugins pour récupérer chaque type d'information (acces, tri, liens)

On peut fournir directement la notice OU un ID, dans ce cas la notice est récupérée dans la base

encodage.php

- **Package** default

void function plugin_catalogue_marcxml_encodage($parametres, 1, 2, 3, 4, 5, 6) *[line 19]*

Function Parameters:

- *mixed* **$parametres**
- *[chaine]* **1** => la chaine à modifier
- *[sens]* **2** => par défaut de a vers b sauf si vaut "ba" (de b vers a)
- *[utf8_decode]* **3** => si 1, la chaine sera utf8_decodée avant d'être traitée
- *[utf8_encode]* **4** => si 1, la chaine sera utf8_encodée après avoir été traitée
- *[caracteres]* **5** => liste des caracteres à substituer
- *[a]* **6** et [b] caracteres à substituer

plugin_catalogue_marcxml_encodage()

Ce plugin encode ou décode une chaine de caractères en fonction d'une liste de caracteres de substitution fournis. L'encodage peut se faire dans un sens ou dans l'autre (paramètre "sens") Les caractères sont fournis sous la forme des codes ASCII éventuellement séparés par des espaces

formate_array.php

- **Package** default

array function plugin_catalogue_marcxml_formate_array($parametres, 1, 2, 3, 4, 5, 6) *[line 19]*

Function Parameters:

- *mixed* **$parametres**
- *[avant]* **1**
- *[apres]* **2**
- *[avant_element]* **3**
- *[avant_element_verif]* **4**
- *[apres_element]* **5**
- *[tableau]* **6** => l'array à formater

plugin_catalogue_marcxml_formate_array()

Formate une array. retourne une chaine de caractères

- **Package** default

void function plugin_catalogue_marcxml_formate_mv($parametres) *[line 3]*
Function Parameters:

- **$parametres**

void function plugin_catalogue_marcxml_formate_mv_get_segment($tableau, $mot) *[line 46]*
Function Parameters:

- **$tableau**
- **$mot**

void function xxx_accent($mot) *[line 64]*
Function Parameters:

- **$mot**

formate_plugins.php

- **Package** default

[texte] function plugin_catalogue_marcxml_formate_plugins($parametres) *[line 28]*
Function Parameters:

- *mixed* **$parametres**

plugin_catalogue_marcxml_formate_plugins()
Ce plugin permet de formater une liste de plugins

Il appelle successicement les plugins comme ceci : [notice]<=([notice])

ATTENTION : à la base ce plugin fonctionnait avec les plugin "get_datafields_xxx"
qui attendaient le paramètre en [notice] et retournaient le résultat en [notice] Mais il peut aussi fonctionner avec d'autres plugins qui ont une autre signature. Dans ce cas, il faut utiliser des alias

formate_plugins_array.php

- **Package** default

array function plugin_catalogue_marcxml_formate_plugins_array($parametres, 1, 2, 3, 4, 5) *[line 22]*

Function Parameters:

- *mixed* **$parametres**
- *[notice]* **1**
- *[plugins]* **2**
- *[toto]* **3** => [nom_plugin]
- *[parametres]* **4**
- *[force_retour]* **5** : si 1, on retourne qqchse, même si 1 ou plusieurs plugins ont retourné des erreurs. Sinon, on propage l'erreur

plugin_catalogue_marcxml_formate_plugins_array()

Ce plugin retourne un tableau associatif, chaque élément du tableau étant généré par un autre plugin Il appelle successicement les plugins comme ceci : [notice]<=([notice])

ATTENTION : à la base ce plugin fonctionnait avec les plugin "get_datafields_xxx"
qui attendaient le paramètre en [notice] et retournaient le résultat en [notice] Mais il peut aussi fonctionner avec d'autres plugins qui ont une autre signature. Dans ce cas, il faut utiliser des alias

get_colonnes.php

- **Package** default

array function plugin_catalogue_marcxml_get_colonnes($parametres, 1, 2) *[line 16]*

Function Parameters:

- *array* **$parametres**
- *[tableau]* **1** => le tableau contenant les colonnes à formater
- *[colonnes][0,1,2...][nom_colonne* **2** | avant | apres | avant_verif] => infos contenant le formatage

plugin_catalogue_marcxml_get_colonnes()

Ce plugin permet de mettre en forme des informations fournies sous forme de tableau. Il est typiquement utilisé pour mettre en forme des infos extraites d'une base de données (ligne de résultat) en particulier les tables obj_xxx_acces

get_colonnes_array.php

- **Package** default

array function plugin_catalogue_marcxml_get_colonnes_array($parametres, 1, 2) *[line 16]*

Function Parameters:

- *array* **$parametres**
- *[tableau]* **1** => le tableau contenant les colonnes à formater
- *[colonnes][0,1,2...][nom_colonne* **2** | nom_champ | avant | apres] => infos contenant le formatage

plugin_catalogue_marcxml_get_colonnes_array()

Comme get_colonnes, mais retourne le résultat sous forme d'array. Fait correspondre la colonne [nom_colonne] au champ [nom_champ] du tableau

get_datafields.php

- **Package** default

$retour["resultat"]["texte"] function plugin_catalogue_marcxml_get_datafields($parametres, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11) *[line 29]*

Function Parameters:

- *array* **$parametres**
- *["notice"]* **1** => notice en marcxml. (inutile si objet tvs_marcxml fourni)
- *["tvs_marcxml"]* **2** => objet tvs_marcxml. Si non fourni, généré à partir de la notice (DomXml)
- *["champs"]* **3** => liste des champs à extraire
- *["champs"][XXX]["tag"]*=> **4** 1 des champs
- *["champs"][XXX]["idx"]*=> **5** position du champ dans la liste ou last()
- *["champs"][XXX][avant|avant_verif|apres]* **6** => chaines de caractères à placer avant, avant (si déjà qqchse avant) ou après le contenu du champ
- *["champs"][XXX]["sous-champs"]*=> **7** sous-champs à extraire pour ce champ
- *["champs"][XXX]["sous-champs"][YYY]["code"]*=> **8** 1 des sous-champs
- *["champs"][XXX]["sous-champs"][YYY]["idx"]*=> **9** position du ss-champ dans la liste ou last()
- *["champs"][XXX]["sous-champs"][YYY]["valeur"]*=> **10** Valeur requise pour un sous-champ
- *["champs"][XXX]["sous-champs"][YYY][avant|avant_verif|apres]* **11** => chaines de caractères à placer avant, avant (si déjà qqchse avant) ou après le contenu du ss-champ

plugin_catalogue_marcxml_get_datafields()

Ce plugin retourne des champs et sous-champs formatés sous forme de string. Le formatage (champs/sous champs à récupérer et la ponctuation, séparateurs...) est indiqué dans l'attribut [champs] La notice peut être fournie soit sous forme d'objet DomXml soit directement tvs_marcxml

get_datafields_array.php

- **Package** default

$retour["resultat"]["texte"] function plugin_catalogue_marcxml_get_datafields_array($parametres, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) *[line 33]*

Function Parameters:

- *array* **$parametres**
- *["notice"]* **1** => notice en marcxml
- *["tvs_marcxml"]* **2** => objet tvs_marcxml. Si non fourni, généré à partir de la notice (DomXml)
- *["champs"]* **3** => liste des champs à extraire
- *["champs"][XXX]["tag"]*=> **4** 1 des champs
- *["champs"][XXX]["idx"]*=> **5** position si plusieurs champs identiques : à partir de 1 ou "last()" pour le dernier
- *["champs"][XXX][avant|avant_verif|apres]* **6** => chaines de caractères à placer avant, avant (si déjà qqchse avant) ou après le contenu du champ
- *["champs"][XXX]["sous-champs"]*=> **7** sous-champs à extraire pour ce champ
- *["champs"][XXX]["sous-champs"][YYY]["code"]*=> **8** 1 des sous-champs
- *["champs"][XXX]["sous-champs"][YYY]["idx"]*=> **9** position du ss-champ dans la liste ou last()
- *["champs"][XXX]["sous-champs"][YYY]["valeur"]*=> **10** Valeur requise pour un sous-champ
- *["champs"][XXX]["sous-champs"][YYY][avant|avant_verif|apres]* **11** => chaines de caractères à placer avant, avant (si déjà qqchse avant) ou après le contenu du ss-champ
- *["champs"][XXX]["sous-champs"][YYY]["plugin_formate"]*=> **12** Un plugin pour formater le ss-champ. Le texte sera envoyé dans l'attribut [texte] et récupéré également dans [texte]

plugin_catalogue_marcxml_get_datafields_array()

Ce plugin retourne une liste de champs (datafields) comme get_data_fields, mais il le fait sous forme de tableau : exemple : [600] [0] => "toto : tutu : titi" [1] => "popo : pupu : pipi" [606] [0] => ...

get_datafields_nodelist.php

- **Package** default

$retour function plugin_catalogue_marcxml_get_datafields_nodelist($parametres, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11) *[line 34]*

Function Parameters:

- *array* **$parametres**
- *["notice"]* **1** => notice en marcxml. (inutile si objet tvs_marcxml fourni)
- *["tvs_marcxml"]* **2** => objet tvs_marcxml. Si non fourni, généré à partir de la notice (DomXml)
- *["champs"]* **3** => liste des champs à extraire
- *["champs"][XXX]["tag"]=>* **4** 1 des champs
- *["champs"][XXX]["idx"]=>* **5** position si plusieurs champs identiques : à partir de 1 ou "last()" pour le dernier
- *["champs"][XXX]["plugin_formate"]=>* **6** Un plugin pour formater le champ. Le texte sera envoyé dans l'attribut [texte] et récupéré également dans [texte]
- *["champs"][XXX]["sous-champs"]=>* **7** sous-champs devant être présents pour extraire ce champ
- *["champs"][XXX]["sous-champs"][YYY]["code"]=>* **8** 1 des sous-champs
- *["champs"][XXX]["sous-champs"][YYY]["valeur"]=>* **9** valeur que doit prendre le sous-champ. Si vide, on accède n'importe quelle valeur du moement que le ss-champ est présent
- *["champs"][XXX]["sous-champs"][YYY]["idx"]=>* **10** position si plusieurs ss-champs identiques : à partir de 1 ou "last()" pour le dernier
- *["champs"][XXX]["sous-champs"][YYY]["plugin_formate"]=>* **11** Un plugin pour formater le ss-champ. Le texte sera envoyé dans l'attribut [texte] et récupéré également dans [texte]

plugin_catalogue_marcxml_get_datafields_nodelist()

Ce plugin retourne un tableau (!! PAS un nodelist) de tous les champs (DomNode) d'une notice présentant certains critères : le nom du champ (200, 700, 464...) la présence de certains sous-champs ($a, $b...) La valeur des sous-champs la position du champ ou du ss-champ dans la liste NOTE : c'est une disjonction. Il suffit que la condition soit remplie pour UN sous sous-champ, et le champ sera validé // p-ê à modifier plus tard La notice peut être fournie soit sous forme d'objet DomXml soit directement tvs_marcxml

modif_marcxml.php

- **Package** default

[notice] function plugin_catalogue_marcxml_modif_marcxml($parametres, 1, 2, 3, 4, 5, 6) *[line 29]*

Function Parameters:

- *mixed* **$parametres**
- *[notice]* **1** => la notice DomXml
- *[modifications][0,1,2...]* **2** => liste des modifications à apporter à la notice
- *----[recherche]* **3** => équation de recherche sur les champs et les ss-champs (cf plus bas)
- *----[type_modif_champ]* **4** => type de modification sur le champ : add => ajout d'un champ | delete => suppression | reset => on remplace complètement le contenu du champ | update on modifie certains ss-champs
- *----[def_champ]* **5** (attention définition différente pour (add, reset) que pour update) => définition du champ à créer (cf plus bas)
- *----[tag]* **6** (pour add) => le tag du champ à créer (add uniquement)

plugin_catalogue_marcxml_modif_marcxml()

- **Package** default

void function plugin_catalogue_marcxml_test_xml($parametres) *[line 3]*
Function Parameters:

- **$parametres**

formulaire_2_recherche.php

- **Package** default

void function plugin_catalogue_recherches_formulaire_2_recherche($parametres, 1) *[line 14]*

Function Parameters:

- *mixed* **$parametres**
- *[array_in]* **1** => le tableau contenant les criteres de recherche

plugin_catalogue_recherches_formulaire_2_recherche()

Ce plugin convertit un tableau de paramètres de recherche retourné par un formulaire en tableau de paramètres pour les fonctions de recherche. Selon les cas, la transformation peut être faible ou plus importante

recherche_mv.php

- **Package** default

array function plugin_catalogue_recherches_recherche_mv($parametres, 1, 2, 3, 4, 5) *[line 20]*

Function Parameters:

- *mixed* **$parametres**
- *[query]* **1** => chaine à rechercher
- *[plugin_recherche]* **2** => le plugin utilisé pour faire la recherche
- *[plugin_formate]* **3** => Plugin pour formater chaque ligne (mettre en valeur les infos pertinentes)
- *[nb_resultats]* **4** => Nombre de notices à retourner
- *[nb_max]* **5** => nombre maximum de pages à parcourir avant d'abandonner si on n'a pas nb_resultats (cas où on aurait beaucoup de doublons)

plugin_catalogue_recherches_recherche_mv()

Ce plugin est utilisé pour effectuer une recherche dans un champ multivaleurs (par exemple le champ des accès auteurs d'une notice biblio) Il utilise un plugin pour la recherche (qui retournera juste UN champ par notice) Puis un plugin pour mettre en forme ces champs, c'est à dire récupérer les mots pertinents et les remettre dans leur contexte Il se peut que le plugin trouve des doublons (le même auteur dans plusieurs notices). Dans ce cas, il relance la recherche pour les notices suivantes Sauf si [nb_max] est dépassé (pour éviter qu'on parcourts des centaines de notices ce qui ralentirait)

recherche_simple.php

- **Package** default

include_once **$GLOBALS["tvs_global"]["conf"]["ini"]["include_path"]."classes/recherche_simple.php"** *[line 4]*

array function plugin_catalogue_recherches_recherche_simple($parametres, 1) *[line 21]*

Function Parameters:

- *mixed* **$parametres**
- *[param_recherche]* **1** => les paramètres attendus par recherche_simple::init() (cf la classe recherche_simple)

plugin_catalogue_recherches_recherche_simple()

Interface avec la classe recherche_simple Tous les paramètres de la méthode init sont fournis dans le paramètre [param_recherche]

- **Package** default

array function plugin_div_plugins_2_array($parametres) *[line 22]*
Function Parameters:

- *mixed* **$parametres**

plugin_div_plugins_2_array()

plugins_2_texte.php

- **Package** default

array function plugin_div_plugins_2_texte($parametres, 1) *[line 16]*
Function Parameters:

- *mixed* **$parametres**
- *[texte]* **1** => le texte

plugin_div_plugins_2_texte()

- **Package** default

void function plugin_div_test($parametres) *[line 3]*
Function Parameters:

- **$parametres**

eval_bureau.php

- **Package** default

[bureau] function plugin_transactions_bureau_eval_bureau($parametres, 1, 2, 3, 4, 5, 6) *[line 32]*

Function Parameters:

- *mixed* **$parametres**
- *[bureau]* **1** => le bureau
- *[plugin_evaluation]* **2** => le plugin utilisé pour évaluer certaines propriétés du bureau.
- *En* **3** fonction des valeurs trouvées, on appliquera telle ou telle action définies dans le switch
- *[switch][cond1,* **4** cond2, cond3..., else] => les différents retours possibles de [plugin_evaluation]. On peut aussi utiliser la clef "else"
- ---------- **5** [break] => >0 il faudra arrêter les traitements à l'issue de celui-ci'
- ---------- **6** [plugins][0,1,2...] => les plugins à appliquer si cette condition est remplie. Chaque plugin est susceptible de modifier le bureau (mais pas forcément)

plugin_transactions_bureau_eval_bureau()

Ce plugin utilise le plugin [plugin_evaluation] pour évaluer certaines variables du bureau par ex. bureau/nb_docs > bureau/nb_docs_max En fonction de ce qui est retourné (généralement 0 ou 1), on utilise une close [switch] pour appliquer certaines actions Pour chaque cas on pourra déterminer : > une liste de plugins à effectuer (chaque plugin étant susceptible de modifier le bureau) > une close break

Signature du plugin évaluation : ([eval])=plugin_evaluation([type_plugin], [p1], [p2], [liste_param(option)])

Signatue des plugins dans le switch : ([bureau])=plugin([bureau]) => !!! le retour de [bureau] n'est pas obligatoire. S'il n'est pas retourné, le bureau n'est pas modifié.

eval_conditions.php

- **Package** default

[eval] function plugin_transactions_bureau_eval_conditions($parametres, 1, 2, 3) *[line 20]*
Function Parameters:

- *[p1]* **1** => paramètre 1
- *[p2]* **2** => paramètre 2
- *[liste_param]* **3** => liste de paramètres : si non fourni, on utilise p1 et p2 pour générer cette liste (utilisé quand on veut évaluer + de 2 paramètres par exemple pour and, or...)
- *[type_eval]* **$parametres** => type d'évaluation : egal|sup|sup_egal|inf|inf_egal|and|or

plugin_transactions_bureau_eval_conditions()

Ce plugin évalue les paramètres fournis (comparateurs mathématiques, logiques...) [type_eval] indique le type d'évaluation à effectuer

Il retourne [eval] qui vaut généralement 0 ou 1

On peut utiliser un équivalent de parenthèses en incluant des plugins à la place d'un des paramètres avec !!

extract_2_bureau.php

- **Package** default

[bureau] function plugin_transactions_bureau_extract_2_bureau($parametres, 1, 2, 3, 4, 5, 6, 7) *[line 32]*

Function Parameters:

- *mixed* **$parametres**
- *[bureau]* **1** => le bureau à enrichir
- *[plugin_extraction]* **2** => (optionnel) : s'il faut enrichir le bureau avec des variables extraites d'un plugin
- *[extractions][0,1,2...]* **3**
- -------*[destination]* **4** => emplacement dans le bureau où copier la valeur
- -------*[type_data]* **5** => type de valeur à copier. array|json (cf explication plus haut)
- -------*[valeur]* **6** => valeur à copier ou chaine json
- -------*[origine]* **7** => emplacement de la valeur à copier (si extraite du plugin ou du bureau)

plugin_transactions_bureau_extract_2_bureau()

Ce plugin enrichit une variable passée en paramètre (une array appelée $bureau) avec des variables, soit constantes (string, array, json) soit extraites d'un plugin soit extraites d'un autre emplacement du bureau (on copie une vaiable du bureau vers lui-même mais ailleurs)

Si [plugin_extraction] est défini, on récupérera les données d'un plugin (sinon, on copie des valeurs constantes)

Pour chaque extraction, on doit spécifier [destination] qui est l'endroit du BUREAU où copier les données On peut copier 5 types de données : > Une constante : [valeur]=xxx > Une array (vide) : [type_data]=array > Une array complexe via json : [type_data]=json et [valeur]="la chaine json à évaluer" > Un élément retourné par le plugin : [origine] = data_plugin/xx/yy/zz (data_plugin correspond à $tmp[resultat]) > Un élément déjà présent sur le bureau : [origine] = bureau/xx/yy/zz

traitements_bureau.php

- **Package** default

[bureau] function plugin_transactions_bureau_traitements_bureau($parametres, 1, 2) *[line 22]*
Function Parameters:

- *[traitements][0,1,2...]* **1** => liste des traitements à effectuer sur le bureau
- ------ **2** [nom_plugin | parametres...]
- *[bureau]* **$parametres** => le bureau

plugin_transactions_bureau_traitements_bureau()

Ce plugin va appliquer une série de plugins sur le bureau Ces plugins vont généralement modifier le bureau (mais pas obligatoirement)

Certains plugins sont susceptibles de retourner une close [break], auquel cas les traitements sont interrompus Le plugin retournera alors lui-même une close [break-1], ce qui fait que plusieurs plugins traitements_bureau peuvent être inclus les uns dans les autres

Signature des plugins du traitement : ([bureau], [break])=plugin([bureau])

Appendices

Appendix A - Class Trees

Package default

Index

A

C

E

F

G

M

N

P

R

S

T

V

X