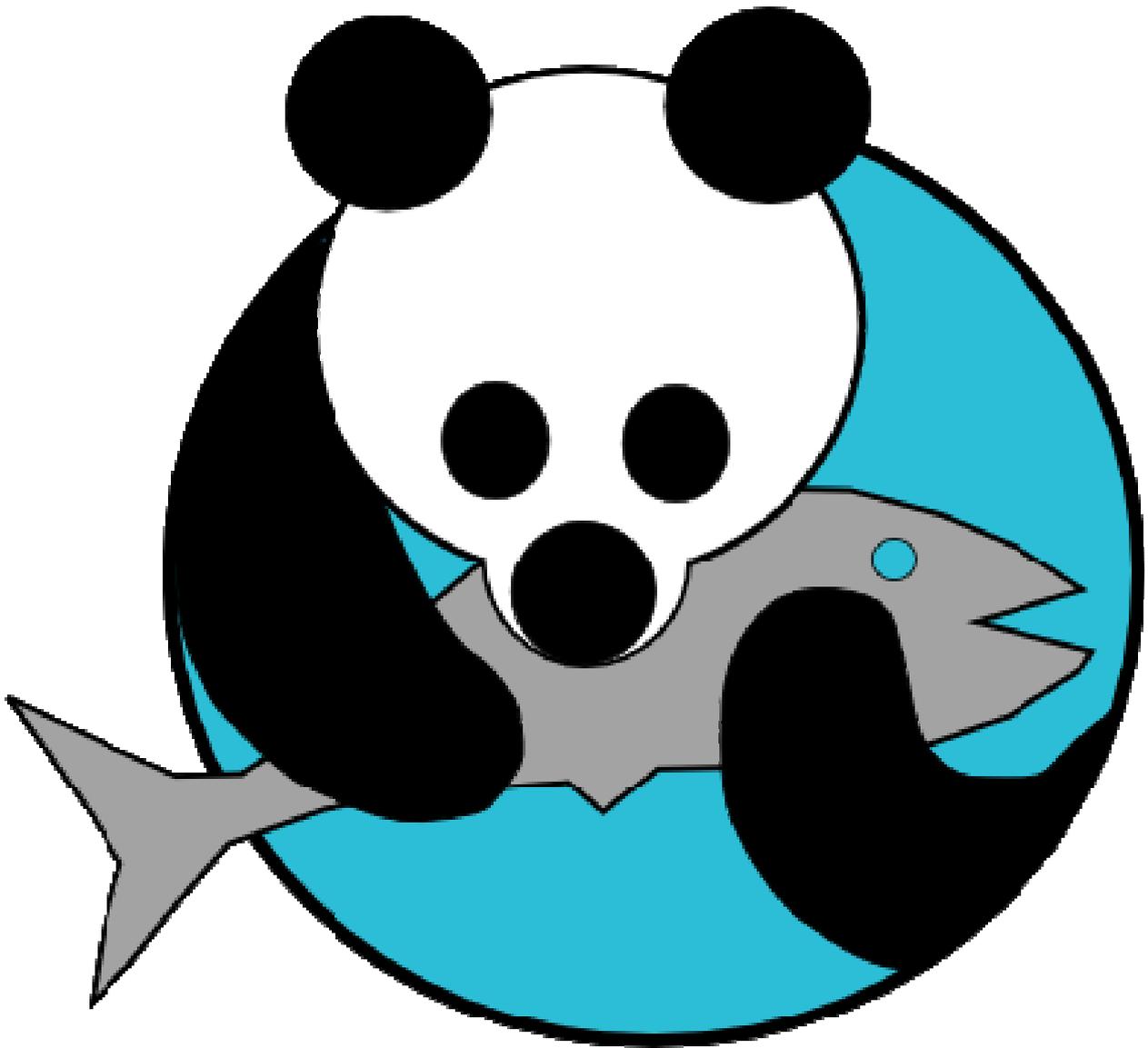


# Waterbear

## Guide de l'administrateur

Version 1.0 du 01/09/2015





## Table des matières

A.	Administration standard.....	5
	Interface d'administration simplifiée .....	6
	La gestion des listes.....	8
	La gestion des sauvegardes .....	10
	Transferts vers Bokeh .....	12
	Traitements de nuit .....	13
	Lettres de rappel et de réservation .....	14
	Listings des réservations à récupérer et remettre en rayon .....	15
	Traitements par lot.....	16
	Administration de Bokeh .....	18
B.	Administration avancée .....	19
	La gestion des objets .....	20
	Le registre – interface graphique .....	23
	Le registre – fonctionnement du framework .....	30
	La gestion des pages .....	31
	Les plugins .....	34
	Déclarer un plugin dans le registre .....	34
	Appeler un plugin.....	36
	##, !! et ?? : les codes secrets des plugins.....	38
	Aide graphique à la navigation entre plugins.....	42
	Alias et alias_retour .....	43
	Tirer partie de l'héritage dans les pages .....	46
	La personnalisation.....	48
	Bonnes pratiques de la personnalisation dans Waterbear.....	48
	La gestion des droits dans Waterbear.....	49
C.	Comment paramétrer vous-même.....	50
	Paramétrer les grilles de catalogage.....	51
	Généralités .....	51
	Les différents types de sous-champs .....	54
	Autres paramètres notables des sous-champs .....	54
	Les switchers.....	55
	Paramétrer les masques .....	57
	Paramétrer les accès et les tris.....	58
	Paramétrer les formulaires de recherche et les tris.....	61
	Paramétrer les barres d'icônes et de menus.....	64
	Paramétrer le wizard emplacements.....	66
	Le paramétrage des quotas du prêt .....	68
	Le paramétrage des quotas de réservations.....	70
	Paramétrage des abonnements.....	71
	Paramétrage des colonnes du prêt / retour / réservations .....	72
	Paramétrer l'affichage des notices.....	73
	Paramétrer les sections et les types docs.....	74
D.	ANNEXES.....	76
	Définitions des principaux plugins.....	77
	Plugin de formatage catalogue/marcxml/get_datafields .....	77
	Plugin de formatage catalogue/marcxml/formate_plugins.....	78
	Plugin de création d'objet marcxml catalogue/marcxml/crea_marcxml .....	79

Plugin de modification d'objet marcxml catalogue/marcxml/modif_marcxml .....	80
Plugin de recherche catalogue/recherches/recherche_simple .....	81
Plugin utilitaire pour les dates div/util_dates.....	85
Plugin utilitaire pour les chaines de caractères div/util_string.....	86
Paramétrage avancé des champs de recherche et champs statistiques .....	87
Comprendre les liens entre objets .....	94
Utiliser les champs d'auto-complétion.....	98

**A. Administration standard**

## **Interface d'administration simplifiée**

Les paramètres les plus courants peuvent être réalisés via l'interface d'administration simplifiée.

Depuis la page d'accueil, menu administration >> administration

Les paramètres sont regroupés par onglets. Pour chaque paramètre à modifier, un petit descriptif indique son utilité.

Pour modifier un paramètre, il suffit de saisir la nouvelle valeur puis de cliquer en dehors de la zone de saisie. Un message « OK » vous confirme alors que la modification a bien été prise en compte (pas besoin d'autres validations).

En fait, chaque paramètre de l'interface d'administration est un raccourci vers le registre (voir la rubrique consacrée au registre dans ce guide). L'interface simplifiée évite simplement d'avoir à gérer la complexité du registre qui comprend plus de 60.000 paramètres différents.

Il est cependant possible d'accéder au nœud (ou aux nœuds) du registre

correspondant à chaque paramètre en cliquant sur l'icône . Vous devez être connecté en administrateur pour y accéder.

Par ailleurs certains paramètres sont trop complexes pour être traités directement dans l'administration simplifiée. Mais on y trouve cependant un raccourci permettant d'accéder à la zone du registre correspondante



L'administration simplifiée est elle-même paramétrable dans le registre. Cela signifie qu'il vous est possible de rajouter des onglets ou des paramètres modifiables.

Vous pouvez également créer plusieurs administrations simplifiées différentes qui seraient accessibles à des utilisateurs différents.

Quota et durée des prêts | Abonnements | Transit | Quota et durée des réservations | Rappels | Géolocalisation | Divers | **Codes barres**

**Codes barres exemplaires fonds propre : longueur**   
Longueur des codes barres exemplaires du fonds propre  
6

**Codes barres exemplaires fonds propre : préfixe**   
Préfixe des codes barres exemplaires du fonds propre (laisser vide si pas de préfixe)  
a

**Codes barres exemplaires BDP : longueur**   
Longueur des codes barres exemplaires de la BDP  
10

**Codes barres exemplaires BDP : préfixe**   
Préfixe des codes barres exemplaires de la BDP (laisser vide si pas de préfixe)  
b

**Codes barres des lecteurs : longueur**   
Longueur des codes barres des lecteurs  
5

**Codes barres lecteurs : préfixe**   
Préfixe des codes barres des lecteurs (laisser vide si pas de préfixe)

**Paramétrage des codes barres**   
Paramétrage avancé (si plus de codes barres...). ATTENTION ce paramétrage s'effectue directement dans le registre.

Pour modifier un paramètre, saisissez le dans la zone de texte, puis cliquez en dehors. Vous obtiendrez un message de confirmation « OK ». Pas besoin d'autre validation.

Vous pouvez accéder à la zone du registre correspondant à ce paramétrage en cliquant sur l'icône

Certains paramètres contiennent uniquement un raccourci vers le registre

## La gestion des listes

Waterbear permet de gérer facilement les différentes listes utilisées dans le logiciel. Par exemple, la liste des bibliothèques, des emplacements, des CSP, des quartiers...

Pour cela, depuis la page d'accueil, menu administration >> listes du catalogue ou listes des lecteurs et des prêts.

Une fois dans l'interface, sélectionnez la liste à modifier via le menu déroulant en haut à gauche.

La liste correspondante s'affiche. Chaque élément de la liste est constitué d'un code et d'une valeur (intitulé). Par exemple pour un quartier le code pourrait être « CEN » et la valeur « Centre ville »

**ATTENTION** : vous pouvez modifier les intitulés sans que cela porte à conséquence. Par exemple, si un de vos quartiers a changé de nom, modifiez son intitulé : tous les lecteurs appartenant à ce quartier déjà enregistrés dans la base bénéficieront immédiatement du nouvel intitulé.

En revanche, si vous modifiez le code, sachez que les objets déjà enregistrés dans la base avec l'ancien code ne sont pas modifiés. Vous devrez faire un traitement par lot pour les modifier.

Pour modifier les codes ou intitulés existants, modifiez les simplement dans la zone de texte puis cliquez en dehors de la zone de texte. La modification est instantanée et ne nécessite pas de validation.

Pour supprimer une paire code/valeur, cliquez sur l'icône  en face de la ligne

Pour créer une nouvelle paire code/valeur, saisissez les dans la dernière ligne (vierge) et cliquez sur l'icône . La paire sera rajoutée à la liste.

Comme pour l'administration simplifiée, la gestion des listes est un raccourci vers le registre. Vous pouvez accéder à la zone du registre concernée en cliquant sur

l'icône .



Là encore, la gestion des listes est paramétrable dans le registre. Vous pourrez créer de nouvelles listes et les ajouter à l'interface de paramétrage.

Par ailleurs, les listes peuvent gérer plusieurs langues simultanément (on associera une valeur par langue à chaque code). Par défaut cependant, tout est en français.

Pour modifier les codes et valeurs existants, saisissez les directement dans les zones de textes, puis cliquez en dehors. Pas besoin d'une autre validation.

Waterbear peut gérer des listes multilingues, mais par défaut, tout est en français

1. Sélectionnez la liste à modifier

Liste : bibliothèques

Langue : français

code	valeur	action
-	-	✖
BEL	Bellevue	✖
BOU	Bourg	✖
HER	Hermeland	✖
SIL	Sillon	✖
		+

Pour ajouter une nouvelle paire, saisissez le code et la valeur dans la dernière ligne, puis cliquez sur l'icône « plus ».

Pour supprimer une paire code/valeur, cliquez sur la croix en face de la ligne

## La gestion des sauvegardes



Cette rubrique ne concerne que la version en ligne de Waterbear

Sur Waterbear.info, les sauvegardes sont automatisées. Elles ont lieu tous les soirs. Les sauvegardes sont conservées une semaine.

En cas de problème grave, vous avez donc une semaine pour restaurer une sauvegarde. Cela peut se faire très simplement via une interface en ligne.

Vous devez vous connecter en administrateur, puis depuis la page d'accueil : menu administration >> gestion des sauvegardes.

L'interface de gestion des sauvegardes est en fait externe à Waterbear. De fait, vous pouvez y accéder même si votre site est totalement planté et que vous n'arrivez pas du tout à ouvrir Waterbear.

Pour cela, allez sur <http://moccam-en-ligne.fr/metawb/sauvegarde.php>

Ici, vous devez saisir le nom de votre domaine ainsi que vos identifiants administrateur.

Dans l'interface qui s'affiche, cliquez simplement sur le jour de la sauvegarde à restaurer, puis cliquez sur « OK ». En complément vous pouvez éventuellement indiquer un commentaire sur la restauration.

**ATTENTION** : les sauvegardes se font entre minuit et 6h du matin. La sauvegarde du mercredi (par exemple) correspond donc en fait au mardi soir, et ainsi de suite.

Une fois la restauration effectuée (ça ne dure que quelques secondes), tous les utilisateurs de Waterbear doivent se déconnecter puis se reconnecter. Faute de faire cela, ils continueront à travailler sur l'ancienne base.

**Note importante** : Lorsque vous restaurez une sauvegarde, la base en cours n'est pas supprimée : il vous est possible de la restaurer à son tour en la sélectionnant dans l'historique des sauvegardes.

1. Sélectionnez le jour de la sauvegarde à restaurer

*Attention : les sauvegardes se font le matin entre minuit et 6h. Si vous sélectionnez la sauvegarde*

- Lundi :
- Mardi :
- Mercredi :
- Jeudi :
- Vendredi :
- Samedi :
- Dimanche :

### Restaurer une ancienne sauvegarde

*Chaque fois que vous restaurez une sauvegarde, Waterbear conserve une copie de la base de données. Aucune ancienne sauvegarde enregistrée*

### Indiquer le motif de la restauration

Motif :

OK

Vous avez accès à l'historique des restaurations de sauvegardes, et avez la possibilité de revenir en arrière après une restauration

2. Indiquez le motif de la restauration

3. Validez

## **Transferts vers Bokeh**



Cette rubrique ne concerne que la version en ligne de Waterbear

Bokeh est le portail en ligne accessible au public de la bibliothèque. Waterbear et Bokeh sont deux logiciels distincts.

Les documents catalogués dans Waterbear sont périodiquement exportés vers Bokeh. Il existe deux types d'exports :

- L'export incrémentiel a lieu plusieurs fois par jour et ne concerne que les créations et modifications de notices
- L'export total a lieu le dimanche et concerne la totalité de la base (remise à zéro). Les suppressions de notices ne sont transférées que via les exports totaux.

Tous ces processus sont automatisés et vous n'avez pas besoin de vous en préoccuper.

Cependant, il peut arriver que vous ayez besoin de lancer vous-mêmes les mises à jour (particulièrement quand vous faites des tests)

Pour cela, depuis la page d'accueil : menu administration >> transferts vers Bokeh.

Vous pouvez également accéder directement à l'interface en ligne :

<http://moccam-en-ligne.fr/metawb/transferts.php>

Indiquez votre domaine et vos identifiants administrateur.

Ensuite, sélectionnez l'export incrémentiel ou total et validez.

## Traitements de nuit

Waterbear effectue un certain nombre d'opérations de routine le soir, par exemple la suppression des réservations non récupérées ou bien encore la mise à jour des notices via moccam-en-ligne



Sur la version en ligne de Waterbear, ces traitements de nuit sont lancés automatiquement, vous n'avez pas à vous en préoccuper

Pour lancer manuellement les traitements de nuit, depuis la page d'accueil : menu traitements >> traitements quotidiens



Lorsque les traitements sont lancés, un « verrou » est placé sur la base pour éviter que plusieurs traitements soient lancés en même temps. En cas de problème, le verrou n'est pas enlevé, ce qui fera planter les traitements suivants.

Pour supprimer ce verrou, allez dans le registre : system/verrous/traitements et mettez la valeur « 0 » au lieu de « 1 »

Si vous constatez d'autres problèmes, il y a un log qui consigne le déroulement des traitements de nuit (afin de diagnostiquer). Pour le consulter, allez dans le registre (en administrateur) puis cliquez sur l'icône « gestion des logs » puis sur le log « GLOBAL »



Il est possible d'ajouter des traitements supplémentaires aux traitements de nuit (par exemple des traitements par lot)

## Lettres de rappel et de réservation

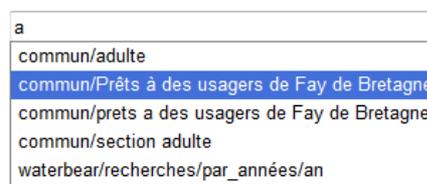
Pour éditer les lettres de rappel (pour avertir les lecteurs ayant des livres en retard) et les lettres de réservation (pour les avertir qu'un ouvrage réservé est disponible), cliquez sur les liens correspondant dans le menu démarrage.

Par défaut, Waterbear va éditer les lettres pour la totalité de la base, mais il est possible de restreindre cette édition à certains documents seulement. Par exemple, dans le cas d'un réseau, on peut souhaiter n'envoyer les lettres qu'aux lecteurs d'une certaine bibliothèque.

Pour cela, il faut créer des paniers dynamiques correspondant à la restriction recherchée. Pour les rappels, il faudra aller en recherche de prêts, rechercher les prêts des documents de la bibliothèque X puis créer un panier dynamique correspondant.

Ensuite, au moment d'éditer les lettres on commence à saisir les premières lettres du panier dynamique et Waterbear vous proposera les paniers existants.

**\*\*option\*\*** restreindre les prêts faisant l'objet d'un rappel à un panier donné



Le principe est le même pour les lettres de réservation, mais il faudra faire un panier dynamique de réservations.



Les lettres peuvent être envoyées par courrier, mail (si l'adresse mail est renseignée) ou les 2. Cela se paramètre dans l'administration simplifiée (onglets « quota et durée des réservations » et « rappels »).

On peut également paramétrer le délai entre les lettres de rappel ainsi que le texte de ces lettres dans l'onglet « rappels ».

Pour modifier le texte des lettres de réservation, il faut aller dans le registre :  
profiles/default/plugins/plugins/transactions/resas/messages\_resas/standard/p  
arametres

modifier les nœuds « bloc\_avant » et « bloc\_apres »

## **Listings des réservations à récupérer et remettre en rayon**

Si vous avez autorisé la réservation de documents disponibles, vous devrez régulièrement aller récupérer les documents réservés qui se trouvent en rayon.

De la même manière, vous pouvez souhaiter éditer un listing des réservations qui n'ont pas été récupérées par l'utilisateur et doivent donc être remises en rayon.

Pour cela, cliquez sur les liens correspondant sur la page d'accueil.

Il s'agit simplement d'un raccourci vers la recherche de réservations (avec des paramètres pré-remplis). Vous pourrez si vous le désirez modifier ces critères.

**N'oubliez pas, dans les deux cas, de passer les documents en retour avant de les mettre sur l'étagère des réservations ou de les remettre en rayon (cela actualise le statut). Faute de faire cela, les documents apparaîtront encore dans le listing la prochaine fois.**



Les lettres peuvent être envoyées par courrier, mail (si l'adresse mail est renseignée) ou les 2. Cela se paramètre dans l'administration simplifiée (onglets « quota et durée des réservations » et « rappels »).

On peut également paramétrer le délai entre les lettres de rappel ainsi que le texte de ces lettres dans l'onglet « rappels ».

Pour modifier le texte des lettres de réservation, il faut aller dans le registre :  
profiles/default/plugins/plugins/transactions/resas/messages\_resas/standard/parametres

modifier les nœuds « bloc\_avant » et « bloc\_apres »

## Traitements par lot

Un traitement par lot désigne une opération que l'on effectue sur un grand nombre d'objets. Par exemple, modifier un même champ d'un grand nombre de notices.

C'est un outil extrêmement puissant et utile quand on veut gagner du temps et éviter de répéter la même opération de nombreuses fois, mais il peut être dangereux si on l'utilise mal. Il faut donc être très prudent quand on exécute un traitement par lot.

Les traitements par lot peuvent être exécutés soit sur tous les objets d'un certain type (par exemple toutes les notices, tous les lecteurs...) ou bien sur une sélection d'entre eux contenue dans un panier (le panier peut être statique ou dynamique).



Les traitements par lot peuvent porter sur tous les types d'objets utilisés par Waterbear (notices, exemplaires, lecteurs, prêts, réservations...)

Vous avez également la possibilité de créer vos propres traitements en utilisant le système de plugins de Waterbear, mais c'est une opération d'une grande technicité.

Pour accéder aux traitements par lot, depuis la page d'accueil : menu traitements >> traitements par lot.

Vous devez commencer par sélectionner le type d'objets sur lequel portera le traitement (notices, exemplaires, lecteurs...)

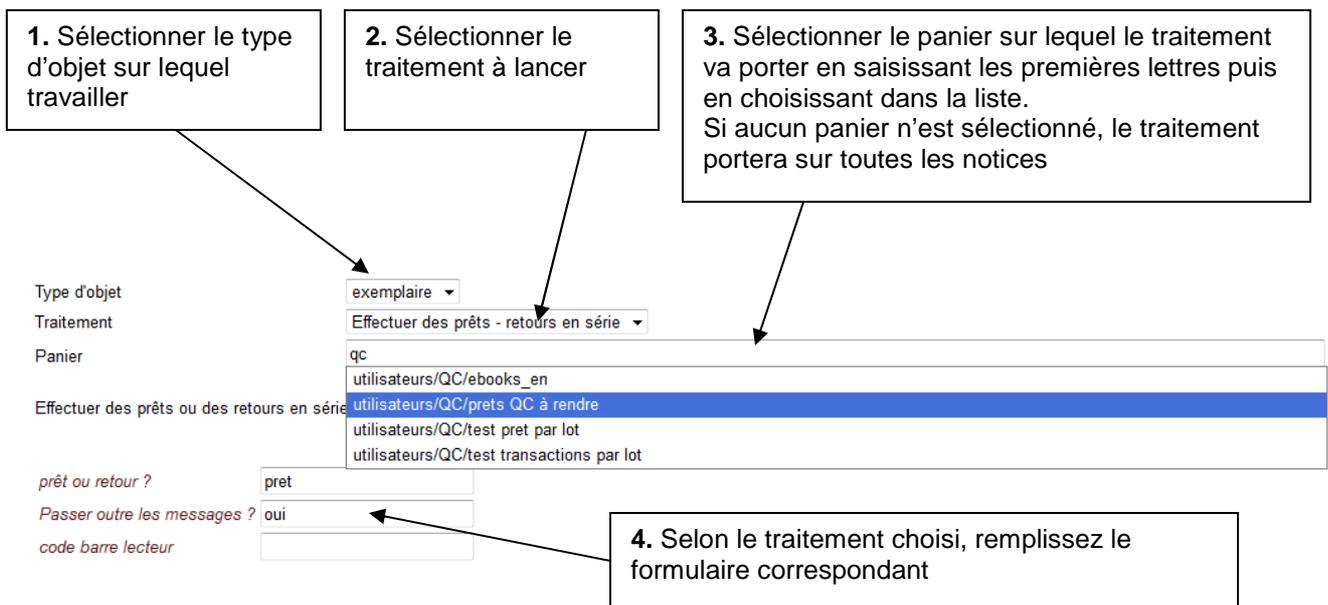
Ensuite sélectionnez le traitement que vous souhaitez effectuer. Selon le traitement choisi, un formulaire peut apparaître. Chaque traitement aura un formulaire différent.

Remplissez le formulaire en vous référent aux indications fournies. Une aide peut être associée à chaque champ du formulaire. Pour la voir, laissez simplement la souris quelques secondes sur le nom du champ.

Avant de valider, vous devez choisir sur quelles notices portera le traitement. Pour cela, sélectionnez un panier en saisissant ses premières lettres puis en choisissant dans la liste de propositions. Le panier peut être statique ou dynamique.

**ATTENTION : si vous ne sélectionnez aucun panier, le traitement portera sur la totalité de la base.**

Le traitement peut ensuite être assez long. Vous pouvez visualiser l'avancée des traitements.



Parmi les traitements, certains sont communs à tous les types d'objets, d'autres sont spécifiques à un type d'objet particulier.

### Traitements communs à tous les objets

1. Réindexation : ce traitement permet de régénérer les critères de recherche et de tri. Par exemple, si vous avez créé un nouveau critère de recherche, il s'applique immédiatement aux nouvelles notices créées, mais pas aux anciennes. Pour cela, il faut réindexer la base.
2. Ajouter un champ : permet de créer un champ contenant certains sous-champs. Vous indiquerez le nom du sous champs ainsi que les sous-champs qui le composent et leur valeur.
3. Modifier un champ existant : Vous pouvez modifier les sous-champs d'un champ existant. Pour chaque sous-champ vous pouvez spécifier une nouvelle valeur ainsi que des options : type de modification :  
 add -> ajoute un ss-champ ;  
 update -> modifie si ss-champ existe ;  
 add\_update -> modifie si ss-champ existe sinon ajoute ;  
 add\_si\_absent -> ajoute ss-champ si n'existe pas ;  
 delete -> supprime le ss-champ

### Traitements spécifiques à certains objets

1. Effectuer des prêts / retours en série (objet exemplaires) : ce plugin permet de prêter ou rendre un panier d'exemplaires.
2. Pilonner des exemplaires : associe les statut pilon à un panier d'exemplaires.

## Administration de Bokeh



Cette rubrique ne concerne que la version en ligne de Waterbear

Il est important de comprendre que Waterbear et Bokeh sont deux logiciels différents. Waterbear est le SIGB (partie professionnelle) : il sert à faire du catalogage, du prêt...

Bokeh permet un accès par le public aux informations créées dans Waterbear via le catalogue en ligne, le compte lecteur et les réservations (OPAC). Il permet également de constituer un portail web en créant des pages (CMS).

Chaque logiciel est donc autonome, mais il existe des échanges d'informations entre Waterbear et Bokeh :

- Les exports : Toutes les notices créées / modifiées dans Waterbear sont exportées vers Bokeh qui possède donc « un double » de la base de Waterbear. Ces exports ont lieu plusieurs fois par jour pour les exports incrémentiels (seules les créations de nouvelles notices et les modifications sont exportées) et le dimanche pour un export total.
- Certains échanges ont lieu en temps réel via des web services : l'affichage de la disponibilité des exemplaires, l'affichage du compte lecteur, les réservations.

Bokeh dispose de 2 interfaces d'administration : l'une généraliste permettant essentiellement d'administrer le CMS (création et paramétrage des pages web constituant le site). L'autre appelée cosmogramme va permettre de gérer les échanges entre Waterbear et Bokeh.

Pour accéder à l'administration de Bokeh, vous devez d'abord accéder au portail en allant sur <http://mabib.fr/XXX> (en remplaçant XXX par votre domaine).

Ensuite, vous accéderez à l'administration standard de Bokeh en allant sur <http://mabib.fr/admin> et à Cosmogramme en allant sur <http://mabib.fr/cosmogramme>

Les identifiants sont les mêmes que pour Waterbear.

Vous trouverez des documentations sur l'administration de Bokeh ici : <http://wiki.bokeh-library-portal.org/index.php/Bokeh>

**ATTENTION** : Ne modifiez rien dans Cosmogramme sans savoir ce que vous faites.

## B. Administration avancée



Waterbear est un logiciel extrêmement paramétrable, mais l'utilisation des outils avancés décrits ci-dessous comporte un certain nombre de risques et est réservée à des utilisateurs avertis.

## ***La gestion des objets***

Waterbear permet de modifier dynamiquement la base de données pour s'adapter aux besoins des utilisateurs. C'est le cas en particulier si vous souhaitez paramétrer de nouveaux accès (critères de recherche) ou de nouveaux critères de tri.

Pour cela, il faut modifier la base de données et créer de nouvelles « colonnes » associées aux objets concernés.

Vous pouvez également les accès et tris actuellement paramétrés.

Pour accéder à la gestion des objets, vous devez vous connecter en administrateur, puis à partir de la page d'accueil : menu administration >> gestion des objets

Pour modifier un objet existant, vous devez le sélectionner dans le champ déroulant en haut à gauche. la liste de ses accès et tris existants s'affiche alors (un onglet pour les accès et un autre pour les critères de tri)

### **Pour modifier un des accès ou tris existants**

Cliquez dessus dans la liste puis modifiez les informations dans le formulaire de droite, et enfin validez .

Les informations à saisir sont assez spécifiques aux bases de données. Les seules véritablement importantes sont :

- Le nom de la colonne (les accès doivent commencer par « a\_ » et les tris par « t\_ »)
- Le type de colonne (varchar, text, int, date...). Les plus utilisés sont date (pour les dates), int (pour les nombres) et text (pour le texte)
- Le type d'index : normal (pour des nombres, des dates, des codes) ou fulltext si on souhaite des recherches plein texte ou dans le cas de champs multivaleurs (champs répétables par exemple)

### **Pour créer un nouvel accès ou tri**

Cliquez sur « Nouvel accès » ou « nouveau tri » en bas de la liste, puis renseignez le formulaire sur la droite de la même manière : le nouvel accès/tri s'ajoute alors à la liste.

### **Pour supprimer un accès ou un tri**

Cliquez dessus dans la liste de gauche, puis cliquez sur l'icône  dans le formulaire de droite.

## Pour créer / supprimer un type d'objet

Saisissez le nom de l'objet dans le champ en haut à droite et cliquez sur . Pour le supprimer, sélectionnez l'objet dans le champ déroulant de gauche, puis allez dans le menu objet >> supprimer l'objet sélectionné

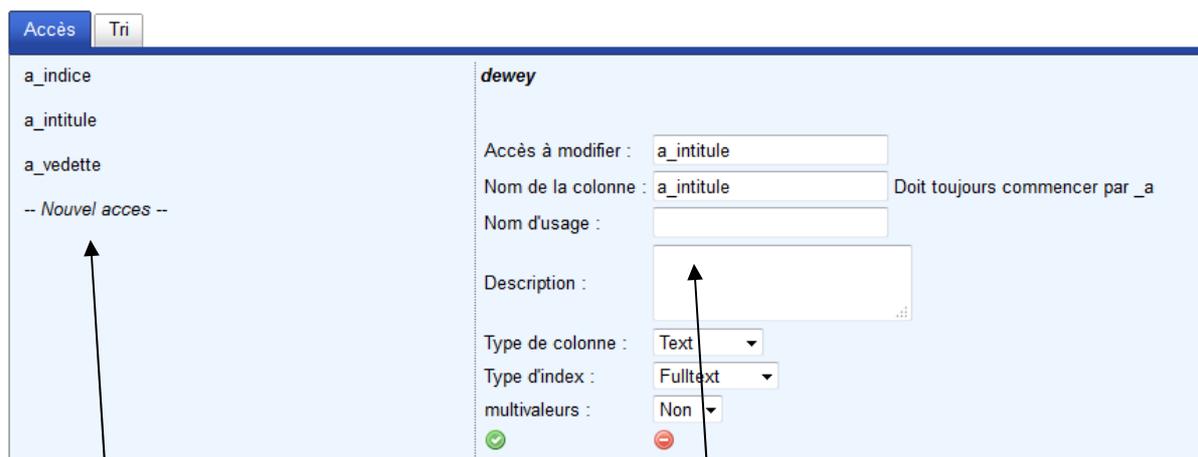
## Pour vider une table

Vider une table permet de supprimer tous les objets d'un certain type : par exemple, tous les lecteurs, tous les prêts. A utiliser avec grande précaution (typiquement quand on est en phase de test).

Pour cela, sélectionnez l'objet dans le champ déroulant de gauche puis cliquez sur « vider la table XXX » en haut à droite de l'écran.

On commence par sélectionner le type d'objet sur lequel on veut travailler

Objets : dewey    Créer un nouvel objet :      **Vider la table dewey**



On sélectionne ensuite l'accès / le tri que l'on veut modifier ou on clique sur « nouvel accès »

On remplit ensuite le formulaire



Attention, les modifications dans la gestion des objets doivent être accompagnées de paramétrages dans le registre, faute de quoi on obtiendra des erreurs sévères

## ***Le registre – interface graphique***

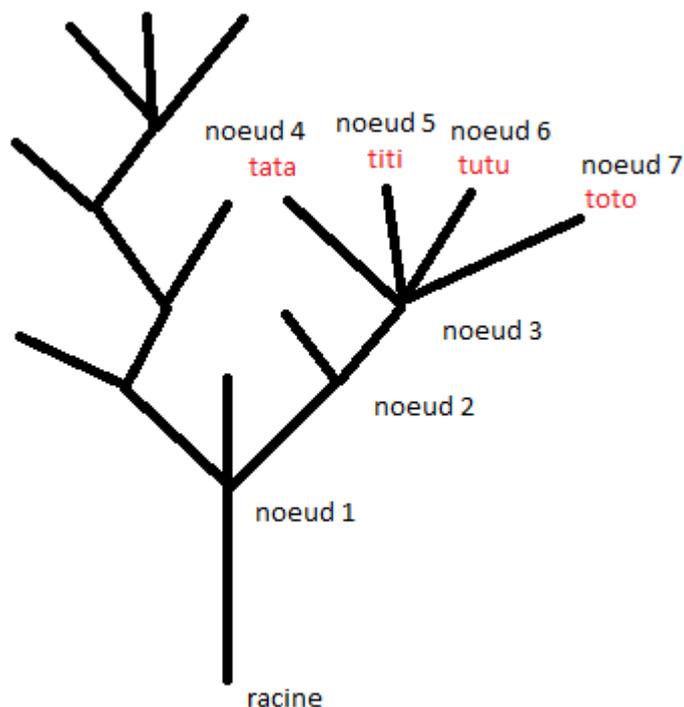
Le registre est le cœur de Waterbear. Tout le paramétrage de Waterbear y est contenu, mais c'est un euphémisme, car le registre contient plus de 60.000 nœuds. C'est qu'en vérité, Waterbear est conçu de telle sorte qu'une partie considérable de son développement a été transféré dans le registre, ce qui fait que les utilisateurs peuvent modifier le fonctionnement de Waterbear et l'adapter à leurs besoins.

Cette grande paramétrabilité a son revers : les paramétrages avancés peuvent parfois être assez complexes et nécessiter une certaine expertise.

Pour accéder au registre, il faut être connecté en administrateur. Depuis la page d'accueil : menu administration >> registre

Le registre se présente sous la forme d'un arbre avec une racine et des branches, qui se subdivisent à leur tour et ainsi de suite sur plusieurs dizaines de niveau. Chaque branche, que l'on appelle plutôt « nœud » possède un nom. Il peut soit avoir une valeur, soit se subdiviser en sous nœuds.

### **Illustration d'un arbre avec ses nœuds qui peuvent se subdiviser ou contenir des valeurs (en rouge)**





## Modifier un nœud

Pour modifier un nœud, il faut cliquer sur son nom. On obtient alors sur la partie droite de l'écran un petit formulaire qui va permettre de modifier le nœud. On pourra modifier :

- Son nom (le nom qui apparaît dans l'arborescence)
- Sa valeur. **NOTE** : un nœud ne peut avoir une valeur que s'il n'a pas de sous-nœuds)
- Sa description (optionnel)

Une fois la modification effectuée, il faut l'enregistrer en cliquant sur l'icône  dans la barre d'icônes

Chemin :  
profiles/default/plugins/plugins/catalogue/import

Nom du nœud :  
nom\_plugin

Valeur :  
catalogue/import\_export  
/formate\_ss\_champ

Description :

Le chemin indique l'emplacement du nœud dans l'arborescence du registre (non modifiable)

### **Supprimer un noeud**

Il faut cliquer sur le nœud (son détail s'affiche dans la partie droite de l'écran) puis cliquer sur  dans la barre d'icône (confirmation demandée)

**ATTENTION** : supprimer un nœud supprimera également tous ses sous-nœuds (son arborescence)

### **Créer un noeud**

Il faut d'abord se positionner dans le nœud qui sera le parent du nœud à créer en cliquant sur son nom.

**Rappel** : si on crée un nœud « b » dans le nœud « a », alors le nœud « a » ne peut pas avoir de valeur.

Ensuite il faut cliquer sur l'icône  dans la barre d'icônes. Cela va créer un nouveau nœud appelé « nouveau nœud » : il faudra ensuite renommer ce nœud et éventuellement lui donner une valeur et une description et enregistrer.

**Note** : deux sous nœuds d'un même nœud ne peuvent pas avoir le même nom

### **Recharger le registre**

A la fin de toutes vos modifications dans le registre, il faut réactualiser ce dernier en cliquant sur  dans la barre d'icônes. Cela est équivalent à se déconnecter puis se reconnecter. Cette manœuvre est à faire sur tous les postes connectés au moment du paramétrage.

## Copier / coller un nœud

Beaucoup de paramètres dans le registre peuvent se ressembler. Le fait de pouvoir copier/coller un nœud et son arborescence (ses sous-nœuds) permet de gagner énormément de temps et d'éviter les erreurs.

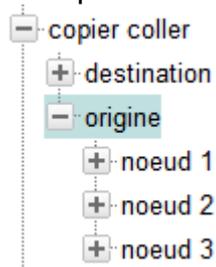
Pour copier un nœud (et l'ensemble de ses sous-nœuds s'il en a), cliquer sur son nom puis sur  dans la barre d'icônes. Le chemin complet du nœud copié apparaît dans le champ « presse-papier » en bas de l'écran.

Pour coller ce nœud et son arborescence, il faut se positionner dans le nœud où on veut le coller en cliquant sur son nom.

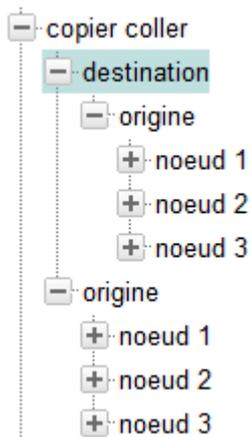
Ensuite on a deux possibilités : soit on colle directement le nœud en cliquant sur  soit on colle le contenu du nœud en cliquant sur  dans la barre d'icônes.

Voici une illustration :

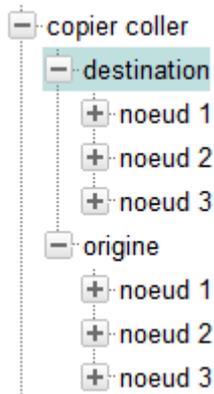
1. on copie le nœud « origine » en cliquant sur 



2. Si je clique sur  (coller le nœud dans le nœud « destination »), j'obtiens



3. Si en revanche je clique sur  (coller les sous-nœuds dans le nœud « destination »), j'obtiens



### **Exporter visuellement un nœud et son arborescence**

Il peut être utile de d'exporter toute l'arborescence d'un nœud (ses sous-nœuds) pour avoir un aperçu synthétique de sa structure. Pour ce faire, il suffit de se cliquer sur le nœud à exporter puis sur  dans la barre d'icônes.

On obtient alors quelque chose du genre :

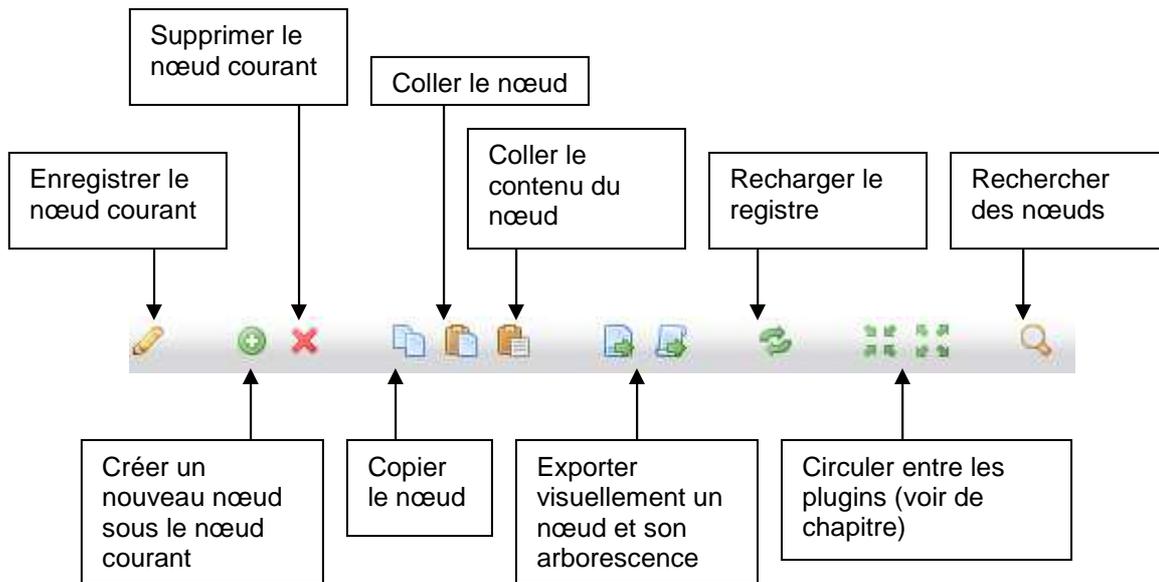
```
+ 100
+ ss_champs
  + a
    + 01 - supprimer espaces vides
      + nom_plugin => catalogue/import_export/formate_ss_champ
      + parametres
        + plugin_formate
          + nom_plugin => div/util_string
          + parametres
            + traitements
              + 01 - supprimer espaces vides
                + methode => str_replace
                + remplace
```

Ce qui donne une bonne image de l'arborescence : les nœuds / sous-nœuds et les valeurs (ici mises en rouge)

### **Rechercher un nœud par son nom ou sa valeur**

Si l'on ne connaît pas le chemin exact d'un nœud, on peut accéder à une interface de recherche simplifiée en cliquant sur 

## Récapitulatif de la barre d'icônes



## Le registre – fonctionnement du framework



La section qui va suivre est très technique. Même si le paramétrage du registre n'est pas de la programmation, son utilisation avancée fait appel à certaines notions qui s'en rapprochent (paramètres, héritage, variables...)

A ce titre, cette section est réservée aux utilisateurs avancés qui souhaitent consacrer beaucoup de temps au paramétrage fin de Waterbear (les passionnés, les responsables système, les personnes souhaitant proposer des prestations payantes autour de Waterbear...)

Pour les autres, il est préférable de sauter cette section et :

- Soit se référer à des paramétrages « clef en main » décrits plus loin dans ce guide, sans forcément comprendre tous les détails de leur mise en œuvre
- Soit faire appel à la communauté via la liste de diffusion
- Soit contracter une maintenance payante auprès d'un partenaire

C'est grâce à l'utilisation du registre que Waterbear est aussi paramétrable. La plupart des logiciels sont écrits dans divers langages de programmation et autorisent certains paramétrages qui sont stockés dans la base de données.

La conception de Waterbear est différente. Elle repose sur un framework spécifique dans lequel les paramétrages de la base de données et le code sont indissociables l'un de l'autre.

Cela permet de modifier certains comportements du logiciel en modifiant uniquement les paramétrages du registre, là où pour d'autres logiciels, il faudrait modifier le code informatique.

Ce framework repose sur deux mécanismes principaux :

- La gestion des pages
- La gestion des plugins

## La gestion des pages

Dans Waterbear, chaque page qui va s'afficher (et chaque Web Service utilisé) correspond à un nœud du registre.

Dans ce nœud, on pourra paramétrer énormément d'informations (titre et icône de la page, scripts php, javascript et css utilisés, barres d'icônes et barres d'outils, web services et plugins utilisés...).

Il est possible de créer de nouvelles pages avec de nouveaux paramètres en rajoutant des nœuds dans le registre.

Chaque page de Waterbear à une url du type :

<http://waterbear.info/bib.php?module=transactions/prets/standard>

Cela concerne également les Web Services de Waterbear qui ne s'affichent pas mais permettent aux différentes pages de récupérer dynamiquement des informations. Leur url a la forme :

[http://waterbear.info/bib\\_ws.php?module=transactions/prets/standard](http://waterbear.info/bib_ws.php?module=transactions/prets/standard)

Les scripts bib.php et bib\_ws.php sont les seuls qui sont appelés dans Waterbear. Ce qui va déterminer quelle page va s'afficher, est la valeur de la variable « module ».

Dans l'exemple précédent, module vaut « transactions/prets/standard » Cela signifie que l'on doit afficher la page qui est paramétrée dans le registre à l'emplacement :

profiles/default/pages/bib/**transactions/prets/standard** (pour les pages)  
profiles/default/pages/bib\_**ws**/**transactions/prets/standard** (pour les web\_services)

Chaque page définie dans le registre contient des informations (sous-nœuds) qui indiquent comment la page doit s'afficher et quel sera son comportement. Voici les principaux sous-nœuds :

- **\_page** : ce nœud contient l'emplacement du script php qui doit être appelé. C'est ce script qui va exécuter les opérations (par exemple recherches et modifications de la base de données...) avant l'affichage à l'écran.
- **\_template** : ce nœud contient des sous-nœuds qui déterminent comment les informations vont s'afficher à l'écran. Les templates sont en quelque sorte des modèles d'affichage. Plusieurs templates peuvent être appelés correspondant à différentes parties de la page. C'est également ici qu'on définira quelles feuilles de style CSS et quels scripts javascript seront utilisés.

- **\_parametres** : ce nœud contient des sous-nœuds qui fournissent des informations complémentaires.  
C'est ici par exemple, qu'on indiquera quelles informations doivent figurer dans les barres de menus et les barres d'icônes.  
On pourra également déterminer le titre de la page et son icône (qui s'affichent en haut du navigateur).  
Enfin, le plus important, on fournira des paramètres spécifiques au script PHP utilisé, comme les noms des web-services et des plugins (ce concept est défini plus loin) à utiliser.

Voici un exemple du nœud du registre « profiles/default/pages/bib/transactions/resas/standard » qui correspond à la page qui s'affiche quand on veut effectuer une réservation. Les valeurs des nœuds apparaissent en rouge.

```
+ _page => bib/transactions/resas.php
+ _parametres
  + favicon => IMG/icones/key_add.png
  + plugin_liste_bibs
    + nom_plugin => div/get_liste_choix
    + parametres
      + nom_liste => catalogue/catalogage/grilles/exemplaire/bibliotheque
  + titre_page => bib/transactions/resas/standard/titre_page
  + ws_doc => bib_ws.php?module=autocomplete/biblio/standard/vedette&
  + ws_lecteur => bib_ws.php?module=autocomplete/lecteur/standard/vedette&
  + ws_url => bib_ws.php?module=transactions/resas/standard
+ _template
  + include_css
    + 09 - resas => css/resas.css
  + include_js
    + 03 - datasource => js/yui/datasource/datasource-min.js
    + 04 - autocomplete => js/yui/autocomplete/autocomplete-min.js
  + tpl_javascript => bib/transactions/resas/javascript.php
  + tpl_main => bib/transactions/resas.php
```

On reconnaît les 3 nœuds principaux : **\_page**, **\_parametres** et **\_template** (en gras).

Dans les templates, on reconnaît des appels à des fichiers css (qui déterminent l'apparence de la page : couleur, taille des caractères...) et javascript.

Dans **\_parametres**, on voit qu'on définit un plugin (**plugin\_liste\_bibs** en bleu) qui sert à afficher la liste des bibliothèques dans lesquelles on veut retirer sa réservation.

On y détermine également 3 web services (en vert) :

**Ws\_doc** qui sert à l'affichage du champ d'auto complétion quand on commence à saisir le titre du document à réserver

**Ws\_lecteur** : qui sert à l'affichage du champ d'auto complétion quand on commence à saisir le nom du lecteur réservataire

**Ws\_url** qui est le web service appelé pour valider la réservation.

A travers ce simple exemple, on se rend compte de l'étendue de ce qu'on pourrait paramétrer, pour cette seule page de création d'une réservation :

Non seulement il est possible de modifier des choses « superficielles » comme le titre de la page ou la manière dont les informations apparaissent.

Mais on peut modifier des choses beaucoup plus fondamentales : par exemple, si on ne veut pas que le lecteur puisse récupérer sa réservation dans toutes les

bibliothèques mais seulement dans certaines d'entre elles, on modifiera le plugin  
« plugin\_liste\_bibs »

Si on souhaite (exemple farfelu mais qui illustre l'étendue des possibilités) qu'il ne soit possible de réserver que des ouvrages de plus de 5 ans, on modifiera le plugin  
« ws\_doc »... et ainsi de suite !

Par ailleurs, on n'est pas obligé de modifier cette page de réservation des documents telle quelle. On peut très bien la dupliquer et créer plusieurs variantes de cette page de réservation. Pour cela, il suffit de dupliquer dans le registre le nœud

« profiles/default/pages/bib/transactions/resas/standard » et de créer des nœuds

« profiles/default/pages/bib/transactions/resas/standard**2** »

« profiles/default/pages/bib/transactions/resas/standard**3** »

...

Chacun avec des paramétrages différents, ce qui créera des pages de réservations distinctes, avec chacune ses spécificités.

## Les plugins

Dans les logiciels développés de manière « traditionnelle », le code est divisé en petites entités appelées « fonctions » ou « méthodes ». Chaque fonction se charge de réaliser une petite opération assez simple. Un programme complexe résulte de l'enchaînement d'une multitude de fonctions et de leurs interactions.

La manière dont les fonctions sont définies, s'enchainent les unes après les autres et interagissent est définie dans le code et ne peut être modifiée.

Le framework de Waterbear fonctionne différemment. A chaque fonction PHP utilisée correspond une définition dans le registre appelée « plugin ». Dans ce plugin, on va pouvoir indiquer un certain nombre d'informations (paramètres) et en particulier le nom des autres plugins utilisés par cette fonction.

Cela rend le code de Waterbear totalement modulaire. Il est possible de modifier, dans le registre, les plugins utilisés et leurs paramètres. Ces modifications peuvent se faire de manière globale, ou bien différemment selon les utilisateurs.

### Déclarer un plugin dans le registre

Dans le registre, tous les plugins sont déclarés dans `profiles/default/plugins/plugins`. Ils sont ensuite répartis dans les différents sous-répertoires.

La déclaration d'un plugin prend la forme suivante :

```
+ Nom du plugin
  + chemin_fichier => catalogue/catalogage/grilles
  + nom_fonction => enregistrer_notice
  + parametres
    + xxx
    + yyy
    + zzz
```

Il faut commencer par créer un nœud pour le plugin correspondant à son nom, puis créer 3 sous-nœuds : `chemin_fichier`, `nom_fonction` et `parametres`.

`Chemin_fichier` et `nom_fonction` correspondent à l'emplacement et au nom du fichier PHP contenant la fonction à utiliser.

Dans le nœud « `parametres` » on créera des sous-nœuds correspondant aux différentes informations dont a besoin le plugin.

Parmi ces informations, certaines ont un statut particulier : les sous-plugins. En effet, la plupart du temps un plugin a besoin d'autres plugins pour fonctionner. Le nom de ces sous-plugins doit être indiqué dans les paramètres du plugin (voir rubrique « appeler un plugin »).

## Voici un exemple :

```
+ chemin_fichier => catalogue/marcxml
+ nom_fonction => crea_marcxml
+ parametres
  + definition
    + 02 - 200
      + definition
        + 01 - a =>
          + ##valeur => titre_fascicule
          + code => a
      + tag => 200
    + 03 - 201
      + definition
        + 01 - a =>
          + ##valeur => numero
          + code => a
        + 02 - b
          + ##valeur => date
          + code => b
      + tag => 201
    + 04 - 461
      + definition
        + 01 - v =>
          + ##valeur => numero
          + code => v
      + plugin_get_lien_explicite
        + nom_plugin => catalogue/marcxml/db/get_lien_explicite
        + parametres
          + ##ID => ID_periodique
          + type => biblio
      + tag => 461
```

« profiles/default/plugins/plugins/catalogue/periodiques/crea\_fascicule/standard » est un plugin appelé au moment du bulletinage pour créer la notice du fascicule. Parmi les paramètres, on reconnaît la liste des champs et sous-champs à créer (200\$a, 201\$a, 201\$b, champ 461...) et on voit que pour le champ 461, un sous-plugin « catalogue/marcxml/db/get\_lien\_explicite » est appelé. Ce sous-plugin va extraire les informations nécessaires de la notice du périodique pour les afficher dans le champ 461.

On voit dans cet exemple que les paramètres du plugin peuvent être extrêmement ramifiés. Certains plugins peuvent avoir des centaines de paramètres sur des dizaines de niveaux différents.

Cette façon de faire, rend Waterbear extrêmement paramétrable. Par exemple, si on souhaite rajouter ou modifier des champs/sous-champs aux fascicules créés au moment du bulletinage, on pourrait le faire dans ce plugin.

## Appeler un plugin

Pour être exécuté, le plugin qui a été déclaré dans le registre doit être appelé. Cela se paramètre également dans le registre. Il y a deux moments pour appeler un plugin :

- Soit dans une page
- Soit dans un autre plugin (on parle alors de sous-plugin)

Dans les 2 cas, c'est la page ou le plugin appelants qui déterminent quels plugins doivent être appelés. Par exemple, dans la page `profiles/default/pages/bib_ws/catalogue/periodiques/bulletinage/standard` qui s'exécute quand on veut bulletiner une revue, il faudra appeler les plugins :

- `Plugin_calcule_prochaine_date` : qui sert à calculer la date de réception attendue du prochain numéro
- `Plugin_ddbl_cab` : qui sert à s'assurer que le code barre n'est pas déjà utilisé
- `Plugin_crea_fascicule` : qui va générer la notice du fascicule en marcxml
- ...

Pour appeler les plugins, la syntaxe est toujours la même :

```
+ nom de l'appel
  + nom_plugin => chemin du plugin à appeler
  + paramètres
    + xxx
    + yyy
    + zzz
```

Voici ce que ça donne pour l'exemple ci-dessus :

```
+ _parametres
  + plugin_calcule_prochaine_date
    + nom_plugin => catalogue/periodiques/calcule_prochaine_date
  + plugin_crea_fascicule
    + nom_plugin => catalogue/periodiques/crea_fascicule/standard
  + plugin_ddbl_cab
    + nom_plugin => catalogue/recherches/recherches_util/exemplaire/par_cab
```

Ces plugins sont appelés depuis une page, ils se trouvent donc sous le nœud « `_parametres` ». S'ils avaient été appelés depuis un autre plugin (sous-plugins) ce serait exactement la même syntaxe, sauf qu'ils se trouveraient sous le nœud « `parametres` » (sans tiret souligné).

On a vu dans la rubrique précédente que lorsqu'on déclare un plugin, on peut lui associer des paramètres. Quand on appelle ce plugin, on peut également lui fournir des paramètres. Les paramètres fournis dans la définition du plugin et au moment de l'appel fusionnent et se combinent.

Par exemple, si on définit le plugin profiles/default/plugins/plugins/test/toto comme suit :

```
+ toto
  + chemin_fichier => catalogue/marcxml
  + nom_fonction => crea_marcxml
  + parametres
    + param1 => toto
```

Et qu'on l'appelle dans une page ou un plugin de la manière suivante :

```
+ parametres
  + plugin_toto
    + nom_plugin => profiles/default/plugins/plugins/test/toto
    + parametres
      + param2 => tutu
```

Au total, 2 paramètres seront fournis au plugin : param1 (qui vaut « toto ») et param2 (qui vaut « tutu »)

Si jamais on fournit 2 paramètres ayant le même nom dans la déclaration du plugin et lors de son appel, c'est le paramètre d'appel qui est prioritaire.



### Plugins statiques et dynamiques

La plupart du temps, quand on appelle un plugin, on l'a préalablement déclaré dans le registre. Le chemin fourni dans le nœud « nom\_plugin » correspond alors à l'emplacement de la définition du plugin dans le registre. On parle de **plugins dynamiques**.

Cependant, il peut arriver qu'on n'ait pas besoin de fournir de paramètres au plugin au moment de sa déclaration. Cette dernière ne servant à rien, on peut appeler directement le script PHP. Dans ce cas, le chemin fourni dans le nœud « nom\_plugin » correspond à l'emplacement du fichier PHP sur le serveur. On parle de **plugins statiques**.

## ##, !! et ?? : les codes secrets des plugins

Certains nœud du registre sont précédés de caractères de ponctuation doublés : ##, !! ou ??. Ces codes ont une signification spéciale pour Waterbear.

### ## : les variables incluses

Waterbear permet d'intégrer aux paramètres fournis à un plugin des variables fournies par le script PHP. C'est très utile, entre autres, pour les plugins de création de notices marcxml ou de recherche dans la base de données.

Prenons un exemple : la page `profiles/default/pages/bib_ws/catalogue/periodiques/bulletinage/standard` sert à bulletiner. Selon les commandes de l'utilisateur, elle peut utiliser plusieurs plugins, parmi lesquels le plugin `catalogue/periodiques/crea_fascicule/standard` qui sert à générer la notice du fascicule en marcxml.

Ce plugin contient dans ses paramètres la structure de la notice à créer (champs, sous-champs...), mais c'est la page appelante qui va lui fournir les paramètres qui lui manquent : le titre du fascicule, son numéro, la date de bulletinage. Ces variables vont être réinjectées dans la structure de la notice à créer en utilisant les variables incluses :

```
+ chemin_fichier => catalogue/marcxml
+ nom_fonction => crea_marcxml
+ parametres
  + definition
    + 02 - 200
      + definition
        + 01 - a =>
          + ##valeur => titre_fascicule
          + code => a
        + tag => 200
      + [...]
```

Dans cet exemple, la ligne en rouge correspond à une variable incluse. La page appelante a passé en paramètre au plugin la variable « titre\_fascicule » qui correspond au titre du numéro saisi dans l'interface de bulletinage. Admettons que l'utilisateur ait saisi « le Nouvel observateur n° 12345 : le retour de la franc-maçonnerie ». La variable « titre\_fascicule » a cette valeur, et lors de l'exécution du plugin, la ligne

**+ ##valeur => titre\_facscicule**

Se transformera en

**+ valeur => le Nouvel observateur n° 12345 : le retour de la franc-maçonnerie**

Ce qui conduira à la création d'un sous-champ `200$a` ayant la valeur « le Nouvel observateur n° 12345 : le retour de la franc-maçonnerie »

## ?? : la gestion du multilinguisme

Waterbear est un logiciel multilingue. Cela signifie que tous les textes paramétrés dans le registre doivent pouvoir être dédoublés en autant de langues que nécessaire.

Prenons l'exemple du plugin  
profiles/default/plugins/plugins/transactions/resas/traite\_resa\_pret/standard qui est appelé lorsqu'on veut prêter un document réservé par une autre personne.

Entre autres paramètres, ce plugin contient un nœud « message » qui correspond au message d'erreur qui sera renvoyé à l'utilisateur qui cherchera à emprunter ce document.

La forme habituelle de ce plugin serait donc :

```
+ chemin_fichier => transactions/resas
+ nom_fonction => traite_resa_pret
+ parametres
  + message => Vous ne pouvez emprunter ce livre car il est réservé
```

Mais cette façon de faire ne gère pas le multilinguisme. Si on se connecte à Waterbear en anglais, on aura toujours un message en français.

Pour cela, on utilisera la syntaxe suivante :

```
+ chemin_fichier => transactions/resas
+ nom_fonction => traite_resa_pret
+ parametres
  + ??message =>
profiles/default/langues/bib/transactions/prets/standard/_intitules/message_
doc_reserve
```

On préfixe le nom du nœud « message » de 2 points d'interrogation, et on va lui donner pour valeur l'emplacement du registre où seront définis les intitulés de ce message dans les différentes langues : ici

```
profiles/default/langues/bib/transactions/prets/standard/_intitules/message_cab_lecteur_inconnu
```

A cet endroit du registre on définira les intitulés comme suit :

```
+ message_cab_lecteur_inconnu
  +_fr => Vous ne pouvez emprunter ce livre car il est réservé
  +_en => You can't loan this book as it's reserved
```

## !! : les plugins inclus

Waterbear permet d'injecter le résultat d'un plugin dans les paramètres d'un autre en utilisant la syntaxe « !! ». Ce mécanisme est appelé « plugin inclus ».

Il ne faut pas confondre les plugins inclus et les sous-plugins vus plus haut. L'utilisation d'un sous-plugin est entièrement déterminée par la page ou le plugin dans laquelle il est appelé.

Un plugin inclus, en revanche peut être utilisé dans n'importe quel plugin sans que cela ait forcément été prévu.

Dans l'exemple suivant, on va définir un plugin qui sert à effectuer une recherche dans la base de données. On fournit en paramètres à ce plugin les différents critères de recherche (titre...). Parmi ces critères on voudrait ajouter un critère sur la date de création de la notice : rechercher uniquement des ouvrages qui ont été catalogués il y a moins de 15 jours. Comme ce plugin peut être utilisé n'importe quand, on ne peut pas mettre la date « en dur ». On va donc utiliser le plugin `div/util_dates` qui permet de retourner et modifier des dates :

```
+ chemin_fichier => catalogue/recherches
+ nom_fonction => recherche_simple
+ parametres
  + param_recherche
    + bool_parse_contenu => 0
    + criteres
      + 01 - titre
        + ##valeur_critere => titre
        + intitule_critere => a_titre
        + type_recherche => str_contient
      + 02 - date saisie
        + !!valeur_critere
          + nom_plugin => div/util_dates
          + parametres
            + nb_modif => 15
            + date
            + format_sortie => us
            + modif => moins
            + unite_modif => j
          + intitule_critere => a_date_saisie
          + type_recherche => int_egale
        + format_resultat => donnees
    + type_objet => biblio
```

Le plugin `div/util_dates` va retourner la date du jour moins 15. En effet, ses paramètres sont :

- Date : on aurait pu fournir une date quelconque, mais si on laisse vide, c'est la date du jour
- Modif : « moins » : on veut soustraire des choses à la date du jour
- Unite\_modif : « j » (pour jour) : on enlève donc 15 jours

Lors de l'appel de ce plugin, toute la partie en rouge sera remplacée :

```
+ !! valeur_critere [...] deviendra
+ valeur_critere => 2015-08-11
```

L'utilisation principale faite des plugins inclus est de scinder le paramétrage parfois très important de certains plugins en petits blocs réutilisables.

La meilleure illustration en est le paramétrage des grilles de catalogage. Une grille de catalogage correspond à un plugin avec un paramétrage gigantesque qui récapitule tous ses onglets, pour chaque onglet tous ses champs, pour tous ses champs tous ses sous-champs, et pour tous ses sous-champs, toutes les propriétés.

Cela fait plusieurs milliers de paramètres pour une grille de catalogage, ce qui est peu pratique et peu maniable : si on veut créer une 2<sup>e</sup> grille de catalogage avec seulement quelques modifications, il faut tout dupliquer.

L'utilisation des plugins inclus combinée au plugin **div/plugins\_2\_array** permet de simplifier considérablement cela.

Le plugin `div/plugins_2_array` se contente de renvoyer les informations qui lui sont fournies en paramètre sans les modifier.

Ainsi si on paramètre un plugin A comme suit :

```
+ A
  + chemin fichier => div
  + nom_fonction => plugins_2_array
  + parametres
    + param1 => toto
    + param2 => tutu
```

Et un plugin B :

```
+ B
  + chemin fichier => div
  + nom_fonction => plugins_2_array
  + parametres
    + param3 => tata
    + param4 => titi
```

Et enfin un plugin C

```
+ C
  + chemin fichier => xxx
  + nom_fonction => xxx
  + parametres
    +champs
      + !!01 - champ A
        + nom_plugin => xx/xx/xx/A
      + !!02 - champ B
        + nom_plugin => xx/xx/xx/B
```

On obtiendra la même chose que si on avait saisi :

```
+ C
+ chemin_fichier => xxx
+ nom_fonction => xxx
+ parametres
  +champs
    + 01 - champ A
      + param1 => toto
      + param2 => tutu
    + 02 - champ B
      + param3 => tata
      + param4 => titi
```

Remarquez comme les « !! » ont disparu.

## Aide graphique à la navigation entre plugins

Avec l'utilisation des sous-plugins et des plugins inclus, la consultation du registre peut rapidement devenir un vrai casse-tête. Pour aider l'administrateur, Waterbear dispose de fonctionnalités permettant de circuler facilement de l'appel d'un plugin (que ce soit dans une page, via un sous-plugin ou un plugin inclus) à sa déclaration.

Pour cela, positionnez-vous sur le nœud « nom\_plugin » dans lequel le nom du plugin à appeler est inscrit, et cliquez sur l'icône  (flèches vers l'intérieur). Cela vous amène immédiatement à l'emplacement du registre où ce plugin est défini.

Cette fonctionnalité fonctionne aussi avec les nœuds commençant par « ?? » (multilinguisme). Si vous cliquez sur l'icône, vous arriverez à l'endroit du registre où les intitulés (dans les différentes langues) sont paramétrés.

**NOTE** : Vous pouvez, ouvrir cette déclaration dans un nouvel onglet du navigateur en cochant préalablement la case prévue à cet effet en haut à gauche de Waterbear. De cette manière vous obtenez 2 onglets : l'un avec la définition du plugin, l'autre avec son appel.

Cela peut également marcher dans l'autre sens : partir de la définition d'un plugin et afficher tous les endroits du registre où ce plugin est appelé. Pour cela, placez vous sur le nœud racine du plugin (le nœud qui contient les sous-nœuds

« chemin\_fichier » et « nom\_fonction ») et cliquez sur l'icône  (flèches vers l'extérieur).

Cela vous ouvre une interface de recherche dans un nouvel onglet avec la liste des appels à ce plugin. Il vous suffit de cliquer sur le chemin du nœud pour vous y rendre.

## Alias et alias\_retour

On l'a vu, le fonctionnement de Waterbear est basé sur un enchaînement de plugins. Grâce à certains mécanismes comme les sous-plugins ou les plugins inclus, cet enchaînement est totalement modulable. On peut remplacer un plugin par un autre, modifier ses paramètres...

Une difficulté qui survient parfois, est que les différents plugins n'attendent pas leurs paramètres de la même manière.

Par exemple, un plugin A peut attendre un paramètre appelé « texte » tandis qu'un plugin B attendra un paramètre appelé « chaine ». Or il arrive dans Waterbear qu'on souhaite remplacer dans le registre un plugin par un autre, pour modifier le comportement du logiciel.

Dans ce cas, il faudra utiliser des alias qui vont en quelque sorte renommer les variables fournies en paramètre au plugin.

Voici un exemple :

```
+ plugin_crea_objet
+ alias
  + variables|nom => definition/01/definition/02/valeur
  + variables|ville => definition/01/definition/01/valeur
+ nom_plugin => catalogue/marcxml/crea_marcxml
+ parametres
  + definition
    + 01
      + definition
        + 01
          + code => a
          + valeur => remplacé par alias
        + 02
          + code => c
          + valeur => remplacé par alias
    + tag => 200
```

Il s'agit d'un sous plugin utilisé pour créer une notice d'éditeur. Le sous plugin reçoit en paramètres de la part du script PHP « variables/nom » qui contient le nom de l'éditeur et « variables/ville » qui contient la ville de l'éditeur.

Pour générer la notice éditeur, le plugin définit également en paramètres la structure de la notice à créer (ici on a une structure très simple : 200\$c => nom et 200\$a => ville, mais on pourrait avoir besoin de rajouter d'autres champs).

L'objectif est donc de fusionner les paramètres fournis par le script (le nom de l'éditeur et la ville) avec les paramètres définis dans le registre (la structure de la notice avec ses champs et ses sous-champs).

Pour cela on va utiliser un alias (en rouge). Ici on va demander de renommer la variable variables/nom en definition/01/definition/02/valeur, c'est-à-dire l'emplacement où on souhaite avoir apparaître le nom de l'éditeur (en bleu).

Dans l'exemple précédent, admettons que le script PHP fournisse au plugin les paramètres :

- Variables/nom => Gallimard
- Variables/ville => paris

L'utilisation des alias ci-dessus transformera l'appel du plugin en ceci :

```
+ plugin_crea_objet
+ nom_plugin => catalogue/marcxml/crea_marcxml
+ parametres
  + definition
    + 01
      + definition
        + 01
          + code => a
          + valeur => Paris
        + 02
          + code => c
          + valeur => Gallimard
    + tag => 200
```

Au niveau de la syntaxe, on crée un nœud « alias » juste sous la racine de l'appel du plugin (au même niveau que le nœud « nom\_plugin »).

Les sous-nœuds d'alias correspondent aux renommages à effectuer : le nom du nœud correspond au nœud de la variable actuelle, la valeur du nœud correspond à la manière dont il faut la renommer. ainsi :

```
+alias
  + toto => tutu
```

Renommara la variable « toto » en « tutu ».

**NOTE** : les variables peuvent être simples, mais elles peuvent aussi être structurées sous la forme de ce qu'on nomme en informatique un tableau (array). C'est le cas dans l'exemple précédent où on a une variable fournie sous la forme d'une array à 2 niveaux (variables|nom) et est convertie en array à 5 niveaux (definition/01/definition/02/valeur). La syntaxe n'est pas tout à fait la même entre le nom de l'alias (où les niveaux sont séparés par des « | » et sa valeur (où les niveaux sont séparés par des « / »).

C'est simplement dû au fait qu'on ne peut pas utiliser le caractère « / » pour le nom d'un nœud. Dans ce contexte (et dans ce contexte seulement) le « | » remplace le « / »

On a exactement le même mécanisme pour les informations renvoyées par les plugins. Là encore, elles peuvent être structurées différemment selon les plugins, et il peut être utile de les renommer.

Prenons l'exemple d'un plugin A qui retourne une valeur « chaine ». Il est appelé dans un plugin B, qui, lui, attend une valeur « texte ».  
Pour contourner le problème, on utilisera un alias\_retour de la manière suivante :

```
+ plugin_xxx
  + nom_plugin => xxx/xxx/A
  + alias_retour
    + chaine => texte
```

La valeur de retour « chaine » est remplacée par « texte ».



### Alias et variables incluses

L'utilisation des alias ressemble beaucoup à ce qu'on avait vu avec les variables incluses (##). L'utilisation de ces dernières semblait même plus simple.

En vérité, le contexte est différent. Les variables incluses peuvent être utilisées dans la **définition** d'un plugin, tandis que les alias et alias\_retour sont utilisés lors de son **appel**.

## ***Tirer partie de l'héritage dans les pages***

On l'a vu plus haut, à chaque page de Watreabear qui s'affiche (ou à chaque web-service) correspond un nœud dans le registre. Ce nœud contient de nombreux sous-nœuds avec toutes les informations nécessaires à l'affichage de la page et à son exécution.

Ces informations sont contenues dans des sous-nœuds dont les principaux sont `_template` et `_parametres`.

Or il arrive couramment que plusieurs pages aient des paramètres en commun, particulièrement des pages qui sont « sœurs ». Par exemple, à chaque grille de catalogage correspond un nœud du registre. Dans chaque nœud, un paramètre est défini pour indiquer quels masques peuvent s'appliquer cette grille. Or pour l'ensemble des grilles de catalogage de notices bibliographiques (livres, cd, dvd...), ces masques sont les mêmes.

Waterbear possède donc un mécanisme permettant d'éviter de saisir plusieurs fois les paramètres redondants : l'héritage.

Le principe en est que les paramètres ou les templates définis au niveau d'un nœud sont hérités par tous ses sous-nœuds.

Par exemple, le paramétrage suivant

```
+ A
  + B
  + C
  +_parametres
    + toto => tutu
```

Est équivalent (mais plus élégant) que

```
+ A
  + B
    +_parametres
      +toto => tutu
  + C
    +_parametres
      +toto => tutu
```

Le paramètre « toto » défini dans le nœud A est hérité par ses sous-nœuds B et C

Par ailleurs, il est possible de surcharger un héritage, c'est-à-dire de redéfinir dans un nœud enfant un paramètre qui a déjà été défini dans le nœud parent. C'est le paramètre défini dans l'enfant qui l'emporte sur celui du parent. Pour reprendre l'exemple précédent :

```

+   A
    + B
    + C
    + D
      +_parametres
        + toto => tata

    +_parametres
      + toto => tutu

```

Ici, le paramètre “toto” vaudra “tutu” dans les nœuds A et B (par héritage) mais il vaudra “tata” dans le noeud D (surcharge).

Ce mécanisme d’héritage et de surcharge est particulièrement utilisé pour les menus et les barres d’icônes qui sont paramétrés dans le registre.

Pour chaque page, il est possible de paramétrer les menus et les icônes qui apparaîtront (dans le nœud `_parametres`), mais il serait extrêmement redondant de le faire à chaque fois. On paramètre donc des menus par défaut au niveau de chaque module (catalogage, prêt...) qui peuvent être surchargés (par exemple les icônes du catalogage des revues ne sont pas les mêmes que celles du catalogage des autres supports).

## **La personnalisation**

Il est possible de personnaliser les paramètres d'une page en fonction de l'utilisateur ou du poste (ou du groupe d'utilisateurs ou du groupe de postes).

Pour cela, on peut placer un noeud **\_profiles** dans le noeud **\_parametres**. Le noeud **\_profiles** contiendra un sous-noeud pour chaque profile que l'on voudra définir. On peut créer des profiles liés aux utilisateurs et aux postes.

Un profile à la forme suivante : **UTILISATEUR:POSTE** sachant que ces deux éléments peuvent décrire soit un utilisateur ou un poste particulier, soit des groupes. Dans le premier cas, on préfixe par "P", dans le deuxième par "G".

Ainsi :

Ptoto:Gtutu désigne l'utilisateur toto sur un poste du groupe tutu

Gtata:Gtutu : un utilisateur du groupe tata sur un poste du groupe tutu

Ptoto:Ptiti : l'utilisateur toto sur le poste titi

Ptoto:\* : l'utilisateur toto sur n'importe quel poste

\*:\* : n'importe quel utilisateur sur n'importe que poste

Il y a une hiérarchie des profiles. En gros, du plus particulier au plus large. Ainsi, pour un même noeud, on pourra définir un profil pour un groupe d'utilisateurs et un autre pour un utilisateur particulier du même groupe : c'est ce dernier qui sera utilisé. Dans l'ordre (du moins prioritaire au plus prioritaire) :

\*:\*, \*:GGposte, GGuser:\*, GGuser:GGposte, \*:Pposte, Puser:\*, GGuser:Pposte, Puser:GGposte, Puser:Pposte

L'intérieur du noeud **\_profiles/XXXX/** est le même que celui du noeud **\_parametres**. Cependant, il n'est pas nécessaire de dédoubler totalement ces 2 noeuds. Leurs contenus seront fusionnés. On ne déclarera donc dans le noeud du profile que les variables spécifiques. Si la même variable est définie dans **\_parametres** et dans **\_profiles/XXX**, c'est cette dernière qui l'emporte.

## **Bonnes pratiques de la personnalisation dans Waterbear**

Waterbear offre à l'utilisateur des possibilités de personnalisation bien supérieures aux autres logiciels de bibliothèque. Mais cette puissance de paramétrage peut complexifier les mises à jour du logiciel.

En effet, lorsqu'une nouvelle version de Waterbear paraît, les mises à jour concernent également le registre : des modifications sont apportées aux pages et aux plugins.

Dans certains cas, il peut arriver que ces mises à jour viennent écraser les modifications apportées par l'utilisateur. Pour éviter ce genre de choses, il est important d'observer quelques bonnes pratiques :

1. Si vous créez des nouveaux accès ou critères de tri, préfixez le avec un code. Par exemple, si votre bibliothèque s'appelle Jules Vernes, et que vous voulez

créer un accès « titre », appelez plutôt JV\_titre : pas de risque qu'une future mise à jour crée un accès avec le même nom.

2. A chaque fois que c'est possible, ne modifiez pas directement les paramètres des pages et des plugins, mais utilisez le nœud « \_profiles » (cf. ci-dessus). N'hésitez pas à utiliser le profile « \*:\* » valable pour tous les utilisateurs sur tous les postes. ces nœuds ne profiles ne pourront pas être écrasés lors d'une mise à jour.
3. Pour de grosses modifications, plutôt que modifier une page, ou un plugin, dupliquez le (et préfixez son nom avec vos initiales

## ***La gestion des droits dans Waterbear***

Il est possible d'autoriser l'accès à certaines pages (ou un web Service lié à une page), et ce pour certains utilisateurs, pour certains postes ou pour certains groupes.

Pour cela, il faut créer un nœud \_droits sous le nœud de la page. Le nœud \_droits contiendra un sous nœud pour chaque profile. Ces profiles ont la même syntaxe que pour la personnalisation. Par exemple, Ptoto:Gtutu désignera des droits assignés à l'utilisateur toto sur les postes du groupe tutu.

### **Les droits sur les pages**

Pour assigner des droits sur une page, il suffit de donner la valeur 0 (interdit) ou 1 (autorisé) au noeud du profil. Par exemple :

```
+ _droits
  + *:* => 0
  + Ptutu:Gtoto => 1
```

signifie qu'on interdit l'accès à tout le monde, sauf à l'utilisateur tutu quand il est sur un poste du groupe toto

### **Héritage et hiérarchie des droits**

Il n'est pas nécessaire d'attribuer des droits à chaque page. Waterbear utilise l'héritage des pages. Un droit attribué au niveau du parent sera automatiquement propagé à tous ses enfants. En revanche, il peut être écrasé (redéfini) au niveau d'un nœud enfant.

## C. Comment paramétrer vous-même...



Dans la section qui va suivre, vous allez apprendre à effectuer certains paramétrages avancés dans le registre. Cela nécessite d'avoir lu les chapitres « la gestion des objets » et « le registre – interface graphique ».

La lecture du chapitre « le registre – fonctionnement du framework » (très technique) n'est pas toujours indispensable. Si vous avez un bon niveau, elle vous aidera certainement à comprendre le sens de ce que vous faites, mais si ce n'est pas le cas, elle vous embrouillera plus qu'autre chose.

Comme d'habitude avec le paramétrage avancé, il faut être conscient qu'une erreur dans le paramétrage peut avoir des conséquences fâcheuses sur Waterbear. Soyez donc extrêmement prudent.

**Pour finir, n'oubliez pas de recharger le registre avant de tester vos modifications.**

## Paramétrer les grilles de catalogage

### Généralités

Dans Waterbear, la structure des données est totalement paramétrable. Vous avez la possibilité de créer vos propres champs et sous-champs. Et cela concerne la totalité des objets utilisés par Waterbear.

Nous allons montrer ici comment créer un nouveau champ 999 pour les lecteurs avec un sous-champ \$a.

Avant de paramétrer un champ, il faut paramétrer les sous-champs qui vont le composer : cela se fait dans le registre à l'endroit suivant :  
profiles/default/plugins/plugins/catalogue/catalogage/definitions\_ss\_champs.

Placez vous ensuite dans lecteur/unimarc\_standard à moins que vous préfériez placer vos sous-champs personnalisés dans un nœud spécifique (par exemple lecteurs/nom\_de\_votre\_bib.

Créer un nœud correspondant à votre sous-champ. Le plus simple est de copier – coller un autre sous-champ ayant les mêmes spécificités que le vôtre. Pour commencer, nous allons partir d'un sous-champ de type « textbox », les plus simples. Vous devriez avoir quelque chose qui ressemble à ceci :

```
+ 999_a
+ chemin_fichier => div
+ nom_fonction => plugins_2_array
+ parametres
  + !!evenements
    + nom_plugin =>
catalogue/catalogage/definitions_groupes_evenements/biblio/unimarc_standard/standard
  + !!icomes
    + nom_plugin =>
catalogue/catalogage/definitions_groupes_icomes/biblio/unimarc_standard/ss_champ_rien
  + ??intitule =>
bib/catalogue/catalogage/grilles/lecteur/unimarc_standard/ss_champ_999_a_description
  + auto_plugin =>
catalogue/catalogage/definitions_ss_champs/lecteur/unimarc_standard/999_a
  + nom => a
  + type => textbox
```

Parmi les paramètres on trouve :

- Le type du sous-champ (textbox) : c'est le plus simple : juste un champ de saisie (pas de liste déroulante, d'auto-complétion...)
- Le nom du sous-champ : a (999\$a)
- Son intitulé : la dénomination du sous-champ telle qu'elle apparaîtra dans la grille de catalogage. On aurait pu simplement mettre un intitulé, mais ici on utilise « ?? » pour permettre la gestion du multilinguisme.
- Le paramètre auto\_plugin doit toujours renvoyer le chemin du plugin lui-même

- Le paramètre « icones » permet de personnaliser les icônes liées à ce sous-champ : ici on ne souhaite aucune icône
- Le paramètre « evenements » permet de spécifier quels événements peuvent provoquer une action sur ce sous-champ (du genre cliquer dessus, cliquer en dehors, modifier le contenu...). Il s'agit ici de la valeur par défaut.

Une fois, le sous-champ paramétré, il va falloir l'associer à un champ. Vous pouvez soit l'associer à un champ existant, soit à un nouveau champ. Les champs sont déclarés dans le registre à l'emplacement suivant :

Voici la forme que cela prend :

```
+ 999 =>
+ chemin_fichier => div
+ nom_fonction => plugins_2_array
+ parametres
  + !!icones
    + nom_plugin =>
catalogue/catalogage/definitions_groupes_icones/biblio/unimarc_standard/ss_champ_rien
  + ??intitule =>
bib/catalogue/catalogage/grilles/lecteur/unimarc_standard/champ_999_description
  + auto_plugin =>
catalogue/catalogage/definitions_champs/lecteur/unimarc_standard/999
  + nom => 999
  + ss_champs
    + !!01-a
      + nom_plugin =>
catalogue/catalogage/definitions_ss_champs/lecteur/unimarc_standard/999_a
[...] on placera en dessous les autres sous-champs
```

Certains paramètres ressemblent à ceux des sous-champs (auto-plugin, nom, icones, intitule).

Le paramètre « ss\_champs » va contenir les appels vers les plugins des sous-champs composant ce champ sous la forme de plugins inclus ( !!).

Selon le même principe, on associera ce champ à un onglet qui se paramètre dans profiles/default/plugins/plugins/catalogue/catalogage/definitions\_onglets puis l'onglet dans une grille qui se paramètre dans profiles/default/plugins/plugins/catalogue/catalogage/definitions\_grilles.

Pour cette dernière partie, il y a 2 plugins par grille : un pour la création et l'autre pour la modification. Les appels vers les onglets sont à placer dans le plugin de création, tandis que le plugin de modification contient un lien vers celui de création.

**Si vous ne faites que modifier une grille existante, cela ne demande pas plus d'effort que cela. Si en revanche, vous souhaitez créer une nouvelle grille, il faudra faire un peu plus :**

Si par exemple on veut créer une nouvelle grille de catalogage des lecteurs appelée « toto », il faudra créer la page correspondante ainsi que la page de web service.

Commencez par créer le web service ici  
profiles/default/pages/bib\_ws/catalogue/catalogage/grilles  
Dupliquez le web\_service par défaut (unimarc\_standard) et collez le dans un nouveau nœud « toto » puis modifiez les paramètres plugin\_formulaire et plugin\_formulaire\_modification pour qu'ils pointent vers les plugins de la nouvelle grille que vous avez créés.

```
+ toto
+ _page => bib_ws/catalogue/catalogage/grilles.php
+ _parametres
  + plugin_formulaire
    + nom_plugin =>
catalogue/catalogage/definitions_grilles/lecteur/toto
  + plugin_formulaire_modification
    + nom_plugin =>
catalogue/catalogage/definitions_grilles/lecteur/toto_modification
  + plugin_switcher
    + nom_plugin =>
catalogue/catalogage/switchers/lecteur/unimarc_standard
```

Ensuite créez la page à l'emplacement  
profiles/default/pages/bib/catalogue/catalogage/grilles.  
Là encore, dupliquez la grille par défaut (unimarc\_standard) et appelez la « toto »

```
+ toto
+ _page => bib/catalogue/catalogage.php
+ _parametres
  + masque_defaut => default
  + page_ws =>
bib_ws.php?module=catalogue/catalogage/grilles/lecteur/toto
  + plugin_get_masques
    + nom_plugin =>
catalogue/catalogage/listes_masques/lecteur/unimarc_standard/standard
```

Modifiez le paramètre « page\_ws » pour qu'il pointe vers le web service que vous venez de créer.

Désormais, vous pouvez accéder à cette nouvelle grille de catalogage via l'url  
<http://waterbear.info/bib.php?module=catalogue/catalogage/grilles/lecteur/toto>



Vous pouvez tout à fait définir plusieurs variantes d'un même champ ou sous champ... qui seront utilisées dans des grilles différentes

## Les différents types de sous-champs

Nous avons vu le type « textbox » qui est le plus simple. D'autres types existent :

**Textarea** : comme textbox, mais sur plusieurs lignes (pour les notes et résumés)

**select** : contient une liste de choix. Il faudra alors ajouter un paramètre liste\_choix contenant la liste des choix possibles. Généralement, cela se fait sous la forme d'un plugin inclus comme ceci :

```
+ !!liste_choix
  + nom_plugin => div/get_liste_choix
  + parametres
    + nom_liste => catalogue/catalogage/grilles/lecteur/type_carte
```

On précisera dans « nom\_liste » l'emplacement de la liste par rapport à profiles/default/langues/listes

On pourra également mettre un paramètre « valeur\_defaut » correspondant à l'élément de la liste qui doit être sélectionné par défaut.

**Autocomplete** : Il s'agit des champs avec auto-complétion (des suggestions apparaissent au fur et à mesure que l'on saisit). Ce type de sous-champ est plus complexe.

Il faut modifier le paramètre « evenements » et le faire pointer vers profiles/default/plugins/plugins/catalogue/catalogage/definitions\_groupes\_evenement\_s/biblio/unimarc\_standard/autocomplete qui prend en compte des événements spécifiques à l'auto-complétion.

Il faut également un champ ws\_url indiquant le nom du web-service qui fournira les suggestions (voici cette rubrique).

Le paramètre « forceSelection » : s'il vaut « true » on est obligé de choisir un élément de la liste. S'il vaut « false » on peut saisir un élément non proposé.

Il faudra enfin modifier les switchers (voir cette rubrique) pour adapter le comportement du sous-champs en fonction des besoins du paramétrage.

## Autres paramètres notables des sous-champs

- bool\_garde\_ss\_champ\_vide : si vaut 1, le sous-champ ne sera pas effacé au moment de l'enregistrement, même s'il est vide
- readonly : si vaut « readonly » le sous-champ ne peut être modifié.

## Les switchers

Les switchers décrivent quelles actions accomplir en fonction du comportement de l'utilisateur : il clique sur telle icône, il modifie tel champ/sous-champ, il sélectionne tel élément...

Les switchers sont définis dans le registre à l'emplacement `profiles/default/plugins/plugins/catalogue/catalogage/switchers`

Chaque grille de catalogage est associée à un switcher (il est donc possible de les dupliquer pour avoir des comportements différents selon les grilles). Cette association se fait dans le paramètre « `plugin_switcher` » de la page du web service (cf plus haut)

Un plugin de switcher va associer un certain événement survenu dans un certain sous-champ/champ/onglet à un plugin précis. Voici un exemple :

```
+ liste_champs
  + 999
    + a
      + validation
        + nom_plugin => xxxx
    + delete
      + nom_plugin => yyyy
```

Dans cet exemple, lorsque l'événement « validation » (quand on modifie la valeur d'un sous-champ) survient sur le sous-champ 999\$a, on déclenche le plugin xxxx, tandis que si l'événement « delete » (suppression) survient sur le champ 999, c'est le plugin yyyy qui est appelé.

Le plugin switcher comprend 6 paramètres principaux :

> `Défaut_champ/nom_de_l_evenement` => plugin à utiliser par défaut pour cet événement dans tous les champs (sauf surcharge)

> `Défaut_onglet/nom_de_l_evenement` => plugin à utiliser par défaut pour cet événement dans tous les onglets

> `Défaut_ss_champ/nom_de_l_evenement` => plugin à utiliser par défaut pour cet événement dans tous les ss-champs

> `liste_formulaire/nom_de_l_evenement` => plugin à utiliser pour cet événement dans le formulaire

> `liste_onglet/ID_onglet/nom_de_l_evenement` => plugin à utiliser pour cet événement dans cet onglet

> `liste_champs/nom_champ/nom_de_l_evenement` OU  
`liste_champs/nom_champ/nom_ss_champ/nom_de_l_evenement`  
=> plugin à utiliser pour cet événement dans ce champ ou dans ce ss-champ

Une autre fonctionnalité très utile permet à un switcher « d'hériter » d'un autre. Si par exemple, vous voulez dupliquer un switcher, mais que les modifications à apporter sont minimales, vous n'avez pas forcément envie de recopier tous les paramètres.

Dans ce cas, on peut utiliser le paramètre « `switcher_herite` » :

```
+ unimarc_xs_cd
+ chemin_fichier => catalogue/catalogage/grilles
+ nom_fonction => switcher
+ parametres
  + liste_champs
    + 461
      + 9a
        + wizard_creation_notice
          + nom_plugin =>
catalogue/catalogage/grilles/actions_grilles/wizard_creation_notice
+ switcher_herite
  + nom_plugin => catalogue/catalogage/switchers/biblio/unimarc_xs
```

Dans cet exemple, le switcher `unimarc_xs_cd` hérite du switcher `unimarc_xs`. Tous les paramètres de ce dernier seront réutilisés, sauf lorsque l'événement « `wizard_creation_notice` » survient dans le sous-champ `461$9a`

## Paramétrer les masques

Un masque permet de masquer certains champs / sous-champs inutiles et éventuellement de paramétrer des valeurs par défaut.

Les masques sont regroupés en groupes de masque. On associe ensuite un groupe de masque à une grille donnée.

Les masques sont déclarés à l'endroit suivant :  
profiles/default/plugins/plugins/catalogue/catalogage/masques

```
+ xs
+ chemin_fichier => div
+ nom_fonction => plugins_2_array
+ parametres
  + ??intitule => bib/catalogue/catalogage/grilles/biblio/unimarc_xs/masque_xs
  + champs
    + 010
      + ss_champs
        + b
          + masquer => 1
        + z
          + masquer => 1
    + 100
      + masquer => 1
```

Voici l'exemple du masque XS : il est paramétré pour masquer le champ 100 et les sous-champs 010\$b et 010\$z

Il faut ensuite regrouper les masques en listes de masques à l'emplacement suivant :  
profiles/default/plugins/plugins/catalogue/catalogage/listes\_masques.

Ensuite on va associer une liste de masques à une grille de catalogage. Cela se fait dans les paramètres de la page liée à la grille en question :

- plugin\_get\_masques : indique le plugin à utiliser pour récupérer la liste de masques
- masque\_defaut : le masque par défaut.

## **Paramétrer les accès et les tris**

Dans Waterbear, la structure des données est totalement paramétrable. On a vu dans la rubrique précédente qu'on pouvait modifier les champs / sous-champs composant les différents objets.

Les informations contenues dans les notices peuvent être utilisées pour servir de critère de recherche (on parle alors d'accès) ou bien de critère de tri.

Si par exemple, on crée un nouveau champ 999\$a dans les notices lecteurs pour indiquer la taille du lecteur en centimètres (exemple ridicule, mais soyons fous ;-). On pourra souhaiter par la suite faire une recherche pour trouver tous les lecteurs mesurant telle taille (accès). On pourra également souhaiter, lors d'une recherche de lecteurs, trier ces derniers par taille (tri).

Pour cela, la première étape consiste à déclarer ce nouvel accès ou ce nouveau tri dans la gestion des objets (voir cette rubrique plus haut).

Dans un deuxième temps, il faudra indiquer à Waterbear quelles informations dans la notice doivent alimenter ce nouvel accès. Dans le cas le plus simple, un accès est lié à un seul sous-champ. mais certains accès peuvent être générés par plusieurs sous-champs de plusieurs champs différents (l'accès « auteur » des notices bibliographiques par exemple).

Dans cet exemple simple, nous allons créer un accès « a\_taille » correspondant au champ 999\$a des lecteurs (c'est donc un accès très simple. Nous verrons par la suite comment gérer des accès plus complexes).

Nous allons créer dans le registre le nœud :  
profiles/default/plugins/plugins/catalogue/marcxml/formatage/**lecteur/acces/taille**  
(on distingue dans le chemin le type d'objet (lecteur) et le nom de l'accès (taille)).

Dans ce nœud on va déclarer un plugin comme suit :

```
+ csp
+ chemin_fichier => catalogue/marcxml
+ nom_fonction => get_datafields
+ parametres
  + champs
    + 01 - 999
      + sous-champs
        + 01 - a
          + code => a
      + tag => 999
```

Le script utilisé dans ce plugin est catalogue/marcxml/get\_datafields qui sert à formater des informations contenues dans une notice marcxml (qu'il s'agisse d'une notice bibliographique, d'un exemplaire, d'un lecteur, d'un prêt... dans Waterbear tout marche de la même manière).

En paramètres, on indique quels sont les champs (tag) et sous-champs (code) à afficher. Ici c'est très simple, on n'a qu'un champ (999) et un seul sous-champ (a) et aucun formatage à faire.

Ensuite, il faudra associer cet plugin nouvellement créé au plugin utilisé pour générer tous les accès d'une notice de lecteur :

profils/default/plugins/plugins/catalogue/marcxml/formatage/lecteur/acces/main/default

Ce plugin doit être modifié de la façon suivante (en rouge) :

```
+ default
+ chemin_fichier => catalogue/marcxml
+ nom_fonction => formate_plugins_array
+ parametres
  + plugins
    + a_abos
      + nom_plugin => catalogue/marcxml/formatage/lecteur/acces/abos
    + a_adresse_complete
      + nom_plugin => catalogue/marcxml/formatage/lecteur/acces/adresse_complete
    + a_taille
      + nom_plugin => catalogue/marcxml/formatage/lecteur/acces/taille
    + [...]
```

Noter que le nom du nœud (a\_taille) doit correspondre au nom de l'accès créé dans la gestion des objets.

Pour les tris, le fonctionnement est exactement le même, sauf qu'il faudrait créer le plugin :

profils/default/plugins/plugins/catalogue/marcxml/formatage/**lecteur/tris/taille**

Puis associer ce plugin au plugin qui génère les critères de tri pour les lecteurs à savoir

profils/default/plugins/plugins/catalogue/marcxml/formatage/lecteur/tris/main/default



Cette modification du registre est suffisante pour modifier les accès et les tris des notices qui seront créées ou modifiées à partir de maintenant. Mais elle ne modifie pas les anciennes notices.

Pour cela, il faut utiliser les traitements par lot (voir cette rubrique) et lancer une réindexation complète.

Le script catalogue/marcxml/get\_datafields permet de créer des plugins de formatage extrêmement complexes. Voici en guise d'exemple, une partie du plugin utilisé pour générer l'accès « sujet » pour les notices bibliographiques :

```
+ sujets
+ chemin_fichier => catalogue/marcxml
+ nom_fonction => get_datafields
+ parametres
  + champs
    + 001 - 600
      + apres => |
      + sequentiel => 1
      + sous-champs
        + a
          + avant_verif =>
        + b
          + avant_verif => :
        + x
          + avant_verif => :
        + y
          + avant_verif => :
        + z
          + avant_verif => :
      + tag => 600
    + 002 - 601
      + apres => |
      + sequentiel => 1
      + sous-champs
        + a
          + avant_verif =>
        + b
          + avant_verif => :
        + x
          + avant_verif => :
        + y
          + avant_verif => :
        + z
          + avant_verif => :
      + tag => 601
    + 003 - 602
      + apres => |
      + sequentiel => 1
      + sous-champs
        + a
          + avant_verif =>
        + b
          + avant_verif => :
        + x
          + avant_verif => :
        + y
          + avant_verif => :
        + z
          + avant_verif => :
      + tag => 602
  [...]

```

On voit que le plugin peut gérer le formatage de plusieurs champs (600,601, 602...) sous-champs (a, b, x...) et qu'il peut introduire du texte de formatage avant et après les différents champs/sous-champs.

Ce type de plugin est très courant dans Waterbear : voir en annexe le paramétrage complet des plugins de formatage. Outre get\_datafields, le scripte formate\_plugins est également utilisé pour combiner plusieurs plugins de formatage entre eux.

## Paramétrer les formulaires de recherche et les tris

Une fois les accès et les tris paramétrés, il faut modifier les formulaires de recherche pour pouvoir les utiliser.

Nous allons poursuivre l'exemple précédent en créant un critère de recherche « taille » dans les formulaires de recherche des lecteurs.

Il faut commencer par définir le critère, car il en existe de plusieurs types : pour cela, il faut créer le plugin

profiles/default/plugins/plugins/catalogue/recherches/criteres/lecteur/standard/taille

Là encore, on retrouve dans le chemin du plugin le type d'objet (lecteur) et le nom de l'accès (taille).

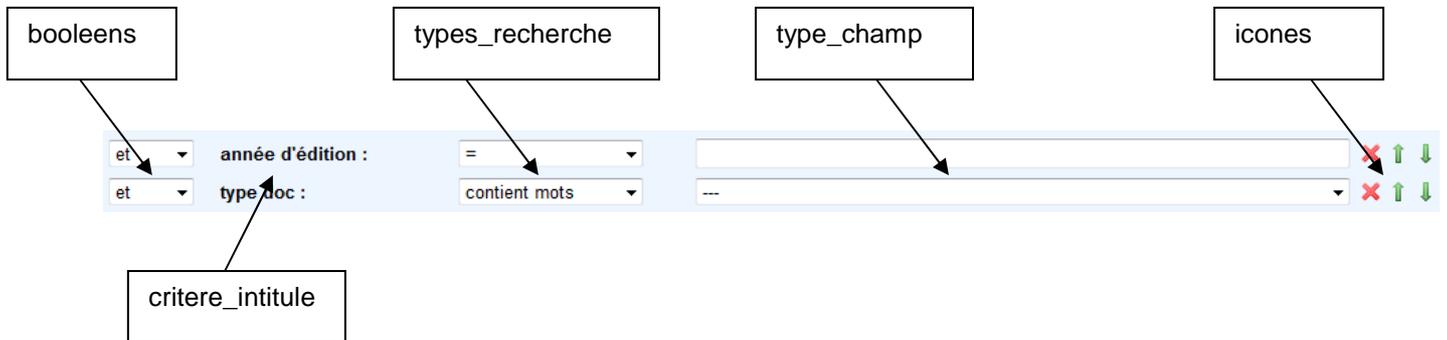
Le plugin aura la forme suivante :

```
+ taille
+ chemin_fichier => div
+ nom_fonction => plugins_2_array
+ parametres
  + !!booleens
    + nom_plugin => catalogue/recherches/booleens/standard
  + !!icônes
    + nom_plugin => catalogue/recherches/icônes_criteres/default
  + !!types_recherches
    + nom_plugin => catalogue/recherches/listes_types_recherches/default
  + ??critere_intitule => bib/catalogue/recherches/criteres/lecteur/standard/taille
  + autoplugin
    + nom_plugin => catalogue/recherches/criteres/lecteur/standard/taille
  + critere => a_taille
  + type_champ => textbox
```

On voit qu'on retrouve beaucoup de choses qui avaient été vues dans le paramétrage des sous-champs de catalogage. La logique est assez similaire. Parmi les paramètres, on retrouve :

- **booleens** : Les critères booléens qui vont précéder le champ de recherche. généralement, c'est toujours les mêmes (et, ou, sauf) mais ça pourrait être modifié.
- **Icones** : les icônes à la fin du champ. Là aussi, c'est le plus souvent les mêmes (monter, descendre, supprimer), mais pas toujours.
- **Types\_recherche** : il s'agit des types de recherches associés à un champ. par exemple « commence par », « contient », « supérieur à »... Ces types de recherche varient en fonction des types de champ : on n'aura pas les mêmes recherches pour un champ de date que pour un champ de texte par exemple.
- **critere\_intitule** : Il s'agit de l'intitulé du critère de recherche. ici par exemple « taille du lecteur ». On peut mettre la valeur directement ici, mais généralement on utilise la syntaxe en « ?? » pour respecter le multilinguisme.

- **Autoplugin** : contient un lien vers le plugin lui-même
- **Critere** : le nom du critère qui a été créé dans les gestion des objets (voir plus haut) : ici « a\_taille »
- **Type\_champ** : « textbox » est le type de champ le plus simple. mais comme pour le catalogage, on trouvera d'autres types comme les listes de choix, les champs avec auto-complétion...



A noter que pour beaucoup de ces paramètres, on utilise des plugins inclus (syntaxe en « !! ») car on réutilise les mêmes paramétrages dans beaucoup de critères de recherche.

Une fois ce plugin créé, il faut l'associer à un ou plusieurs plugins gérant les formulaires de recherche.

Dans Waterbear, on peut créer plusieurs formulaires de recherche différents. Ils sont tous définis dans `profiles/default/plugins/catalogue/recherches/listes_criteres/lecteur`.

Certains formulaires sont directement affichés. D'autres servent à proposer une liste de critères de recherche à ajouter quand on clique sur . Mais leur syntaxe est exactement les mêmes : ils sont interchangeables.

Voici comment faire pour ajouter notre critère de recherche au formulaire « formulaire\_default » qui s'affiche quand on clique sur le recherche lecteur. :

```
+ formulaire_default
+ chemin_fichier => div
+ nom_fonction => plugins_2_array
+ parametres
  + !!01 - cab
    + nom_plugin => catalogue/recherches/criteres/lecteur/standard/cab
  + !!02 - taille
    + nom_plugin => catalogue/recherches/criteres/lecteur/standard/taille
  + [...]
```

**NOTE** : Vous pouvez tout à fait créer un nouveau formulaire au lieu de modifier les formulaires existants. Pour cela il faut dupliquer les plugins de formulaire et faire les modifications nécessaires.

Ensuite vous devez indiquer dans les paramètres de la page quels plugins utiliser pour l'affichage du formulaire : Cela se paramètre dans `profiles/default/pages/bib/catalogue/recherches/formulaires/lecteur`

Modifier les paramètres `plugin_get_formulaire_defaut` (formulaire qui s'affiche par défaut) et `plugin_get_liste_criteres` (liste de critères supplémentaires qui s'affiche quand on clique sur )

Là encore, il vous est possible de modifier la page existante ou de créer une nouvelle page qui en soit une variante (le principe est la même que pour les grilles de catalogage).

### **Autres types de critères de recherche**

L'exemple pris plus haut est le plus simple (textbox). On trouvera également des champs

- `select` (liste de choix) auquel cas, il faut rajouter un paramètre « `liste_choix` » et éventuellement un paramètre « `valeur_defaut` »
- `autocomplete` (avec autocomplétion) : dans ce cas il faut un paramètre « `ws_url` » indiquant l'url du web service qui va vous retourner des propositions.
- D'autres types de recherches sont plus complexes comme la recherche par paniers, le comptage, les jointures... Voir en annexe les paramétrages pour les critères de recherche avancés

### **Paramétrer les tris**

Pour modifier la liste des critères de tri, le principe est très similaire, mais plus simple :

On commence par créer le plugin du critère de tri :

`profiles/default/plugins/plugins/catalogue/recherches/tris/lecteur/standard/taille`

Mais ici, il n'y a que 2 paramètres : l'intitulé du critère de tri et le nom qui a été paramétré dans la gestion des objets.

Ensuite, on associe ce plugin à une liste de critères de tri. Par exemple

`profiles/default/plugins/plugins/catalogue/recherches/listes_tris/lecteur/standard`

Enfin, si nécessaire on peut associer cette liste de critères de tri à une page en particulier via le paramètre `plugin_get_liste_tris`.

## Paramétrer les barres d'icônes et de menus

Chaque page de Waterbear comporte une double barre de menus et une double barre d'icônes. Elles sont doubles, car divisées en deux parties en leur milieu.

Ces barres sont totalement paramétrables. A la fois pour chaque page, mais éventuellement pour des utilisateurs différents sur des postes différents (voir la rubrique personnalisation).

Le détail de chaque barre se trouve dans le registre, dans le nœud « `_parametres` » de chaque page.

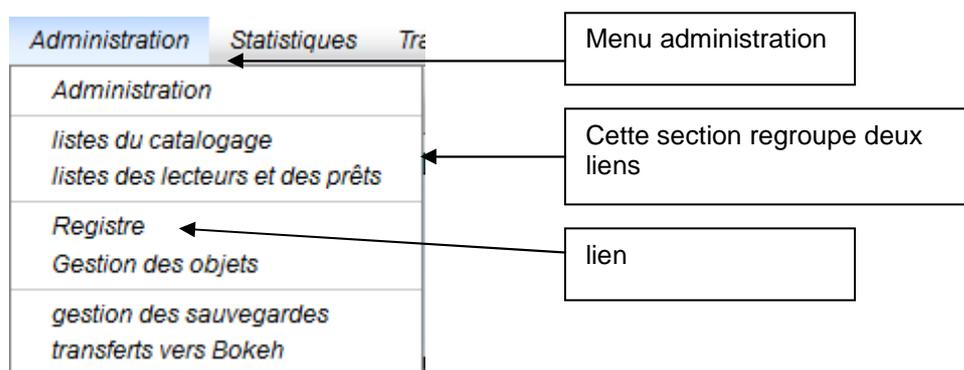
Nous avons vu que Waterbear utilisait un mécanisme appelé héritage pour la gestion des pages (reportez-vous à cette rubrique). Cela veut-dire qu'il n'est pas nécessaire de paramétrer plusieurs fois les barres dans des pages où elles doivent être identiques. On les paramètre au niveau du nœud parent, et ensuite, on ne définit au niveau des enfants que les barres qui sont différentes.

Les barres ont pour nom respectivement `barre_icones1`, `barre_icones2`, `barre_menus1` et `barre_menus2`.

Voici un modèle de paramétrage pour une barre de menus :

```
+ barre_menus1
+ 01 - rechercher
+   chemin_langue => bib/menus/catalogue
+   sections
+   01 - docs
+     menus
+     01 - biblio simple
+       titre => rechercher_biblio_simple
+       url => bib.php?module=catalogue/recherches/formulaires/biblio/simple
+     02- biblio moyen
+       titre => rechercher_biblio
+       url => bib.php?module=catalogue/recherches/formulaires/biblio/moyen
```

Une barre de menu va comporter plusieurs menus. Pour chaque menu on aura une ou plusieurs sections, et dans chaque section, un ou plusieurs liens



Le nœud chemin\_langue indique à quel endroit, par rapport à profiles/defaut/langues sont paramétrés les intitulés des menus (multilinguisme). Que ce soit au niveau des menus ou des liens, on trouve des nœuds « titre » qui indiquent quel intitulé doit être utilisé.

Le nœud « url » indique quelle page afficher quand on clique sur le lien. Pour afficher une page paramétrée dans le registre, on mettra « bib.php ?module=xxx/xxx/xxx » où module indique le chemin de la page dans le module.

Il peut également contenir un appel à une fonction javascript : par exemple

```
js/formulator.ouvrir_page_special('bib.php?module=catalogue/recherches/formulaires/pret/acces&ID_biblio=')
```

Apelé à partir de la page de catalogage permet d'afficher l'historique des prêts d'une notice.

Les barres d'icônes fonctionnent à peu près de la même manière :

```
+ barre_icones2
+ chemin_langue => bib/menus/catalogue/catalogage/grilles
+ icones
  + 00 - ajouter champ
    + img_alt => ajouter_champ
    + img_url => IMG/icones/add.png
    + url => js/formulator.get_liste_champs()
  + 01- voir notice
    + img_alt => voir_notice
    + img_url => IMG/icones/eye.png
    + url => js/formulator.afficher_notice()
  + 03 - enregistrer
    + img_alt => sauvegarder
    + img_url => IMG/icones/disk.png
    + url => js/formulator.enregistrer();
  + 04 - supprimer
    + img_alt => supprimer
    + img_url => IMG/icones/cross.png
    + url => js/formulator.supprimer();
  + 05 - changer masque
    + img_alt => affiche_liste_masques
    + img_url => IMG/icones/plugin.png
    + url => js/formulator.affiche_liste_masques()
+ nb_col => 30
```

Avec les icônes, il n'y a pas de notion de sections, et il faut indiquer pour chaque icône l'url du fichier image correspondant.

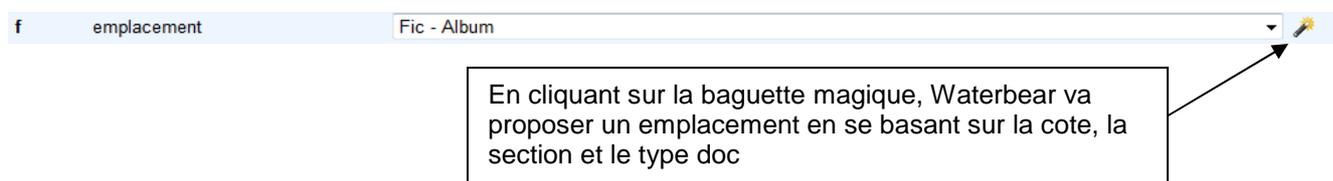
## Paramétrer le wizard emplacements

Le sous-champ 997\$f des notices bibliographiques (correspondant au sous-champ 110\$d de la notice exemplaire) permet d'indiquer la notion d'emplacement.

Dans Waterbear, l'emplacement est simplement un code servant par la suite à faciliter les recherches ou les statistiques. Exemples d'emplacements : albums, policiers, sciences...

Vous pouvez naturellement paramétrer la liste des emplacements comme vous le désirez (via les gestion des listes du catalogue).

Par ailleurs Waterbear dispose d'un wizard qui permet de « deviner » l'emplacement d'un document à partir d'autre informations de la notice comme le type doc, la section et surtout la cote.



Ce Wizard, naturellement, est paramétrable : il se trouve dans le registre à l'emplacement :  
profiles/default/plugins/plugins/catalogue/marcxml/wizard\_emplacement/emplacement\_standard

Les paramètres du wizard ont la forme d'un arbre avec en premier lieu les différents types docs. Pour chaque type doc on aura la liste des sections et enfin pour chaque section la liste des différents débuts de cotes possibles. Cela représente un grand nombre de possibilités, et à chaque possibilité on associera un code emplacement (tels qu'ils auront été définis dans la gestion des listes).

Comme cela peut être très fastidieux et que beaucoup de combinaisons ne seront pas utiles, Waterbear utilise la balise « \_else » pour signifier « dans tous les autres cas ». Voici un exemple :

```
+ emplacement_standard
+ chemin_fichier => catalogue/marcxml
+ nom_fonction => wizard_emplacement
+ parametres
  + choix
    + _else
      + _else =>
        + 0 => d000
        + 1 => d100
        + 2 => d200
    + CD
      + AdulTE
        + 0 => toto
```

Dans l'exemple précédent, on crée une règle pour les CD adultes dont la cote commence par 0 (on leur associe le code emplacement « toto »).

Pour tous les autres types docs (premier « \_else » et quelle que soit la section (2<sup>e</sup> « \_else) on regardera la cote : si elle commence par 0 on aura l'emplacement « d000 », si elle commence par 1 : « d100 » et si elle commence par 2 « d200 ».

**Remarque 1** : pour le 3<sup>e</sup> critère, la cote, la reconnaissance se fait avec troncature à droite : c'est bien ce qui commence par 1, 2, 3... (donc si on a une cote « 305.21 DUR », ça commence bien par « 3 » donc le code emplacement sera « d200 »

**Remarque 2** : Il est possible d'avoir des cotes qui commencent par les mêmes caractères. Par exemple « P » pour les policiers et « POE » pour la poésie. Ou encore « 9 » pour l'histoire et « 91 » pour la géographique. Waterbear fera correspondre en priorité les chaînes les plus longues. Donc « POE » sera prioritaire sur « P » et « 91 » sur « 9 ».

**Remarque 3** : Par défaut, les critères utilisés par le wizard sont le type doc, la section et la cote, mais on peut paramétrer Waterbear pour qu'il utilise d'autres informations de la notice.

## ***Le paramétrage des quotas du prêt***

L'administration simplifiée de Waterbear vous permet de paramétrer les quotas (nombre maximum et durée des prêts). Mais pour que cette interface d'administration conserve sa simplicité, les quotas y sont volontairement simplifiés. Si vous souhaitez utiliser toute la richesse permise par Waterbear, il faut aller directement dans le registre.

Le paramétrage des quotas de prêt se fait à l'endroit suivant du registre :  
profiles/default/plugins/plugins/transactions/prets/parametrages/quotas/par\_type\_doc

Comme d'habitude dans Waterbear, vous pouvez tout à fait dupliquer ce plugin si vous le souhaitez.

Le principe de ce plugin est le suivant : on détermine dans le sous-nœud « criteres » quelles informations doivent être utilisées pour déterminer les quotas (ça peut être des infos de l'exemplaire (type docs, section...) du lecteur ou même de la notice bibliographique.

Voici un exemple de paramétrage où l'on souhaite se baser sur le type doc et la section :

```
+ criteres
+ 01 - type_doc
    + emplacement => infos_exemplaire/a_type_doc
+ 02 - section
    + emplacement => infos_exemplaire/a_section
```

Le nom des critères utilisés correspond aux colonnes paramétrées dans la gestion des objets. On les préfixe de « infos\_exemplaire/ », « infos\_biblio / » ou « infos\_lecteur/ » selon qu'il s'agit d'informations liées à l'exemplaire, la notice biblio ou le lecteur.

Une fois les critères déterminés, il faudra, pour chaque code quota existant, définir un arbre indiquant un nombre maximum de prêts et une durée pour chaque critère. Cela se fait dans le sous-nœud « arbres ».

Dans l'exemple suivant, on définit deux quotas : adulte (A) et jeunesse (J): le quota jeunesse ne peut pas emprunter de livres adultes :

```

+ arbres
  + A
    + _max => 6
    + _label => TOTAL
    + LIV
      + _default
        + _label => Livres
        + _max => 5
        + _duree => 21
    + CD
      + _default
        + _label => CD
        + _max => 3
        + _duree => 14
  + J
    + _max => 6
    + _label => TOTAL
    + LIV
      + _jeunesse
        + _label => Livres jeunesse
        + _max => 5
        + _duree => 21
    + CD
      + _jeunesse
        + _label => CD jeunesse
        + _max => 3
        + _duree => 14

```

Comme on le voit, à chaque niveau de l'arborescence (ici il y en a 3 : le code quota, le type doc, la section) on peut définir un nœud « `_max` » : nombre maximum de documents empruntables : ici on peut emprunter au total 6 documents au total dont 5 livres et 3 CD.

Le nœud « `_duree` » permet de déterminer la durée du prêt tandis que « `_label` » correspond aux intitulés du compteur de prêts en bas de l'écran quand on fait du prêt.

**Remarque 1** : le nœud « `_default` » permet un paramétrage par défaut à chaque niveau de l'arborescence. Par exemple, pour le quota adultes (A), on ne paramètre pas de quotas pour les docs adultes et jeunesse (ce serait redondant). On paramètre juste un quota par défaut (quelle que soit la section).

**Remarque 2** : si vous créez de nouveaux quotas, n'oubliez pas de les déclarer dans la liste `profiles/default/langues/listes/transactions/prets/quotas/default` pour qu'ils apparaissent bien lors de l'inscription des lecteurs.

## ***Le paramétrage des quotas de réservations***

Le paramétrage des quotas de réservation fonctionne exactement comme pour le prêt. Cela se fait à l'emplacement suivant :  
profiles/defaut/plugins/plugins/transactions/prets/parametrages/quotas\_resas/standard

Avec là encore la possibilité de dupliquer le plugin.

La seule différence concerne le nœud « criteres », car dans ce plugin, les informations doivent être tantôt récupérées dans la notice de l'exemplaire, tandis que dans celle de la réservation.

Pour cette raison, on n'a pas un nœud « criteres » comme avec le prêt, mais un nœud « criteres\_exe » et « criteres\_resa » qui récupèrent les mêmes informations, mais l'un dans l'exemplaire l'autre dans la réservation.

Voici un exemple quand on se base uniquement sur le type doc :

```
+ criteres_exe
  + 01 - type_doc
    + emplacement => a_type_doc
+ criteres_resa
  + 01 - type_doc
    + emplacement => a_type_doc_exe
```

Par ailleurs, un autre plugin permet de paramétrer le nombre maximum de réservations par document (contrairement à ce plugin qui indique le nombre maximum de réservations par personne). Ce plugin fonctionne selon le même principe

Cela se fait ici :

profiles/defaut/plugins/plugins/transactions/prets/parametrages/quotas\_reservataire/standard

## **Paramétrage des abonnements**

Par défaut, dans sa version en ligne, Waterbear vous propose d'utiliser 5 abonnements. Mais naturellement, vous pouvez en créer autant que nécessaire via le registre à cet emplacement :  
profiles/default/plugins/plugins/transactions/prets/parametrages/abonnements/standard

```
+ parametres
+ A
  + duree => 365
  + prix => 30
+ B
  + duree => 365
  + prix => 10
+ C
  + duree => 365
  + prix => 0
+ D
  + duree => 365
  + prix => 0
+ E
  + duree => 365
  + prix => 0
```

Créez un nœud par abonnement en spécifiant la durée (généralement 365) et le prix.

Vous devrez également modifier la liste des abonnements (avec leurs intitulés) ici :  
profiles/default/langues/listes/transactions/prets/abonnements/default

Dans le même temps, vous pouvez modifier les motifs de paiement (dans la gestion du porte-monnaie) pour pouvoir facilement effectuer un paiement lié à cet abonnement. cela se fait ici :  
profiles/default/plugins/plugins/transactions/prets/parametrages/motifs\_paiement/standard

Et la liste des motifs de paiement (intitulés) se paramètre ici :  
profiles/default/langues/listes/transactions/prets/motifs\_paiement/default

## ***Paramétrage des colonnes du prêt / retour / réservations***

Dans le module de prêt, il est possible de paramétrer les informations qui s'affichent lors du prêt, des retours ou des réservations.

Pour le prêt :

profiles/default/plugins/plugins/transactions/prets/elem/standard/ajoute\_ligne\_pret

Pour les retours :

profiles/default/plugins/plugins/transactions/prets/elem/standard/ajoute\_ligne\_retour

Pour les réservations :

profiles/default/plugins/plugins/transactions/prets/elem/standard/ajoute\_ligne\_resa

Le paramétrage est cependant assez complexe et fait un usage intensif des variables et plugins inclus.

## ***Paramétrer l'affichage des notices***

Dans Waterbear, tous les affichages de notices sont paramétrables, et ce quel que soit le type d'objet (biblio, exemplaire, lecteur...).

Les différents formats d'affichage sont paramétrés dans :  
profiles/default/plugins/plugins/catalogue/marcxml/formatage/XXX/notice

En remplaçant XXX par le type d'objet (biblio, exemplaire...).

Ces formats d'affichage correspondent aussi bien à l'affichage abrégé (sous forme de liste) qu'à l'affichage détaillé. Certains formats peuvent d'ailleurs être utilisés ailleurs que dans l'interface de recherche (par exemple lors du bulletinage...).

Généralement, les formats d'affichage ne font que regrouper des éléments paramétrés dans  
profiles/default/plugins/plugins/catalogue/marcxml/formatage/XXX/elem\_notice

Le formatage utilise les plugins get\_datafields et formate\_plugins (voir la rubrique « paramétrer et les accès et les tris » et voir en annexe le détail de ces 2 plugins).

Vous pouvez soit modifier les formats existants, soit créer vos propres formats en dupliquant les formats existants.

Si vous créez vos formats, il faut ensuite les associer à la liste des formats d'affichage utilisables (que ce soit pour l'affichage en liste ou l'affichage détaillé).

profiles/default/plugins/plugins/catalogue/marcxml/formatage/XXX/listes\_formats\_notice (pour les affichages détaillés)

profiles/default/plugins/plugins/catalogue/marcxml/formatage/XXX/listes\_formats\_liste (pour l'affichage sous forme de liste)

L'association se fait en utilisant des plugins inclus.

## ***Paramétrer les sections et les types docs***

Certains paramétrages sont plus complexes qu'il y paraît. C'est le cas du paramétrage des sections et des types docs. Leur modification dans Waterbear ne pose pas de difficultés, mais il est nécessaire d'effectuer également certains paramétrages dans Bokeh. Il faut également paramétrer la manière dont Waterbear va exporter les informations vers Bokeh.

### 1. Modification des listes dans Waterbear

Pour éviter que les sections et les types de documents ne soient modifiées « à la légère », elles ne sont pas accessibles via la gestion des listes, mais le registre :  
profiles/default/langues/listes/catalogue/catalogage/grilles/exemplaire/section  
profiles/default/langues/listes/catalogue/catalogage/grilles/exemplaire/type\_doc

### 2. Paramétrage de l'export vers Bokeh

L'export entre Waterbear et Bokeh est normalisé (format unimarc iso 27.09 avec la recommandation 995). Pour créer votre propre section ou un nouveau support, il va falloir paramétrer certains plugins pour convertir les codes de moccan-en-ligne en codes de la norme (ou compatibles avec cette norme).

Pour le type doc, il faut aller dans le registre :  
profiles/default/plugins/plugins/catalogue/marcxml/convertir\_code/exemplaire/rec995\_type\_doc/parametres/liste\_codes

et rajouter la conversion entre votre code Waterbear et un code (sur 2 caractères) spécifique à la recommandation 995 :

Voici le paramétrage par défaut :

```
+ liste_codes
  + az => LIV
  + gf => DVD
  + je => CD
  + jz => CD
```

Par exemple, le support DVD correspond au code « gf ».

Vous trouverez plus d'informations sur la normalisation des codes supports de la recommandation 995 dans ce document : [http://www.adbdp.asso.fr/IMG/pdf\\_r995.pdf](http://www.adbdp.asso.fr/IMG/pdf_r995.pdf) (page 6 : \$q).

Si votre type doc ne correspond à rien dans la liste, vous pouvez quand même le créer (toujours sur 2 caractères), mais il faudra le paramétrer dans Bokeh

Pour la section, ça se paramètre ici :

profiles/default/plugins/plugins/catalogue/marcxml/convertir\_code/exemplaire/rec995\_section/parametres/liste\_codes

Voici le paramétrage par défaut :

```
+ liste_codes
  + a => adulte
  + j => jeunesse
```

Là encore vous pouvez le modifier en vous inspirant de la recommandation 995 (cf plus haut même page : \$r). La norme prévoit peu de sections possible dans le \$r, mais rien ne vous empêche d'en créer d'autres (toujours sur un caractère), du moment que vous l'indiquez ensuite dans Bokeh)

### 3. le paramétrage dans Bokeh

Une fois le paramétrage effectué dans Waterbear, vous devez indiquer à Bokeh à quoi correspondent ces codes (souvenez-vous que Waterbear et Bokeh sont deux logiciels différents : il faut donc effectuer le paramétrage en double).

Pour cela, vous devez vous rendre dans Cosmogramme (voir la rubrique « administration de Bokeh).

Pour le type de document , rendez-vous dans le menu « profil des données >> Waterbear »

Indiquez le code créé (de 2 caractères) dans la colonne « zone 995\$r » en face du type doc désiré.

Pour la section, allez dans le menu « sections » : créez votre nouvelle section en vous inspirant des sections existantes. Si par exemple, vous avez créé une section ados avec un code « d », il faudra mettre « 995\$q=d »

**D. ANNEXES**

## Définitions des principaux plugins

### Plugin de formatage catalogue/marcxml/get\_datafields

```
/**
 * 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 [notice] soit directement tvs_marcxml [tvs_marcxml]
soit via [ID_notice] et [type_doc]
 *
 * Le formatage peut se faire soit dans l'ordre des champs/ss-champs fournis dans le registre, soit dans l'ordre des
champs/ss-champs catalogués
 * C'est par exemple utilise pour formater le champ 200 où l'ordre des ss-champs compte.
 * Dans ce cas, on met une clef [sequentiel] = 1 au niveau des champs ou des ss-champs (cf ci-dessous)
 * On spécifie les noms des champs / ss-champs sans fioriture (par exemple [a] et non pas [001 - a])
 * et pas besoin de spécifier [tag] ou [code] (selon le cas)
 *
 * @param array $parametres
 *
 * @param SOIT ["notice"] => notice en Domxml. (inutile si objet tvs_marcxml fourni)
 * @param SOIT ["tvs_marcxml"] => objet tvs_marcxml. Si non fourni, généré à partir de la notice (DomXml)
 * @param SOIT [ID_notice] et [type_doc]
 *
 * @param ["sequentiel"] => si vaut 1, les champs sont pris dans l'ordre de catalogage (attention : mettre le nom
des champs sans fioriture : "200" pas "001 - 200") et pas besoin de mettre [tag]
 * @param ["defaut"] => valeur par défaut pour l'ensemble du plugin
 * @param ["champs"] => liste des champs à extraire
 * @param ["avant|apres"] => chaines de caractères à placer avant ou après l'ensemble du contenu formaté
 * @param ["champs"][XXX]["plugin_inclus"]=> on peut insérer le contenu d'un plugin en lieu et place d'infos
extraites de la notice [texte] plugin_inclus [notice][champ] ([champ] = définition du champ)
 * @param ["champs"][XXX]["tag"]=> 1 des champs
 * @param ["champs"][XXX]["idx"]=> position du champ dans la liste (commence à 1) ou last
 * @param ["champs"][XXX][avant|avant_verif|apres] => chaines de caractères à placer avant, avant (si déjà qqchse
avant) ou après le contenu du champ
 * @param ["champs"][XXX][plugin_formate] => plugin pour formater le contenu du champ généré
[texte]plugin_formate[texte]
```

```

* @param ["champs"][XXX][defaut] => Valeur par défaut à retourner si aucun champ n'est trouvé
* @param ["champs"][XXX][sequentiel] => si vaut 1, les ss-champs seront pris dans l'ordre de catalogage (pour le
champ 200...) (attention : mettre le nom des ss-champs sans fioriture : "a" pas "001 - a") et pas besoin de mettre
[code]
* @param ["champs"][XXX]["sous-champs"]=> sous-champs à extraire pour ce champ
* @param ["champs"][XXX]["sous-champs"][YYY]["plugin_inclus"]=> on peut insérer le contenu d'un plugin en lieu et
place d'infos extraites de la notice [texte] plugin_inclus [notice][sous_champ] ([sous_champ] = définition du sous
champ)
* @param ["champs"][XXX]["sous-champs"][YYY]["code"]=> 1 des sous-champs
* @param ["champs"][XXX]["sous-champs"][YYY]["idx"]=> position du ss-champ dans la liste (comme à 1) ou last
* @param ["champs"][XXX]["sous-champs"][YYY]["valeur"]=> Valeur requise pour un sous-champ
* @param ["champs"][XXX]["sous-champs"][YYY][avant|avant_verif|apres] => chaines de caractères à placer avant,
avant (si déjà qqchse avant) ou après le contenu du ss-champ
* @param ["champs"][XXX]["sous-champs"][YYY][plugin_formate] => le contenu du ss-champ peut être formaté par un
plugin [texte] plugin_formate [texte]
* @param ["champs"][XXX]["sous-champs"][YYY][defaut] => valeur par défaut à retourner si aucun ss-champ trouvé
* @param ["champs"][XXX]["sous-champs"][YYY][bool_affiche_vide] => si vaut 1, on pourra formater un ss-champ vide
ou inexistant
*
* @return $retour["resultat"]["texte"] => texte trouvé
*/

```

## Plugin de formatage catalogue/marcxml/formate\_plugins

```

/**
* plugin_catalogue_marcxml_formate_plugins()
*
* Ce plugin permet de formater une liste de plugins
*
* Il appelle successivement les plugins comme ceci :
* [texte]<=([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
*
* @param mixed $parametres
* @param["notice"] => notice XML de base (à passer à tous les autres plugins)
* @param["plugins"][0,1,2...][nom_plugin]
*
* [parametres] // parametres du plugin

```

```

*           [avant]
*           [apres]
*           [avant_verif]
*           [seulement_si_vide] => si vaut 1, on n'appliquera le plugin que si les plugins
précédents n'ont rien apporté (ex. auteur principal : on ne cherche les 701 que si 700 n'a rien donné...)
*           [defaut] valeur par défaut si le plugin ne retourne rien
*
*   pour chaque plugin :
*       $tmp[texte] = plugin([notice])
*
* @return [texte] => le texte formaté
*/

```

## Plugin de création d'objet marcxml catalogue/marcxml/crea\_marcxml

```

/**
* plugin_catalogue_marcxml_crea_marcxml()
*
* Ce plugin crée un nouvel objet marcxml selon une définition
*
* Option : pour un champ de lien explicite, on pourra régénérer le champ à partir de la notice liée
* Pour cela fournir un [plugin_get_lien_explicite] au niveau du champ
* Ce plugin doit avoir en [nom_plugin] => catalogue/marcxml/db/get_lien_explicite
* et en paramètre
* ----- [type] => type de la notice
* ----- [ID] ou [notice] ID de la notice ou notice elle-même en XML
* ----- [plugin_formate] le plugin va récupérer et formater les infos dans la notice
*
* @param mixed $parametres
* @param [definition][0,1,...] => la liste des champs
* @param -----[tag] => nom du champ (200, 700...)
* @param -----[definition][0,1,...][code|valeur] => liste des ss-champs avec code (a,b,c...) et
valeur
* @param -----[plugin_get_lien_explicite] => va générer le champ à partir d'un ID de notice ou une
notice XML de notice liée (lien explicite)
*
* Les ss-champs générés s'ajouteront à ceux déjà
déclarés
*
* @return [notice] => notice xml
*/

```

## Plugin de modification d'objet marcxml catalogue/marcxml/modif\_marcxml

```
/**
 * plugin_catalogue_marcxml_modif_marcxml()
 *
 * Ce plugin permet de modifier une notice marcxml
 * ajouter des champs / ss-champs
 * supprimer des champs / ss-champs
 * modifier des champs / ss-champs
 *
 * selon des paramètres qui sont fournis dans [modifications]
 *
 * Ce plugin permet de gérer les liens explicites. Par exemple, on peut générer un champ 700 à partir de l'ID de
 l'auteur lié + rajouter des ss-champs (code fonction...)
 *
 * @param mixed $parametres
 * @param SOIT [notice] => la notice DomXml
 * @param SOIT [tvs_marcxml]
 * @param [modifications][0,1,2...] => liste des modifications à apporter à la notice
 * @param ----[recherche] => équation de recherche sur les champs et les ss-champs (cf plus bas) : utilisé pour la
 recherche des champs (sauf pour add)
 * @param ----[type_modif_champ] => type de modification sur le champ : add => ajout d'un champ | rename => renommer
 le champ | delete => suppression | reset => on remplace complètement le contenu du champ | update on modifie
 certains ss-champs | add_si_absent => on crée le champ si la recherche ne renvoie rien | add_update => si le champ
 existe on le modifie, sinon on le crée
 * @param ----[def_champ] (attention définition différente pour (add, reset) que pour update) => définition du champ
 à créer (cf plus bas)
 * @param ----[tag] (pour add ou rename) => le tag du champ à créer (add ou rename uniquement)
 * @param ----[plugin_get_lien_explicite] => va générer le champ à partir d'un ID de notice ou une notice XML de
 notice liée (lien explicite)
 *
 * Les ss-champs générés s'ajouteront à ceux déjà déclarés
 *
 * Option : pour un champ de lien explicite, on pourra régénérer le champ à partir de la notice liée
 * Pour cela fournir un [plugin_get_lien_explicite] au niveau du champ
 * Ce plugin devra lui-même avoir en paramètre
 * ----- [type] => type de la notice
 * ----- [ID] ou [notice] ID de la notice ou notice elle-même en XML
 * ----- [plugin_formate] le plugin va récupérer et formater les infos dans la notice
 *
 * On pourra renommer un champ ou un sous-champ avec l'option rename
 * Pour un champ, on fournit[tag]
 * Pour un ss-champ on fournit [nouveau_code] ([code] est utilisé pour la recherche)
 *

```

```

* @note format de [recherche] (cf tvs_marxml::get_champs_liste()): utilisé pour la recherche des champs, sauf pour
add
* @note ["recherche"][XXX][tag|idx|sous-champs]
* @note ["recherche"][XXX]["sous-champs"][YYY]["code|valeur|idx"]=> 1 des sous-champs
*
* @note [def_champ][0,1,2...][type_modif_ss_champ] => update (defaut) | add | delete | add_update (on modifie ou
créé si existe pas) | add_si_absent (on crée le ss-champ s'il n'y en a pas déjà un) | duplicate : on copie le ss-
champ | formate : va modifier le contenu du ss-champ avec le plugin plugin_formate
* @note format de [def_champ] pour add ou reset (cf tvs_marxml::add_champ()) :
* @note [def_champ][0,1,2...][code|valeur|rename]
*
* @note format de [def_champ] pour update :
* @note [def_champ][0,1,2...][code|valeur|idx|nouv_valeur|type_modif_ss_champ|nouv_code|plugin_formate]
*
* @note valeur est la valeur actuelle (pour recherche) tandis que nouv_valeur est la valeur de remplacement
*
* @return [notice] => la notice modifiée
*/

```

## Plugin de recherche catalogue/recherches/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]
*
* @param mixed $parametres
* @param [param_recherche] => les paramètres attendus par recherche_simple::init() (cf la classe recherche_simple)
* @param [count_seulement] => si vaut 1, on ne fait que le count
* @param [somme] => nom de la colonne dont on veut la somme ** option **
* @param [somme_seulement] => si vaut 1, on fait count et sum (mais pas requête)
*
* @return array
* @return [nb_pages] => nb de pages
* @return [nb_notices] => nb de notices
* @return [notices] => les notices (peut être très variable selon format choisi : d'une chaîne SQL à une liste
formatée...)
* @return [somme] => la somme d'une colonne ** option **

```

```
*
*/
```

### Détail du paramètre « param\_recherche »

```
catalogue/recherches/recherche_simple
[param_recherche]
  [type_objet] => type d'objet à rechercher
  [criteres][0,1,2...] => cf plus bas
  [tris][0,1,2...] => les colonnes à utiliser pour trier
  [page] => page à afficher
  [nb_notices_par_page]
  [format_resultat] => str_sql : la requete SQL | donnees : une array des résultat (en tableau) contenant ou non
la notice xml | formate : tableau ou chaîne formatés | liste : ID séparées par des virgules :
utilisable dans une requête de type jointure
  [bool_parse_contenu] => est-ce qu'on parse la notice pour en faire un objet DOMXML (qui sera ajouté dans la
colonne xml)
  [plugin_formate_notice] => plugin à utiliser pour formater chaque notice (si format_resultat == formate)
                           passe en paramètre la ligne (toutes les colonnes + éventuellement la notice xml) dans
le paramètre [ligne]
                           récupère directement [resultat]
  [ligne] => /
  [plugin_formate_liste] => plugin à utiliser pour formater la liste (si format_resultat == formate)
                           si non fourni, on retourne une array
                           passe en paramètre le tableau des notices sous [tableau]
                           recupère directement [resultat]
  [tableau] => /
```

=====

Retour :

```
[nb_pages]
[nb_notices]
[notices] => variable selon les formats choisis : une chaîne sql, un tableau, un chaîne de caractères...
[somme] => la somme d'une colonne (option)
```

=====

```
Paramètre [criteres][0,1,2,3...]
  [booleen] (AND, OR...)
  [type_recherche] (str_commence, str_contient...) : cf plus bas
  [intitule_critere] (a_titre, a_auteur...)
  [valeur_critere] (la chaîne à rechercher)
```

[plugin\_formate\_critere] (un plugin pour formater le critère par ex. transformer "an" en "2013") : signature du plugin : [chaine] => [chaine]

=====

#### Types de recherche

// les types suivants ne placent pas de guillemets autour de la chaine à rechercher

int\_egal  
est\_parmi\_int => in() sans guillemets

// les types suivants placent des guillemets autour de lachaine à rechercher

str\_egal  
inf  
inf\_egal  
sup  
sup\_egal  
is\_null => la recherche sera lancée même si aucun terme n'est saisi  
is\_not\_null => la recherche sera lancée même si aucun terme n'est saisi  
non\_egal

// fulltext

str\_commence => troncature à droite  
str\_contient => contient un mot (forme exacte)  
str\_contient\_commence => contient un mot (qui commence par)  
str\_contient\_last\_commence => contient les mots avec forme exacte sauf pour dernier mot saisi (commence)

// dates spéciales

annee => année égal  
annee\_inf\_egal  
annee\_sup\_egal  
mois => mois de l'année (de 1 à 12)  
jour => jour de la semaine de 1=dimanche à 7=samedi

// spécial (cf plus bas)

panier  
panier\_lien  
jointure

=====

#### Formatage des dates

plugin [chaine] div/util\_dates\_var [chaine]

```
an, an1, an2, an3... ==> 2013, 2012, 2011, 2010...
date, date1, date2, date3... ==> 2013-08-15, 2013-08-14, 2013-08-13, 2013-08-12 ...
date_an, date_an1, date_an2, date_an3... => 2013-08-15, 2012-08-15, 2011-08-15, 2010-08-15...
```

=====

#### Recherche d'un Panier

```
[valeur_critere] => chemin du panier
[intitule_critere] => ID
```

=====

#### Recherche d'un Panier de lien

```
[valeur_critere] => chemin du panier
[intitule_critere] => ID
[type_obj_lien] => type d'objet lié
[type_lien] => **opt** spécifier le type de lien entre les 2 objets : par exemple pour un objet auteur lié à un
objet biblio on pourra spécifier 701, 702 ...
[sens_lien] => implicite ou explicite
```

=====

#### Jointure

```
[type_obj_lien] => type d'objet lié
[type_lien] => **opt** spécifier le type de lien entre les 2 objets : par exemple pour un objet auteur lié à un
objet biblio on pourra spécifier 701, 702 ...
[sens_lien] => implicite ou explicite
[intitule_critere] => ID
[valeur_critere] => la recherche à effectuer (même contenu que param_recherche)
```

ATTENTION le format\_resultat de la sous-requête doit être "liste" (liste d'ID séparés par des virgules réutilisable dans une sub-query)

exemple (on montre ici le contenu d'un des critères de recherche qui est de type jointure) :

```
+ booleen => AND
+ intitule_critere => ID
+ sens_lien => explicite
+ type_lien => 461
+ type_obj_lien => biblio
+ type_recherche => jointure
+ valeur_critere =>
```

```

+ bool_parse_contenu => 0
+ criteres
  + 01
    + ##valeur_criterere => ID_periodique
    + intitule_criterere => ID
    + type_recherche => int_egal
+ format_resultat => liste
+ type_objet => biblio

```

## Plugin utilitaire pour les dates div/util\_dates

```

* plugin_div_util_dates()
*
* Ce plugin permet de faire des opérations sur les dates. 2 types sont possibles :
* Conversion (défaut) ou diff ([operation] = diff)
*
* // Conversion
* Permet de convertir une date (ou date du jour) d'un format dans un autre en lui ajoutant/retranchant
éventuellement du temps
*
* // Diff
* Permet de faire la différence entre 2 timestamps
* La plupart du temps ils devront être fournis par des alias
*
* @param mixed $parametres
* @param [operation] => si vaut "diff" fera une différence entre 2 dates
*
* // Conversion
* @param [format_entree] => format de la date fournie (timestamp, us ou fr)
* @param [format_sortie] => format attendu (timestamp, us ou fr)
* @param [date] => date au format indiqué plus haut. Si nul => date actuelle
* @param [modif] => "plus" ou "moins" [OPTION] => si on veut ajouter/retrancher du temps à la date
* @param [nb_modif] => nombre d'unités de temps à ajouter/retrancher
* @param [unite_modif] => unité de temps à ajouter/retrancher (m, h, j) => sinon ou si vide => secondes
*
* // Diff
* @param [timestamp_diff1] => première date (timestamp)
* @param [timestamp_diff2] => 2e date (timestamp)
* @param [unite_diff] => unité de temps pour le retour (m, h, j) => sinon ou si vide => secondes
*
* @return [date] OU [diff]

```

## Plugin utilitaire pour les chaînes de caractères div/util\_string

```
/**
 * plugin_div_util_string()
 *
 * Ce plugin va effectuer un certain nombre de traitements sur une chaîne de caractères
 *
 * Les fonctions gèrent l'unicode (quand c'est possible)
 *
 * @param mixed $parametres
 * @param [texte] => le texte à transformer
 * @param [traitements][0,1,2...][methode] => substr | strtoupper | strtolower | ucfirst...
 * @param [traitements][0,1,2...][XXXX] => autres arguments variables en fonction de la méthode utilisée
 *
 * @return [texte]
 */
```

## ***Paramétrage avancé des champs de recherche et champs statistiques***

Types de champs de recherche :

TEXTBOX

```
+ parametres
+ !!booleens
  + nom_plugin => catalogue/recherches/booleens/standard
+ !!icomes
  + nom_plugin => catalogue/recherches/icones_criteres/default
+ !!types_recherches
  + nom_plugin => catalogue/recherches/listes_types_recherches/juste_egal
+ ##valeur => GET/ID_exe
+ ??citere_intitule => bib/catalogue/recherches/criteres/pre/standard/id_exe
+ autoplugin
  + nom_plugin => catalogue/recherches/criteres/pre/standard/id_exe
+ critere => a_id_exe
+ type_champ => textbox
```

=====

AUTOCOMPLETE

```
+ parametres
+ !!booleens
  + nom_plugin => catalogue/recherches/booleens/standard
+ !!icomes
  + nom_plugin => catalogue/recherches/icones_criteres/default
+ !!types_recherches
  + nom_plugin => catalogue/recherches/listes_types_recherches/juste_egal
+ ##valeur => GET/ID_biblio
+ ??citere_intitule => bib/catalogue/recherches/criteres/pre/standard/id_biblio
+ autoplugin
  + nom_plugin => catalogue/recherches/criteres/pre/standard/id_biblio
+ critere => a_id_biblio
+ type_champ => autocomplete
+ ws_url => bib_ws.php?module=autocomplete/biblio/standard/vedette&
```

=====

SELECT

```

+ parametres
+ !!booleens
  + nom_plugin => catalogue/recherches/booleens/standard
+ !!icones
  + nom_plugin => catalogue/recherches/icones_criteres/default
+ !!liste_choix
  + nom_plugin => div/get_liste_choix
  + parametres
    + nom_liste => catalogue/catalogage/grilles/exemplaire/bibliotheque
+ !!types_recherches
  + nom_plugin => catalogue/recherches/listes_types_recherches/select
+ ??critere_intitule => bib/catalogue/recherches/criteres/pre/standard/bib_exe
+ autoplugin
  + nom_plugin => catalogue/recherches/criteres/pre/standard/bibliotheque
+ critere => a_bib_exe
+ type_champ => select

```

=====

#### PANIER

```

+ parametres
+ !!booleens
  + nom_plugin => catalogue/recherches/booleens/standard
+ !!types_recherches
  + nom_plugin => catalogue/recherches/listes_types_recherches/panier
+ ??critere_intitule => bib/catalogue/recherches/criteres/biblio/standard/panier
+ autoplugin
  + nom_plugin => catalogue/recherches/criteres/pre/standard/panier
+ critere => ID
+ !!icones
  + nom_plugin => catalogue/recherches/icones_criteres/default
+ type_champ => autocomplete
+ ws_url => bib_ws.php?module=autocomplete/pre/panier/standard&

```

=====

#### PANIER D'AUTRES OBJETS

```

+ parametres
+ !!booleens
  + nom_plugin => catalogue/recherches/booleens/standard
+ ??critere_intitule => bib/catalogue/recherches/criteres/pre/standard/panier_biblio

```

```

+ autoplugin
  + nom_plugin => catalogue/recherches/criteres/pret/standard/panier_biblio
+ critere => ID
+ icones
  + 01 - supprimer
    + action => recherchator.delete_champ(#id#)
    + alt =>
    + src => IMG/icones/cross.png
  + 02 - monter
    + action => recherchator.conteneur_recherche.monte_element(#id#)
    + alt =>
    + src => IMG/icones/arrow_up.png
  + 03 - descendre
    + action => recherchator.conteneur_recherche.descend_element(#id#)
    + alt =>
    + src => IMG/icones/arrow_down.png
  + 04 - ouvrir
    + action => recherchator.ouvrir_lien(#id#,
'bib.php?module=catalogue/recherches/formulaires/biblio/complet')
    + alt =>
    + src => IMG/icones/page_go.png
+ liste_types_liens
  + 01 - bidon
    + intitule => notice biblio
    + valeur => 430
+ sens_lien => explicite
+ type_champ => panier_lien
+ type_obj_lien => biblio
+ ws_url => bib_ws.php?module=autocomplete/biblio/panier/standard&

```

=====

JOINTURE [ici exemple d'une recherche de prêts à partir de l'emplacement de l'exemplaire)

```

+ parametres
+ !!booleens
  + nom_plugin => catalogue/recherches/booleens/standard
+ !!icones
  + nom_plugin => catalogue/recherches/icones_criteres/default
+ !!liste_choix
  + nom_plugin => div/get_liste_choix
  + parametres
    + nom_liste => catalogue/catalogage/grilles/exemplaire/emplacement

```

```

+ !!types_recherches
  + nom_plugin => catalogue/recherches/listes_types_recherches/select
+ ??critere_intitule => bib/catalogue/recherches/criteres/pret/standard/emplacement
+ autoplugin
  + nom_plugin => catalogue/recherches/criteres/pret/standard/emplacement
+ critere => ID
+ schema_jointure -----> remplacera valeur_critere (après qu'on aura injecté dedans valeur_critere et
type_recherche
  + bool_parse_contenu => 0
  + criteres
    + 01
      + intitule_critere => a_emplacement
      + type_recherche => @type_recherche -----> sera remplacé par type_recherche
      + valeur_critere => @valeur_critere -----> sera remplacé par valeur_critere
  + format_resultat => liste -----> important !!!
  + type_objet => exemplaire
+ sens_lien => explicite
+ type_champ => select -----> type_champ peut être textbox, select ou autocomplete : les autres
paramètres peuvent varier en fonction
+ type_obj_lien => exemplaire

```

=====

#### DATE

Ce n'est pas à proprement parler un type spécifique, mais on peut rajouter :

```

+ plugin_formate_critere
  + nom_plugin => div/util_dates_var

```

pour utiliser des raccourcis de recherche comme an, an-1...

```

+ !!types_recherches
  + nom_plugin => catalogue/recherches/listes_types_recherches/date

```

critères de recherche spécifiques dates

=====

TRANCHES PREDEFINIES [ici exemple de recherche de lecteurs ayant des prêts dans tel ou tel panier dynamique]

on utilise un type\_champ select mais avec des types\_recherches comme pour un panier (ou un panier d'autres objets)  
De même les paramètres additionnels sont ceux d'un panier (sens lien...)  
Les choix proposés par le select correspondent en fait à des chemins de paniers qui se trouvent dans  
langues/listes/div/paniers\_waterbear

ils ont la forme :

```
+ prets_par_annee
  + _intitules
    + waterbear|recherches|par_annÃ©es|an
      + _fr => cette annÃ©e
    + waterbear|recherches|par_annÃ©es|an-1
      + _fr => l'annÃ©e derniÃ¨re
```

Ou bien waterbear|recherches|tranche\_age|# (terminé par #) si on veut une ventilation parmi tous les paniers d'un répertoire

Il faut utiliser un plugin\_formate\_critere pour transformer les | en /

```
+ parametres
+ !!booleens
  + nom_plugin => catalogue/recherches/booleens/standard
+ !!icomes
  + nom_plugin => catalogue/recherches/icones_criteres/default
+ !!liste_choix
  + nom_plugin => div/get_liste_choix
  + parametres
    + nom_liste => div/paniers_waterbear/prets_par_annee
+ !!types_recherches
  + nom_plugin => catalogue/recherches/listes_types_recherches/panier_lien
+ ??critere_intitule => bib/catalogue/recherches/criteres/lecteur/standard/prets_annee
+ autoplugin
  + nom_plugin => catalogue/recherches/criteres/lecteur/standard/prets_annee
+ critere => ID
+ plugin_formate_critere
  + alias
    + chaine => texte
  + alias_retour
    + texte => chaine
  + nom_plugin => div/util_string
  + parametres
    + traitements
      + 01 - remplacer pipe par slash
        + a_replacer => |
        + methode => str_replace
        + replace => /
+ sens_lien => implicite
+ type_champ => select
```

```
+ type_obj_lien => pret
```

```
=====
```

#### COMPTAGE

Même principe que la recherche par panier on recherche des objets qui sont dans un panier ou qui sont liés à des objets qui sont dans un panier, mais ici, on peut spécifier le nombre d'objets liés. On peut omettre le nom du panier, dans ce cas il comptera le nombre d'objets liés.

```
+ parametres
+ !!booleens
  + nom_plugin => catalogue/recherches/booleens/standard
+ !!types_recherches
  + nom_plugin => catalogue/recherches/listes_types_recherches/nombre
+ ??critere_intitule => bib/catalogue/recherches/criteres/biblio/standard/nb_exe
+ autoplugin
  + nom_plugin => catalogue/recherches/criteres/biblio/standard/nb_exe
+ critere => ID
+ icones
  + 01 - supprimer
    + action => recherchator.delete_champ(#id#)
    + alt =>
    + src => IMG/icones/cross.png
  + 02 - monter
    + action => recherchator.conteneur_recherche.monte_element(#id#)
    + alt =>
    + src => IMG/icones/arrow_up.png
  + 03 - descendre
    + action => recherchator.conteneur_recherche.descend_element(#id#)
    + alt =>
    + src => IMG/icones/arrow_down.png
  + 04 - ouvrir
    + action => recherchator.ouvrir_lien(#id#,
'bib.php?module=catalogue/recherches/formulaires/exemplaire/standard')
    + alt =>
    + src => IMG/icones/page_go.png
+ liste_types_liens
  + 01 - bidon
    + intitule => exemplaire
    + valeur => 997
+ sens_lien => explicite
+ type_champ => comptage
```

```
+ type_obj_lien => exemplaire
+ ws_url => bib_ws.php?module=autocomplete/exemplaire/panier/standard&
```

```
////////////////////////////////////
////////////////////////////////////
```

Pour lancer une recherche automatiquement :

mettre dans les paramètres de la page (bib) : validation\_auto => 1

Pour passer des critères de recherche via l'url : mettre dans la définition du champ de recherche :

```
##valeur => GET/ID_exe
```

(sera remplacé par \$ID\_exe fourni dans l'URL

La variable GET (correspondant à \$\_GET) est fournie au plugin qui génère le formulaire. ATTENTION si ce plugin utilise des plugins inclus (ce qui est quasiment toujours le cas), il faut transmettre la variable GET aux plugins inclus.

On écrira donc qqchse du type :

```
+ !!04 - id lecteur
+ nom_plugin => catalogue/recherches/criteres/prest/standard/id_lecteur
+ parametres
  + ##GET => GET
```

## ***Comprendre les liens entre objets***

### **Comprendre les liens implicites et explicites**

Exemple un champ 700 dans la notice bibliographique pointant vers une notice Auteur

**Biblio** : 700 \$3 : 123456 -----> **Auteur**

-----> lien explicite

<----- lien implicite

Le lien explicite va de la notice qui possède le champ de lien (ici 700\$3) vers la notice liée. Le lien implicite va dans l'autre sens.

### **1) Créer le formatage**

Le formatage permet de décrire quels sont les champs de la notice liée qui devront être reproduits vers la notice d'origine.

plugins/catalogue/marcxml/formatage/**auteur**/liens\_explicites/**biblio\_700**

### **2) Paramétrer les infos sur le lien explicite**

plugins/catalogue/marcxml/db/param\_liens\_explicites/**biblio**/unimarc\_standard/**700**

-> indiquer le plugin de formatage (cf 1)

-> ss-champs à conserver et ss-champs de jointure

-> autres infos sans pb.

### **3) Ajouter ça à la liste des liens explicites**

plugins/catalogue/marcxml/db/param\_liste\_liens\_explicites/**biblio**/unimarc\_standard

mettre un lien **!!700** vers le plugin paramétré en 2)

### **4) Paramétrer le plugin get liste liens explicites**

A faire une seule fois par objet, en plaçant un lien vers le plugin défini en 3)

plugins/catalogue/marcxml/db/get\_liste\_liens\_explicites/**biblio**/unimarc\_standard

### **5) Paramétrer les liens implicites**

C'est la symétrie des liens explicites. On utilise le même plugin de paramètre du lien (cf 2)

plugins/catalogue/marcxml/db/maj\_liens\_implicites/**auteur**/unimarc\_standard

## **Gestion des \$9a et des \$3**

La plupart du temps, un champ de lien comprend un ss-champ de recherche synthétique 9a et un ss-champ de lien 3

### **1) Sous champ 9a**

+ !!evenements  
+ nom\_plugin => catalogue/catalogage/definitions\_groupes\_evenements/**biblio/unimarc\_standard/autocomplete**  
+ !!icones  
+ nom\_plugin => catalogue/catalogage/definitions\_groupes\_icones/**biblio/unimarc\_standard/ss\_champ\_rien**  
+ ??intitule => bib/catalogue/catalogage/grilles/biblio/unimarc\_standard/ss\_champ\_700\_9a\_description  
+ auto\_plugin => catalogue/catalogage/definitions\_ss\_champs/biblio/unimarc\_standard/700\_9a  
+ forceSelection => true  
+ nom => 9a  
+ type => **autocomplete**  
+ ws\_url => **bib\_ws.php?module=autocomplete/auteur/standard/personne\_physique&**

Les événements sont les mêmes pour tous les autocomplete, on peut donc utiliser la définition de biblio

Pas d'icônes

Type autocomplete

web service de recherche (cf plus bas)

### **2) Sous champ 3**

+ !!evenements  
+ nom\_plugin => catalogue/catalogage/definitions\_groupes\_evenements/**biblio/unimarc\_standard/standard**  
+ !!icones  
+ nom\_plugin => catalogue/catalogage/definitions\_groupes\_icones/**biblio/unimarc\_standard/ss\_champ\_rien**  
+ ??intitule => bib/catalogue/catalogage/grilles/biblio/unimarc\_standard/ss\_champ\_700\_3\_description  
+ auto\_plugin => catalogue/catalogage/definitions\_ss\_champs/biblio/unimarc\_standard/700\_3

- + nom => 3
- + readonly => readonly
- + type => textbox

rien de spécial. Pas d'icônes

### **3) Le champ autocomplete**

cf. tutoriel champ autocomplete

### **4) Le Switcher de validation du \$3**

Quand on met un numéro de notice en \$3, il récupère la notice liée et régénère le champ de lien

- + validation
  - + nom\_plugin => catalogue/catalogage/grilles/actions\_grilles/**validation\_lien\_explicite**
  - + parametres
    - + !!param\_lien\_explicite
    - + nom\_plugin => catalogue/marcxml/db/param\_liens\_explicites/**biblio/unimarc\_standard/700**

On utilise le plugin validation\_lien\_explicite en lui fournissant en paramètre le plugin de paramétrage du lien

### **5) Le Switcher de validation du \$9a**

Quand on valide le ss-champ de recherche, il met juste le numéro de notice dans le ss-champ de lien (généralement \$3)

On utilise le plugin validation\_ss\_champ\_synthétique en lui fournissant le nom du ss-champ de lien

- + validation
  - + nom\_plugin => catalogue/catalogage/grilles/actions\_grilles/**validation\_ss\_champ\_synthetique**
  - + parametres
    - + nom\_ss\_champ\_lien => **3**

### **6) Le Switcher wizard creation notice du \$9a**

**\*\* option \*\*** on peut placer un wizard sur le champ 9a de telle sorte que si on ne sélectionne pas une proposition de la liste, il crée automatiquement un nouvel objet lié.

Pour cela, il faudra définir une certaine syntaxe (ponctuation) qui correspondra à des champs/sous-champs

On utilisera le plugin wizard\_creation\_notice en lui fournissant :

- > un plugin pour analyser la chaine et retourner des informations dans le paramètre "variables"
- > un plugin pour créer un objet dans lequel on intégrera les vraibles récupérées précédemment à l'aide d'alias
- > un plugin pour créer l'objet

```

+ wizard_creation_notice
+ nom_plugin => catalogue/catalogage/grilles/actions_grilles/wizard_creation_notice
+ parametres
  + nom_ss_champ_lien => 3
  + plugin_analyse_chaine
    + nom_plugin => catalogue/catalogage/grilles/actions_grilles/wizard_elem_analyse_auteur
  + plugin_crea_objet
    + alias
      + variables|dates => definition/02/definition/03/valeur
      + variables|nom => definition/02/definition/01/valeur
      + variables|prenom => definition/02/definition/02/valeur
    + nom_plugin => catalogue/marcxml/crea_marcxml
    + parametres
      + definition
        + 01
          + definition
            + 01
              + code => a
              + valeur => a
            + tag => label
          + 02
            + definition
              + 01
                + code => a
                + valeur => a_modifier
              + 02
                + code => b
                + valeur => a_modifier
              + 03
                + code => f
                + valeur => a_modifier
            + tag => 200
        + plugin_notice_2_db
          + nom_plugin => catalogue/marcxml/db/notice_2_db/auteur/unimarc_standard

```

## ***Utiliser les champs d'auto-complétion***

### **I Paramétrer l'élément javascript**

Dans chaque élément javascript de type autocomplete, on doit pouvoir indiquer quelquepart (dans le registre) l'URL du Web Service à utiliser (cf point II)

### **II Paramétrer le web service**

A chaque champ autocomplete correspond un WS chargé de retourner les données à afficher dans le champ.  
Ces WS se trouvent :

> pages/bib\_ws/autocomplete/type\_objet/standard/nom\_du\_champ

Le contenu en est de la forme suivante :

+ \_page => bib\_ws/catalogue/catalogage/recherches.php  
+ \_parametres  
  + plugin\_recherche  
    + nom\_plugin => catalogue/catalogage/recherches/ss\_champs\_synthetiques/biblio/unimarc\_standard/700\_9a

La \_page ne change pas, en revanche le plugin\_recherche varie (cf point III)

### **III Paramétrer le plugin de recherche**

Le plugin de recherche se charge de rechercher les infos à afficher dans le champ autocomplete et de les formater (via un plugin spécifique).

Les plugins peuvent se trouver à différents endroits :

> plugins/catalogue/recherches/autocomplete/**recherche\_mv** ou **recherche\_simple**/type\_d\_objet/standard/nom\_champ

Le plugin utilisé sera différent selon que le champ autocomplete se base sur un champ donné ou sur un champ composite (de type tous mots)

### **IV Pour une recherche simple**

Le plugin aura la forme suivante :

- + chemin\_fichier => **catalogue/recherches**
- + nom\_fonction => **recherche\_simple**
- + parametres
  - + param\_recherche
    - + bool\_parse\_contenu => **0** => **sauf si on a besoin de la notice XML pour l'affichage**
  - + criteres
    - + 01 - a\_synthetique
      - + ##valeur\_criteres => **query** => **le texte saisi est envoyé sous le nom query**
      - + intitule\_criteres => **a\_tousmots** => **paramétrable**
      - + type\_recherche => **str\_commence** => **paramétrable**
  - + format\_resultat => **formate**
  - + plugin\_formate\_liste
  - + plugin\_formate\_notice
    - + alias
      - + **ligne** => **tableau** => **si on utilise un plugin non XML**
    - + alias\_retour
      - + **texte** => **/** => **idem**
    - + nom\_plugin => **catalogue/marcxml/formatage/auteur/autocomplete/standard/main (cf VI)**
  - + type\_objet => **auteur** => **paramétrable**

## V Pour une recherche tous mots

Le plugin autocomplete pour les recherches tous mots est plus complexe  
 Il utilise un plugin "recherche simple" classique et un 2e plugin pour re-formater les résultats

- + chemin\_fichier => **catalogue/recherches**
- + nom\_fonction => **recherche\_mv**
- + parametres
  - + nb\_max => **3**
  - + nb\_resultats => **10**
  - + plugin\_formate
    - + nom\_plugin => **catalogue/marcxml/formatage/div/autocomplete\_mv/standard**
  - + plugin\_recherche
    - + alias
      - + **query** => **param\_recherche/criteres/main/valeur\_criteres**
    - + nom\_plugin => **catalogue/recherches/autocomplete/recherche\_simple/biblio/standard/tousmots**

## **VI Formatage recherche simple**

Généralement le plugin se trouve en

plugins/catalogue/marcxml/formatage/type\_objet/autocomplete/standard/main

et a la forme :

+ chemin\_fichier => catalogue/marcxml

+ nom\_fonction => get\_colonnes\_array

+ parametres

+ colonnes

+ 01 - tousmots

+ nom\_champ => nom

+ nom\_colonne => a\_tousmots

+ 02 - ID

+ nom\_champ => id

+ nom\_colonne => ID