



ACube, Framework Ergonomique

Spécification Générale des évolutions de la version 2.6



Version 1.2 du 19/09/2007

Etat : Validé



SUIVI DES MODIFICATIONS

Version	Rédaction	Description	Vérification	Date
1.1	Q DELAPOINTE	Création	X. PRINCE	05/06/07
1.2	X. PRINCE	Mise à jour	X. PRINCE	19/09/07

LISTE DE DIFFUSION

Organisation	Nom	Info	Commentaire	Validation
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



SOMMAIRE

1	OBJECTIFS DU DOCUMENT	4
2	SPECIFICATION FONCTIONNELLE GENERALE.....	5
2.1	Composant Formulaire : Nouvelles fonctionnalités	5
2.1.1	Affichage d'un élément pour lequel le libellé est vide	5
2.1.2	Automatisation de la création des ElementForm.....	5
2.2	Composant Filtre	10
2.2.1	Présentation du composant.....	10
2.2.2	Un composant qui hérite de ElementFormText.....	10
2.2.3	Les différents éléments du composant	10
2.2.4	Fonctionnement du composant.....	10
2.2.5	Attributs et méthodes	12
2.2.6	Autres éléments modifiés	14
2.2.7	Documentation et exemples	15
2.2.8	Le FilterComponent dans la maquette.....	15
2.3	Composant Tableau : Nouvelles fonctionnalités	18
2.3.1	Pager : affichage à droite ou à gauche.....	18
2.3.2	Pager : afficher le nombre total de pages.....	18
2.3.3	Afficher le nombre d'enregistrements d'une page	18
2.3.4	Sauvegarder le paramétrage d'un tableau	19
2.3.5	Dimensionnement automatique des tableaux.....	21
2.3.6	Mise en évidence des colonnes triées.....	21
2.3.7	Correctif sur le filtrage des dates	21
2.4	Composant Arborescence: Nouvelles fonctionnalités	22
2.4.1	Ajout d'une icône pour les documents texte	22
2.4.2	Souplesse du composant.....	22
2.5	XMLObjectCookie: Gestion de la taille	23
2.5.1	Message en cas de création impossible	23
2.5.2	Cookie Multiple.....	23
2.6	Autres évolutions.....	24
2.6.1	Lecture d'un XML sous forme de chaîne de caractères	24
2.6.2	Internationalisation de l'application.....	24
2.6.3	Gestion de l'erreur session invalide en mode popup.....	24
2.6.4	Liens dans la JSDoc	24
2.6.5	Accès au texte d'une balise xml	25
2.7	Exemples supplémentaires implémentés	25



1 OBJECTIFS DU DOCUMENT

Il s'agit dans le cadre de la mise en place des évolutions au sein du Framework Ergonomique ACube de spécifier au travers de ce document le cadre fonctionnel des évolutions mises en place ainsi que leur mode de réalisation au sein du Framework.

Ce document sert donc à la fois de manuel utilisateur à destination des futurs utilisateurs du Framework, mais aussi de référence pour les futures maintenances sur le Framework.

2 SPECIFICATION FONCTIONNELLE GENERALE

Ce chapitre permet de définir dans les grandes lignes les nouvelles fonctionnalités mises en place dans le Framework Ergonomique. Il permet à un utilisateur de prendre rapidement connaissance de ces nouvelles fonctions et de leur contexte d'utilisation.

2.1 COMPOSANT FORMULAIRE : NOUVELLES FONCTIONNALITES

2.1.1 AFFICHAGE D'UN ELEMENT POUR LEQUEL LE LIBELLE EST VIDE

Dans le flux XML, on peut désormais définir un élément de formulaire sans libellé. Pour cela, nous avons modifié le fichier `fw_formulaire_default.js` du Framework : dans la fonction `ecrireBind()` de l'objet `DefaultElementForm`, on appelle systématiquement la fonction `ecrireBoiteHtml()` de l'élément, par contre, la fonction `ecrireLibelleHtml` n'est appelée que dans le cas où le libellé est renseigné :

```
if ((this.ecrireBoiteHtml != null)) {  
    codeHTMLElement += this.ecrireBoiteHtml();  
    if ((this.libelle != "undefined") && this.ecrireLibelleHtml != null)  
        elementlib.innerHTML = this.ecrireLibelleHtml();  
}
```

Un exemple est disponible dans la maquette : dans les onglets accessibles depuis Articles/Consultation, cliquer sur Rechercher, puis sur l'icône pour visualiser l'élément Chemise MAE Bleu d'azur. On arrive alors sur la page associée au flux XML `flux/protected/article/article1.xml` qui ne contient pas de libellé pour la désignation. L'affichage s'effectue normalement.

2.1.2 AUTOMATISATION DE LA CREATION DES ELEMENTFORM

Les flux en entrée :

Le flux XML de création de formulaire comprend maintenant deux parties : le flux de paramétrage et le flux de données.

Le flux de paramétrage, dont la forme correspond à la description faite dans le fichier `dtd/DTD.formulaire.dtd` du framework, définit tous les éléments nécessaires à la construction des éléments de formulaire soit :

- Les éléments concernant le formulaire lui-même : ID, DATA_LISTE (nom de la balise dans le flux de données), METHOD (pour la validation), ONSUBMIT (méthode de soumission de formulaire).
- Les différents éléments de formulaire : ELEMENTFORM_TEXT, ELEMENTFORM_HIDDEN, ELEMENTFORM_PASS, ELEMENTFORM_FILE, COMPOSANT_CALENDRIER, ELEMENTFORM_SELECT, ELEMENTFORM_SELECT_MASTER, ELEMENTFORM_RADIO, ELEMENTFORM_TEXTAREA, ELEMENTFORM_CHECKBOX, ELEMENTFORM_BUTTON_VALIDER, ELEMENTFORM_BUTTON_ANNULER, ELEMENTFORM_BUTTON_LIBRE. Avec pour chacun ID, des propriétés (DIVBIND...) et une balise ZONEDATA qui contient le nom de balise correspondant à l'élément dans le flux des données.

Le flux de données est construit à partir des noms de balises entrés dans DATA_LISTE et ZONEDATA. Il va implémenter toutes les données des éléments de formulaires : texte à entrer dans une zone de saisie, éléments pour créer les `ObjectOptions` d'un `ElementFormSelect` ou d'un `ElementFormRadio`, si la case est cochée ou non pour un `ElementFormCheckBox`, une date pour un `ComposantCalendrier`.

De nouveaux attributs pour l'objet ComposantForm :

- **this.tabOptionElement** : contient les **ObjectOptions** nécessaires à la création d'un élément complexe de formulaire. Initialisé lors de l'appel de la fonction **creerOption**.
- **this.donneeElement** : tableau contenant les noms des balises XML des éléments dans le flux de données (contenu des balises ZONEDATA du flux de paramétrage).
- **this.donneeListe** : nom de la balise racine du flux de données, elle est issue de la balise DATA_LISTE du flux XML de paramétrage.
- **this.XMLObjectInfos** : **XMLObjectSauvegarde** contenant le flux de paramétrage du formulaire. Il est initialisé par l'appel de la fonction **setXMLObjectInfos** dans la méthode **créerFormulaire** (voir ci dessous).
- **this.XMLData** : **XMLObject** contenant le flux des données associé au ComposantForm (contenu des différents éléments de formulaire).
- **this.XMLExport** : **XMLObjectSauvegarde** créé lors de l'appel à la fonction **exportXML** pour sauvegarder les données entrées par l'utilisateur de l'application dans le formulaire. Cette méthode est prévue pour être appelée lors de la soumission du formulaire (redéfinition de la méthode **soumettre()**).

L'exploitation des flux par le Framework :

Afin de créer les éléments du formulaire, le fichier fw_formulaire_composant.js a été complété comme indiqué ci dessous :

Par l'ajout d'une fonction globale **creerFormulaire** qui récupère les données des balises ID, DIVBIND, ACTION, METHOD, ENCTYPE. A partir de ces données, elle appelle le constructeur de **ComposantForm**, puis la méthode de mise à jour de l'élément **XMLFormulaireInfo** : **setXMLObjectInfos**.

Dans le fichier js, après le chargement des flux XML, sont appelées successivement les méthodes **initXMLData**, **importData** et **ecrireBind**.

L'initialisation du flux XML dans **initXMLData** va provoquer ensuite l'appel à la fonction globale **Generique_RetourXML**, qui elle-même appelle **retourImportData** sur le **ComposantForm**. C'est cette méthode qui est alors chargée de traiter les flux afin de créer les **ElementForm** à ajouter au **ComposantForm**, voici les actions effectuées :

Parcours du XML de paramétrage, pour chaque élément de formulaire, parcours des balises des données nécessaires à l'appel du constructeur, puis des autres balises afin d'actualiser les attributs de l'élément. A chaque **ElementForm** correspond une méthode propre appelée dans ce contexte (exemple : **creerElementFormText**). Pour la création du **tabOption** des éléments complexes (**ElementFormSelect** et **ElementFormRadio**), la méthode **instancierDataElt** est appelée sur l'élément Xml contenu dans la balise DATA. Cette méthode permet alors soit de récupérer un fichier XML passé en URL et de le traiter, soit d'appeler la méthode **creerOption** qui parcourt les balises, rassemble les données, crée les **ObjectOption** et remplit le tableau **this.TabOptionElement** qui sert ensuite dans le constructeur de l'élément.

Remarque : la balise DATA peut être renseignée soit dans le flux de paramétrage, elle correspond alors à des valeurs par défaut, soit dans le flux de données, les dernières valeurs seront alors prioritairement affectées à l'**ElementForm**.

Après toutes ces actions, les éléments créés et ajoutés au formulaire s'affichent normalement lors de l'appel à la méthode **ecrireBind** sur la formulaire.

Le flux de sortie :

La méthode **ExportXML** va récupérer les données des **ElementForm** puis les stocker dans l'attribut **this.XMLExport** du **ComposantForm**. Cet attribut de type **XMLObjectSauvegarde** prendra la forme du flux de données que l'on trouve en entrée d'un formulaire.

Exemples dans la maquette :

Un exemple est disponible dans la maquette, dans le menu développement : Articles/Saisie (modification du fichier [menuDeveloppement.xml](#) de la maquette).

La page js est dans [jsclient/article/developpement/saisieDev.js](#). La page html est dans [html/article/developpement/saisieDev.html](#). Les flux XML sont dans [xml/article/developpement/saisieDev.xml](#) pour le paramétrage et [flux/protected/article/developpement/saisieDevData.xml](#) pour les données avec [saisieDevRadioUtilisable.xml](#) qui définit séparément les [ObjectOption](#) d'un [ElementFormRadio](#) à créer.

La page html est de la même forme qu'en une version antérieure.

La page js apparaît très simplifiée, voici les appels principaux de la construction de la page :

```
//Creation du formulaire depuis le flux XML
composantFormulaire=creerFormulaire(XMLInfosFormulaire);
composantFormulaire.initXMLData( "@URL_XML_XmlArticleSaisieDevData@" );
//Import et écriture des données
composantFormulaire.importData();
composantFormulaire.ecrireBind();
```

Voici une partie du flux Xml de paramétrage :

```
<PAGE>
  <ELEMENTFORM>
    <ID>composantFormulaire</ID>
    <DATA_LISTE>LISTE_ELEMENTS</DATA_LISTE>
    <METHOD>POST</METHOD>
    <ELEMENTFORM_TEXT>
      <ID>ref</ID>
      <VALUE>Valeur par défaut</VALUE>
      <LIBELLE>Référence de l'article</LIBELLE>
      <DIVBIND>Position_ref</DIVBIND>
      <DIVLIBBIND>Position_libelle_ref</DIVLIBBIND>
      <SIZE>30</SIZE>
      <MAXLENGTH>30</MAXLENGTH>
      <TABINDEX>1</TABINDEX>
      <ONCHANGE>onChangeFormText</ONCHANGE>
      <OBLIGATOIRE>true</OBLIGATOIRE>
      <ALT>Reference de l'article</ALT>
      <URLAIDE>aide:/Articles/Saisie et Consultation/Rechercher un
article/Par sa référence</URLAIDE>
      <ZONEDATA>REF</ZONEDATA>
    </ELEMENTFORM_TEXT>
    ...
    <ELEMENTFORM_SELECT>
      <ID>unite</ID>
      <DATA>
        <OPTIONS>
          <OPTION>
            <LIBELLE>Valeur par défaut</LIBELLE>
            <VALUE>1</VALUE>
          </OPTION>
        </OPTIONS>
      </DATA>
      <LIBELLE>Unité</LIBELLE>
      <DIVBIND>Position_unite</DIVBIND>
      <DIVLIBBIND>Position_libelle_unite</DIVLIBBIND>
      <TABINDEX>3</TABINDEX>
      <ONCHANGE>onChangeFormSelect</ONCHANGE>
      <SIZE>1</SIZE>
```

```

        <OBLIGATOIRE>true</OBLIGATOIRE>
        <ZONEDATA>UNITE</ZONEDATA>
    </ELEMENTFORM_SELECT>
    <ELEMENTFORM_TEXTAREA>
        <ID>caracteristique</ID>
        <LIBELLE>Caractéristiques</LIBELLE>
        <DIVBIND>Position_caracteristique</DIVBIND>
        <DIVLIBBIND>Position_libelle_caracteristique</DIVLIBBIND>
        <SIZE>10</SIZE>
        <COLS>50</COLS>
        <ROWS>6</ROWS>
        <TABINDEX>4</TABINDEX>
        <ONCHANGE>onChangeFormTextArea</ONCHANGE>
        <OBLIGATOIRE>>false</OBLIGATOIRE>
        <ZONEDATA>CARACTERISTIQUE</ZONEDATA>
    </ELEMENTFORM_TEXTAREA>
    <COMPOSANT_CALENDRIER>
        <ID>disponibilite</ID>
        <LIBELLE>Date de disponibilité</LIBELLE>
        <DIVBIND>Position_disponibilite</DIVBIND>
        <DIVLIBBIND>Position_libelle_disponibilite</DIVLIBBIND>
        <NBANNEES>13</NBANNEES>
        <PASANNEES>2</PASANNEES>
        <FORMATDATE>DD mmm YYYY</FORMATDATE>
        <TABINDEX>5</TABINDEX>
        <ONCHANGE>onChangeFormCalendrier</ONCHANGE>
        <OBLIGATOIRE>>false</OBLIGATOIRE>
        <ZONEDATA>DISPONIBILITE</ZONEDATA>
    </COMPOSANT_CALENDRIER>
    <ELEMENTFORM_BUTTON_LIBRE>
        <ID>btLibre</ID>
        <HREF>javascript:onChangeButtonLibre();</HREF>
        <LIBELLE>Cookie multiple</LIBELLE>
        <DIVBIND>Position_btLibre</DIVBIND>
    </ELEMENTFORM_BUTTON_LIBRE>
    ...
    <ELEMENTFORM_CHECKBOX>
        <ID>imputation</ID>
        <LIBELLE>Imputation budgétaire (auto si PU &lt; 100 Euros)</LIBELLE>
        <DIVBIND>Position_imputation</DIVBIND>
        <DIVLIBBIND>Position_libelle_imputation</DIVLIBBIND>
        <TABINDEX>7</TABINDEX>
        <ONCHANGE>onChangeFormChekbox</ONCHANGE>
        <OBLIGATOIRE>true</OBLIGATOIRE>
        <ZONEDATA>IMPUTATION</ZONEDATA>
    </ELEMENTFORM_CHECKBOX>
    ...
    <ONSUBMIT>soumettre();</ONSUBMIT>
</ELEMENTFORM>
</PAGE>

```

Voici des exemples de données :

```

<LISTE_ELEMENTS>
    <REF>
        <VALUE>3102 C00</VALUE>
    </REF>
    ...
    <UNITE>

```



```

    <DATA>
    <OPTIONS>
      <OPTION>
        <LIBELLE>Boite</LIBELLE>
        <VALUE>1</VALUE>
      </OPTION>
      ...
    </OPTIONS>
  </DATA>
</UNITE>
<CARACTERISTIQUE>
  <VALUE/>
</CARACTERISTIQUE>
<DISPONIBILITE>
  <VALUE>01/07/2007</VALUE>
</DISPONIBILITE>
<PRIX>
  <VALUE>0</VALUE>
</PRIX>
<FORMAT_NB>
  <VALUE>[+ |-(|[0 000][0 000][|)]][0.4+|2|. | EUR|4|1|]</VALUE>
</FORMAT_NB>
<IMPUTATION>
  <VALUE>true</VALUE>
</IMPUTATION>
<UTILISABLE>
  <DATA>
    <URL>@URL_XML_XmlArticleSaisieDevDataRadioUtilisable@</URL>
  </DATA>
</UTILISABLE>
<FORMAT_NUM>
  <VALUE>[+33 ][0 000][ (45 ct\u20AC/min)][]</VALUE>
</FORMAT_NUM>
</LISTE_ELEMENTS>

```

Voici un exemple de flux pour définir les ObjectOption d'un ElementForm complexe (ici, ElementFormRadio):

```

<OPTIONS>
  <OPTION>
    <LIBELLE>Poste</LIBELLE>
    <VALUE>1</VALUE>
  </OPTION>
  <OPTION>
    <LIBELLE>Administration centrale</LIBELLE>
    <VALUE>2</VALUE>
  </OPTION>
  <OPTION>
    <LIBELLE>Les deux</LIBELLE>
    <VALUE>3</VALUE>
  </OPTION>
</OPTIONS>

```

2.2 COMPOSANT FILTRE

2.2.1 PRESENTATION DU COMPOSANT

Le `ComposantForm` peut contenir divers éléments et composants. Le `FilterComponent` ou composant filtre a été créé pour être intégré à un formulaire. L'objectif est de proposer à l'utilisateur de l'application une aide à la saisie à partir de données extraites de la base référentielle. Il propose donc une zone de texte pour saisir des éléments, puis un bouton de recherche qui lance un appel serveur et retourne un flux xml. Ce flux est lui-même intégré à une liste de choix dans laquelle l'utilisateur n'aura plus qu'à cliquer pour sélectionner la donnée qu'il souhaite retenir dans la zone de texte.

En résumé, le `FilterComponent` est donc un composant intégré à un `ComposantForm` pour faciliter la saisie d'un utilisateur.

2.2.2 UN COMPOSANT QUI HERITE DE `ELEMENTFORMTEXT`

La partie principale du `FilterComponent` est un `ElementFormText` de saisie. Afin de pouvoir appliquer les méthodes de cet élément directement au composant, celui-ci en hérite. De même que pour le composant calendrier on peut appliquer les méthodes de l'`ElementFormText`.

Toutefois, afin d'écrire le `FilterComponent` complet, la méthode `EcrireBoiteHtml` de l'`ElementFormText` a été surchargée. Elle récupère la div associée au composant, y ajoute d'autres divs pour les `ElementFormSelect` et le bouton à afficher et crée du code html.

2.2.3 LES DIFFERENTS ELEMENTS DU COMPOSANT

`ElementFormText` : zone de texte pour la saisie de données.

`ElementFormSelect` : liste de choix pour le type de filtrage : =, <, <=, >, >= ou !=. Cette liste de choix n'est affichée que si le flux de paramétrage indique `<FILTER_CHOICE>OUI</FILTER_CHOICE>`.

`ElementFormSelect` : liste de choix pour le résultat de la recherche. Cette liste s'affiche une fois la recherche lancée.

`ElementFormButton` : bouton « rechercher » qui permet de lancer l'appel serveur.

Une image d'attente : celle-ci est indiquée dans le flux xml de paramétrage si l'on souhaite l'afficher (`<WAITING_IMG>@URL_IMG_ImagesGrapheAnim_attente@</WAITING_IMG>`). Elle apparaît à la suite du bouton de recherche quand le flux est en cours de chargement pour indiquer à l'utilisateur que la recherche est bien lancée.

2.2.4 FONCTIONNEMENT DU COMPOSANT

Le `FilterComponent` a besoin d'un flux de paramétrage. Celui-ci est décrit dans la dtd `DTD.FilterComponent.dtd`. Il peut être associé directement au composant par un chargement dans le js et un passage d'objet xml en paramètre du constructeur du `FilterComponent`. Il peut également être directement intégré au flux de paramétrage total du formulaire (décrit dans la dtd `DTD.formulaire.dtd`) et dans ce cas, toute la création et implémentation du composant est gérée de façon automatique.

Voici la dtd décrivant le flux de paramétrage, les éléments prévus dans l'initialisation du composant par appel au constructeur ou à des méthodes héritées de `ElementFormText` ne sont pris en compte que si l'on utilise la génération automatique du formulaire (id, value, libelle, divBind, divLibBind, onChange, onBlur, tabIndex, statut, obligatoire, size, maxlength, alt, altaide, urlaide, altassist, urlassist et title) :

```
<!-- Flux de configuration du filterComponent de formulaire -->
```

```
<!-- Définition des éléments-->
<!ELEMENT ID (#PCDATA)>
<!ELEMENT VALUE (#PCDATA)>
<!ELEMENT LIBELLE (#PCDATA)>
<!ELEMENT DIVLIBBIND (#PCDATA)>
<!ELEMENT DIVBIND (#PCDATA)>
<!ELEMENT ONCHANGE (#PCDATA)>
<!ELEMENT ONBLUR (#PCDATA)>
<!ELEMENT TABINDEX (#PCDATA)>
<!ELEMENT STATUT (#PCDATA)>
<!ELEMENT OBLIGATOIRE (#PCDATA)>
<!ELEMENT SIZE (#PCDATA)>
<!ELEMENT MAXLENGTH (#PCDATA)>
<!ELEMENT ALT (#PCDATA)>
<!ELEMENT ALTAIDE (#PCDATA)>
<!ELEMENT URLAIDE (#PCDATA)>
<!ELEMENT ALTASSIST (#PCDATA)>
<!ELEMENT URLASSIST (#PCDATA)>
<!ELEMENT ZONEDATA (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>

<!-- Définition des éléments propres aux COLONNES -->
<!ELEMENT COLUMN (#PCDATA)><!-- Nom de colonne -->
<!ELEMENT COLUMNS (COLUMN*)>
<!ELEMENT COLUMNS_FILTER (COLUMN*)>
<!ELEMENT COLUMNS_EDIT (COLUMN*)>

<!-- Définition des éléments propres au FILTER_COMPONENT -->
<!ELEMENT URL_XMLDATA (#PCDATA)><!-- Url du flux de données envoyé vers le serveur -->
<!--
<!ELEMENT FLOW_TYPE (#PCDATA)><!-- Type du flux demandé : FULL: un seul appel
serveur et traitement -->
<!-- coté client. Sinon, un appel serveur pour chaque clic sur le bouton du
composant -->
<!ELEMENT TUPLE_NAME (#PCDATA)><!-- Nom du tuple que l'on souhaite retourner -->
<!ELEMENT COLUMN_ID (#PCDATA)><!-- Id de la colonne qui fait l'objet de la
recherche -->
<!ELEMENT FILTER_CHOICE (#PCDATA)><!-- Yes, oui, no, ou non, possibilité de
choisir le type de filtrage à appliquer -->
<!ELEMENT CSS_RESULT_ZONE (#PCDATA)><!-- CSS pour la selectBox de resultat -->
<!ELEMENT CSS_FILTER_ZONE (#PCDATA)><!-- CSS pour l'elementFormText de saisie -->
<!ELEMENT WAITING_IMG (#PCDATA)><!-- On peut renseigner ici une url pour l'image
d'attente lors du chargement du flux -->
<!-- Définition de l'élément de formulaire FILTER_COMPONENT -->
<!ELEMENT FILTER_COMPONENT (ID, URL_XMLDATA, VALUE?, LIBELLE?, DIVBIND?,
DIVLIBBIND?, FLOW_TYPE?, TUPLE_NAME?, COLUMNS?, COLUMN_ID?, COLUMNS_FILTER?,
COLUMNS_EDIT, FILTER_CHOICE?, CSS_RESULT_ZONE?, CSS_FILTER_ZONE?, WAITING_IMG?,
ONCHANGE?, ONBLUR?, SIZE?, MAXLENGTH?, TABINDEX?, STATUT?, OBLIGATOIRE?, TITLE?,
ALT?, ALTAIDE?, URLAIDE?, ALTASSIST?, URLASSIST?, ZONEDATA?)>
```

La balise `COLUMNS` correspond aux colonnes à présenter dans le flux de retour. `COLUMNS_FILTER` correspond aux colonnes concernées par la comparaison avec la valeur pour le filtrage et `COLUMNS_EDIT` aux colonnes à afficher dans la selectBox de résultat. `URL_XMLDATA` précise l'url d'appel serveur. On définit un

flux en complet ou partiel avec la balise [FLOW_TYPE](#) (FULL ou vide). La possibilité de choisir le type de filtrage (=, < etc...) est indiquée dans [FILTER_CHOICE](#). On peut ajouter des CSS et une image d'attente.

Dans tous les cas, il est nécessaire de prévoir une div pour le composant. Celle-ci, pour des questions de présentation doit être intégrée dans un tableau de la manière suivante :

```
<tr>
  <td colspan='2'><table><tr>
    <td style="width:150px"><div id="Position_libelle_myFilterComponent"></div></td>
    <td><div id="Position_myFilterComponent"></div></td>
  </tr></table></td>
</tr>
```

Concernant le type de filtrage, si la balise [FILTER_CHOICE](#) est renseignée à « oui » ou « yes », le flux [xml/filterComponent/filterType.xml](#) va être chargé dans la framework et associé à une liste de sélection. Seront alors affichées les libellés du fichier xml (dans cette version « = », « < », « <= », « > », « >= », « != »).

Le flux de retour :

Les différents éléments vont être comparés à la valeur s'ils étaient dans les [COLUMN_FILTER](#) du flux de paramétrage et seront affichés dans la liste de choix [searchReturn](#) s'ils étaient dans les [COLUMN_EDIT](#) du même flux.

Voici un exemple du flux :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<ELEMENT>
  <MY_TUPLE_NAME>
    <MY_COL_A>Valeur 1</MY_COL_A>
  </MY_TUPLE_NAME>
  <MY_TUPLE_NAME>
    <MY_COL_A>Valeur 2</MY_COL_A>
  </MY_TUPLE_NAME>
</ELEMENT>
```

2.2.5 ATTRIBUTS ET METHODES

2.2.5.1 LES ATTRIBUTS

<private> Boolean [boolFlowFull](#)

Booléen indiquant si on fonctionne en flux complet ou partiel.

<private> Object [ecrireBoiteHtml_parent](#)

La propriété privée «ecrireBoiteHtml_parent()» enrichit la méthode «ecrireBoiteHtml()» de la classe parent "[ElementFormText](#)" avec la génération HTML du composant filtre.

<private> String [id](#)

La propriété privée «id» est l'identifiant du composant.

<private> String [libelle](#)

La propriété privée «libelle» est le libelle du composant texte.

<private> [ElementFormSelect resultSearch](#)

Liste déroulante des résultats de la recherche.

<private> [ElementFormButton searchButton](#)

Bouton pour lancer la recherche.

<private> [ElementFormSelect selectChoiceFilter](#)

Liste déroulante des types de filtre.

<private> String [statut](#)

La propriété privée «statut» correspond au statut du composant.

<private> String [urlWaitingImg](#)

Url de l'image d'attente du composant.

<private> String [urlXMLData](#)

La propriété privée «urlXMLData» est l'url du fichier xml de données du composant.

<private> [XMLObject XMLData](#)

La propriété publique «XMLData» représente les données XML associées au composant Filtre.

<private> [XMLObjectSauvegarde XMLObjectInfos](#)

La propriété privée «XMLObjectInfos» est un objet de type [XMLObjectSauvegarde](#) qui contient le flux XML des paramètres du filtrage.

<private> [XMLObject xmlTypeFilter](#)

xml de données pour le choix de type de filtre.

2.2.5.2 LES MÉTHODES

<private> string [ecrireBoiteHtml\(\)](#)

La méthode privée «[ecrireBoiteHtml\(\)](#)» de la classe parent "[ElementFormText](#)" est enrichie pour intégrer aussi la génération HTML du bouton et des [ElementFormSelect](#).

Object [extend ElementFormText](#)(<String_Obligatoire> id,
<XmlObjectSauvegarde_Facultatif> xmlData, <String_Facultatif> value,
<int_Facultatif> size, <int_Facultatif> maxlength, <String_Facultatif> libelle,
<Boolean_Facultatif> obligatoire, <Boolean_Facultatif> statut, <int_Facultatif>
tabindex)

Crée une instance de la classe [ElementFormText](#)

void [prepareConstructor\(\)](#)

La méthode publique «[prepareConstructor\(\)](#)» Charge le xml de type de filtre si l'utilisateur est autorisé à le sélectionner (paramétrage xml).

```
<private> void search()
```

La méthode privée «search()» est appelée lors d'un clic sur le bouton de recherche. Elle charge le flux XMLData des données de l'ElementFormSelect.

Dans le cas d'un mode de flux complet, le xml est chargé seulement la première fois et on appelle la méthode searchReturn directement les fois suivantes.

Voici les paramètres du fichier xml envoyé au serveur :

-filtre : valeur String de la zone de texte.

-type : valeur 1 pour '=', 2 pour '<', 3 pour '< =', 4 pour '>', 5 pour '> =', 6 pour '!'.

-tupleName : "NomTuple" (Nom du tuple dans le flux de retour)

-cols : "NomColonne1"+"NomColonne2"+

-idCol : "NomColonne1"

-colsFilter : "NomColonne1"+"NomColonne2"+

Exemple :

?filtre=valeur&type=2&tupleName=NOM_TUPLE&cols=NomCol1+NomCol2&idCol=NomCol1&colsFilter=NomCol1+

```
<private> void searchReturn()
```

La méthode privée «searchReturn()» est appelée lors de la réception du flux de recherche. Elle crée et initialise l'ElementFormSelect d'affichage des données.

```
void setStatut\_parent(<String_Obligatoire> statut)
```

La méthode publique «setStatut_parent()» est appelée pour mettre à jour le statut du composant.

```
void setUrlXMLData(<String_Obligatoire> urlXMLData)
```

La méthode publique «setUrlXMLData()» sert à mettre à jour l'attribut urlXMLData.

2.2.6 AUTRES ELEMENTS MODIFIES

Génération automatique

Afin de pouvoir générer le [FilterComponent](#) de façon automatique, une méthode [creerFilterComponent](#) a été ajoutée au fichier [fw_Formulaire_composant.js](#). Celle-ci peut exploiter un flux de paramétrage inclus dans le xml d'entrée du formulaire.

ElementFormSelect

Afin de pouvoir mettre à jour l'[ElementFormText](#) depuis l'action [onChange](#) de l'[ElementFormSelect](#) de résultat [resultSearch](#), un attribut a été ajouté à [ElementFormSelect](#) :

```
<private> DefaultElementForm elementFormToChange
```

Attribut «elementFormToChange» de ElementFormSelect.

Cet attribut et les méthodes [setElementFormToChange](#) et [getElementFormToChange](#) implémentées permettent d'associer à l'[ElementFormSelect](#) l'[ElementFormText](#) à modifier. La méthode [onChange](#) a alors été implémentée sur le [resultSearch](#) afin de mettre à jour la zone de texte associée.

La méthode [getValueLibelle](#) a été ajoutée à l'[ElementFormSelect](#), elle permet de retourner le libellé de la valeur sélectionnée (balise [LIBELLE](#) de fichier Xml et non pas balise [VALUE](#)). Elle est utilisée dans le code du [FilterComponent](#) pour écrire le libellé sélectionné dans la zone de saisie.

2.2.7 DOCUMENTATION ET EXEMPLES

Dans la documentation, on trouve un lien vers la page du [FilterComponent](#). Celle-ci présente les attributs et méthodes, mais également un aperçu du diagramme uml du composant, un lien vers un exemple de constructeur implémenté, et un accès à une popup complète d'exemple.


2.2.8 LE FILTERCOMPONENT DANS LA MAQUETTE

Dans la maquette le [FilterComponent](#) est disponible à deux endroits : dans le menu principal et dans le menu de développement dans la page Article/Saisie. Ce composant se présente sous la forme suivante :



Avant chargement :

Après chargement :



Dans la page [saisie.js](#), une méthode [preparePage](#) a été ajoutée afin de charger le flux xml de paramétrage du [FilterComponent](#). Ce chargement se fait dans la variable globale [XMLFiltre](#) de type XMLObject :

```
if (XMLFiltre == null)
{
    XMLFiltre = new parent.XMLObject(null, null,
"window.frames['CONTENU_WIN'].construireFormulaire",
"@URL_XML_XmlArticleSaisieFiltre@", false);
    //XMLFiltre.setDivBindMessage("Position_myFilterComponent");
    XMLFiltre.initAsynchrone(false);
    XMLFiltre.importXML();
}
else
{
    construireFormulaire();
}
```

La flux de paramétrage comprend les éléments essentiels pour une création manuelle du composant :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?><!DOCTYPE PAGE SYSTEM
"@URL_DTD_DtdFilterComponent@">

<PAGE>
  <FILTER_COMPONENT>
    <ID/>
    <URL_XMLDATA/>
    <VALUE/>
    <LIBELLE/>
    <DIVBIND/>
    <DIVLIBBIND/>
    <FLOW_TYPE>FULL</FLOW_TYPE>
    <TUPLE_NAME>UNITE_DISPO</TUPLE_NAME>
```

```

    <COLUMNS>
      <COLUMN>UNITE</COLUMN>
      <COLUMN>COULEUR</COLUMN>
    </COLUMNS>
    <COLUMN_ID>UNITE_ID</COLUMN_ID>
    <COLUMNS_FILTER>
      <COLUMN>UNITE</COLUMN>
    </COLUMNS_FILTER>
    <COLUMNS_EDIT>
      <COLUMN>UNITE</COLUMN>
      <COLUMN>COULEUR</COLUMN>
    </COLUMNS_EDIT>
    <FILTER_CHOICE>OUI</FILTER_CHOICE>
    <CSS_RESULT_ZONE></CSS_RESULT_ZONE>
    <CSS_FILTER_ZONE></CSS_FILTER_ZONE>
    <WAITING_IMG>@URL_IMG_ImagesGrapheAnim_attente@</WAITING_IMG>
    <ONCHANGE/>
    <ONBLUR/>
    <SIZE>20</SIZE>
    <MAXLENGTH>20</MAXLENGTH>
    <TABINDEX>1</TABINDEX>
    <STATUT>STATUT_ENABLED</STATUT>
    <OBLIGATOIRE>NON</OBLIGATOIRE>
    <TITLE/>
    <ALT/>
    <ALTAIDE/>
    <URLAIDE/>
    <ALTASSIST/>
    <URLASSIST/>
    <ZONEDATA>UNITE_DISPO</ZONEDATA>
  </FILTER_COMPONENT>
</PAGE>

```

Puis dans la méthode `construirePage`, on crée le `XMLObjectSauvegarde` puis l'objet `FilterComponent` que l'on ajoute ensuite au formulaire :

```

//Chargement de données pour le composant filtre
var testErreurFiltre = XMLFiltre.parseErrorXML();
if (testErreurFiltre == 0)
{
  if (XMLInfosFiltre == null)
  {
    XMLInfosFiltre = new parent.XMLObjectSauvegarde();

    XMLInfosFiltre.sauvegardeDOM(parent.getElements(XMLFiltre.xmlDoc, "FILTER_COM
PONENT")[0]);
  }
}
//ComposantFiltre : accès aux unités de 100 feuilles
var myFilterComponent=new
FilterComponent("myFilterComponent",XMLInfosFiltre,"@URL_XML_XmlArticleSaisieFiltr
eData@", "Entrer une valeur", 30, 40, "Unité disponible pour 100 feuilles", false,
null, 20, "Rechercher");
composantFormulaire.addElement(myFilterComponent);

```

Dans la page `saisieDev.js`, aucun code n'est rajouté. Seul le xml de paramétrage du formulaire a été complété : ajout de la balise `FILTER_COMPONENT` et de ses noeuds enfants correspondants en complétant les

libellés nécessaires au constructeur tels l'id, le libellé etc... La balise `ZONEDATA` renverra vers le flux de données du formulaire. Si une balise `DATA` est alors définie, elle complètera l'`ElementFormText` associé au `FilterComponent`.

Dans les deux exemples, le flux bouchonné d'accès serveur est le suivant :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?><!DOCTYPE PAGE SYSTEM
"@URL_DTD_DtdFilterComponentData@">
<ELEMENT>
  <UNITE_DISPO>
    <UNITE>Boite</UNITE>
    <COULEUR>Blanc</COULEUR>
  </UNITE_DISPO>
  <UNITE_DISPO>
    <UNITE>Bloc</UNITE>
    <COULEUR>Blanc</COULEUR>
  </UNITE_DISPO>
  <UNITE_DISPO>
    <UNITE>Rame</UNITE>
    <COULEUR>Blanc</COULEUR>
  </UNITE_DISPO>
  <UNITE_DISPO>
    <UNITE>Rame</UNITE>
    <COULEUR>Beige</COULEUR>
  </UNITE_DISPO>
  <UNITE_DISPO>
    <UNITE>Rame</UNITE>
    <COULEUR>Bleu</COULEUR>
  </UNITE_DISPO>
  <UNITE_DISPO>
    <UNITE>Paquet</UNITE>
    <COULEUR>Blanc</COULEUR>
  </UNITE_DISPO>
</ELEMENT>
```

Ce flux comprend les tuples de la base de données. Les données vont être récupérées, comparées à la valeur de l'`ElementFormText` si elles sont dans les `COLUMNS_FILTER` du flux de paramétrage, affichées dans la `selectBox` si elles sont dans les `COLUMNS_EDIT`. Ici, « `UNITE_DISPO` » correspond au `TUPLE_NAME` du flux de paramétrage.

2.3 COMPOSANT TABLEAU : NOUVELLES FONCTIONNALITES

2.3.1 PAGER : AFFICHAGE A DROITE OU A GAUCHE

Le pager peut désormais être affiché en bas à droite (voir dans Articles/Catalogue) ou en bas à gauche du tableau (voir dans Articles/Catalogue Simplifié).

Pour cela, une balise <SIDE> a été ajoutée à la balise <PAGER> dans DTD du flux XML de paramétrage du tableau. Les valeurs du noeud XML doivent être soit left soit right. Le pager est affiché par défaut à gauche.

Exemple dans catalogue.xml : `<PAGER>`
 `<MAX_LIGNES>20</MAX_LIGNES>`
 `<SIDE>right</SIDE>`
 `</PAGER>`

Pour cet affichage, des images d'arrondis ont été ajoutées, et le fichier navig.css du framework comporte désormais 4 styles au lieu de 2 pour le bon affichage du pager. La gestion se fait dans la méthode `ecrireHTML_Pager` du ComposantTableau dans `js/fw_tableau` du Framework.

2.3.2 PAGER : AFFICHER LE NOMBRE TOTAL DE PAGES

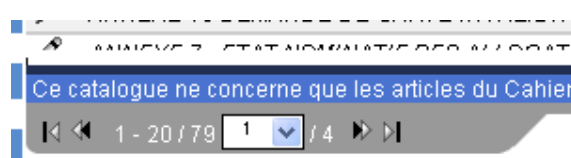
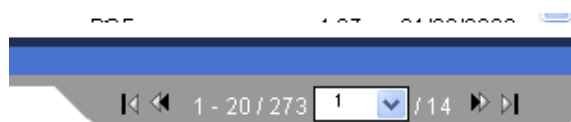
Le nombre total de pages est affiché dans le pager à la suite de la selectBox contenant l'ensemble des numéros de page. La modification a été réalisée dans la méthode `ecrireHTML_Pager` du ComposantTableau dans `js/fw_tableau` du Framework.



2.3.3 AFFICHER LE NOMBRE D'ENREGISTREMENTS D'UNE PAGE

- Dans le cas d'un pager existant

L'affichage se fait devant la boîte de sélection. On affiche les numéros des enregistrements affichés suivi du nombre total d'enregistrements du tableau.



- Dans le cas d'un tableau sans pager

L'affichage se fait dans la zone vide du pager. Il est de la forme « n enregistrements » avec enregistrements qui est défini sous forme d'une variable en français ou en anglais dans le fichier `fw_tableau.js`.



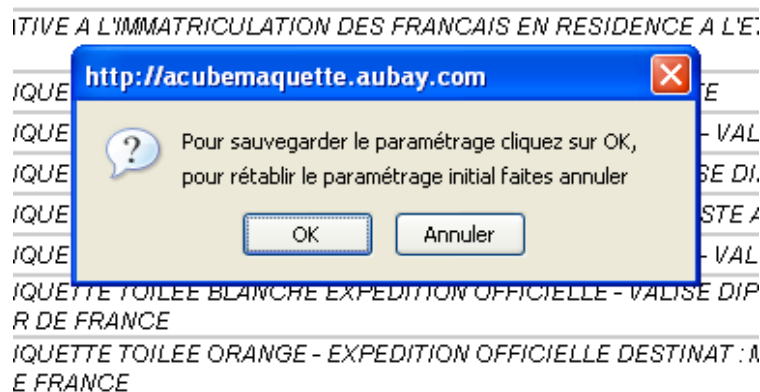
Le style du texte est observable dans tableau.css.

2.3.4 SAUVEGARDER LE PARAMETRAGE D'UN TABLEAU

- Un outil pour la sauvegarde

Pour pouvoir effectuer une sauvegarde, nous avons ajouté un outil dans la DTD du tableau. Dans la balise `<OUTILS>` du flux de paramétrage, il suffit d'ajouter `<ACTION_PARAMETER_SAVE_PRESENT>true</ACTION_PARAMETER_SAVE_PRESENT>` pour afficher l'image correspondante dans la barre d'outils. Cet élément Xml est repris dans la fonction `initInfosOutilsByXML()` du ComposantTableau dans le fichier `js/ergonomique/fw_tableau.js`. Il permet alors d'afficher une icône de sauvegarde qui est elle-même associée à la fonction `manageParameterSetting` du tableau décrite ci-dessous.

Quand on clique sur l'outil de sauvegarde, une boîte de dialogue s'affiche alors permettant soit de sauvegarder effectivement le paramétrage (appel de la méthode `saveParameterSetting`, soit de rétablir le paramétrage initial (suppression du cookie).



- Un objet XMLObjectCookie

Dans le ComposantTableau, la méthode `saveParameterSetting` effectue les opérations suivantes :

- Création de l'objet XMLObjectCookie :

On lui donne pour id par défaut le `this.id` du ComposantTableau. On peut également choisir l'id du cookie comme dans l'exemple de la maquette :

```
ComposantTableauTest = new ComposantTableau("ComposantTableauTest",
XMLInfosTableau, "@URL_XML_XmlArticleListeArticleParCritere@", true,
false);
```

```
ComposantTableauTest.setCookieName("theMagicRecupParametrage");
```

La méthode `setCookieName` a été créée dans le ComposantTableau. Elle met à jour l'attribut `cookieName` du composant puis appelle la méthode `getParameterSetting` qui recherche si le cookie existe pour mettre à jour les colonnes du composant si c'est le cas.

- Initialisation de cet objet avec les données de paramétrage du tableau : on renseigne les colonnes, leur tri éventuel, si elles sont cachées ou non, et leur critère de filtre s'il y a lieu. Ces informations forment en fait un cookie contenant les données sous la forme suivante :

```
<PAGE>
  <TABLEAU>
    <ID_TAB>Catalogue des articles</ID_TAB>
    <COLONNES>
      <COLONNE>
        <DATA>DESIGNATION</DATA>
        <ID_COL></ID_COL>
        <TRI>
          <TYPE>CROISSANT</TYPE>
          <NIVEAU>1</NIVEAU>
        </TRI>
        <CACHE>NON</CACHE>
        <FILTRE></FILTRE>
      </COLONNE>
      <COLONNE>
        <DATA>UNITE</DATA>
        <ID_COL></ID_COL>
        <TRI>
          <TYPE>CROISSANT</TYPE>
          <NIVEAU>2</NIVEAU>
        </TRI>
        <CACHE>NON</CACHE>
        <FILTRE>
          <CRITERE>F</CRITERE>
          <JOKER></JOKER>
          <SIGNE>=</SIGNE>
        </FILTRE>
      </COLONNE>
      <COLONNE>
        <DATA>REFERENCE</DATA>
        <ID_COL></ID_COL>
        <TRI></TRI>
        <CACHE>NON</CACHE>
        <FILTRE></FILTRE>
      </COLONNE>
      <COLONNE>
        <DATA>PU</DATA>
        <ID_COL></ID_COL>
        <TRI></TRI>
        <CACHE>OUI</CACHE>
        <FILTRE></FILTRE>
      </COLONNE>
      <COLONNE>
        <DATA>TARIF</DATA>
        <ID_COL></ID_COL>
        <TRI></TRI>
        <CACHE>NON</CACHE>
        <FILTRE></FILTRE>
      </COLONNE>
    </COLONNES>
    <NUMERO_PAGE>2</NUMERO_PAGE>
  </TABLEAU>
</PAGE>
```

- Lecture du cookie et restauration

Lorsque l'on ouvre la page contenant le tableau, lors de l'initialisation des colonnes (méthode `initTabByXML()` du ComposantTableau dans `fw_tableau.js`), on appelle la méthode de restauration des données de paramétrage : `getParameterSetting()`.

Cette méthode effectue les opérations suivantes :

- Récupère le cookie, soit à partir du nom `this.cookieName` s'il est initialisé, soit à partir de l'id du tableau qui sert de nom par défaut.
 - Elle parcourt le cookie et accède aux informations sauvegardées : les données sur le tri ou non d'une colonne : le type (croissant, décroissant, non) et le niveau (Integer indiquant l'ordre pour le tri dans le cadre du tri multiple); le fait que la colonne est cachée ou pas; les informations sur le filtrage éventuel d'une colonne (critère, signe et joker).
- Elle parcourt l'attribut `tabColonnes` du ComposantTableau et met à jour les `ObjectColonne` à partir des données récupérées précédemment.
 - La gestion des éléments null ou vides est prise en compte.
- Exemple dans la maquette

Un exemple est disponible dans la page Articles/Catalogue de la maquette. On peut alors modifier les tris, les filtrages ou l'affichage des colonnes, puis cliquer sur l'icône de sauvegarde. Si on clique alors à nouveau dans le menu sur le tableau concerné, le paramétrage est conservé.

2.3.5 DIMENSIONNEMENT AUTOMATIQUE DES TABLEAUX

La taille du tableau peut être paramétrée à la valeur `AUTO` plutôt qu'un nombre de pixels ou un pourcentage de la taille de l'écran. La taille du tableau est alors calculée automatiquement en fonction de la taille spécifiée pour chacune des colonnes qui le constituent. Si au moins une des colonnes ne spécifie pas de taille, le tableau a alors comme taille 100%.

Un exemple est disponible dans la page Articles/Catalogue de la maquette.

2.3.6 MISE EN EVIDENCE DES COLONNES TRIÉES

Le style des colonnes triées est paramétrable dans les css. Les classes concernées sont accessibles dans le framework : `css/commun/tableau.css`, classes `tableau_tri_croissant` et `tableau_tri_decroissant`. Aujourd'hui, le style met en italique les colonnes triées, on peut facilement voir un exemple dans un tableau trié de la maquette (comme Articles/Catalogue).

L'ajout des classes de style CSS aux lignes du tableau se fait alors dans la méthode `ecrireHTML_Ligne` du ComposantTableau.

2.3.7 CORRECTIF SUR LE FILTRAGE DES DATES

L'égalité des Objets dates javascript ne fonctionnant pas, le filtrage sur une date précise ne retournait aucune valeur dans un tableau. Un test a été ajouté dans la fonction `testFiltre` du tableau pour que dans le cas d'une date, si le signe demandé est « = » on teste les dates non supérieures et non inférieures.

2.4 COMPOSANT ARBORESCENCE: NOUVELLES FONCTIONNALITES

2.4.1 AJOUT D'UNE ICONE POUR LES DOCUMENTS TEXTE

L'icône affichée dans une arborescence pour un document texte n'existait pas dans les images du Framework. Une image par défaut a donc été copiée afin d'obtenir dans images/bouton/arbo les fichiers IcoFichierTxt.gif et IcoFichierTxtover.gif.

Un exemple est accessible dans la maquette : dans Postes/Fiche de poste, dans Maroc, la deuxième ville (modification du fichier xml des postes et ajout d'un fichier texte Marrakech.txt dans divers/).

2.4.2 SOUPLESSE DU COMPOSANT

On peut désormais afficher une icône sans libellé en renseignant les balises IMAGE, IMAGEOVER, IMAGEEXPANDER et IMAGEEXPANDEDOVER du fichier XML de données. L'exemple de la deuxième ville du Maroc illustre cette possibilité : il ne présente pas de libellé. La modification du code a consisté à ajouter un test dans la fonction `ecrireBindRecurcif` du fichier `jsclient/ergonomique/fw_arborescence.js`

Il a été vérifié que les images déclarées dans les balises seront affichées en priorité par rapport à celles définies par défaut pour un type de fichier.

2.5 XMLOBJECTCOOKIE: GESTION DE LA TAILLE

2.5.1 MESSAGE EN CAS DE CREATION IMPOSSIBLE

Dans la fonction `saveCookieXML` de l'objet `XMLObject`, qui instancie les données dans le cookie, un test a été ajouté afin de déterminer si le cookie contient bien les données. Si ce n'est pas le cas, un message d'alert est affiché. Ce dernier utilise des constantes définies sous les noms de `ERR_CLI_ALERT_004_FR`, `ERR_CLI_ALERT_004_EN` et `ERR_CLI_ALERT_004_ES` dans la documentation.

2.5.2 COOKIE MULTIPLE

En cas de taille de données à stocker trop importante pour un seul cookie, il est possible d'activer l'option cookie multiple qui permet de stocker cette donnée au sein de plusieurs cookies qui porteront comme nom le nom définit suivi d'un indice de 0 à n.

Pour activer cette option il suffit de valoriser à true le paramètre « `cookieMultiXML` » (paramètre n°8 du constructeur) lors de la création de l'objet `XMLObjectCookie`. Lors de la création des cookies multiples (appel de la fonction `saveCookieXML` on contrôle :

- la bonne création des cookies (cf. cf-dessus),
- que le nombre de cookie ne dépasse par la valeur maximum (cf. variable « `cookieXMLMaxNumber` »),
- que la taille total de la donnée à stockée ne dépasse pas la taille maximum (cf. variable « `cookieXMLMaxSize` »).

Un exemple de cookie multiple est implémenté lors du clic sur le bouton « Cookie Multiple » du formulaire automatique de saisie dans le menu développement : Articles/Saisie.

2.6 AUTRES EVOLUTIONS

2.6.1 LECTURE D'UN XML SOUS FORME DE CHAÎNE DE CARACTÈRES

Afin de pouvoir lire un fichier XML sous forme de String, la fonction globale `convertStringToDOM(chaineXML)` a été ajoutée dans le fichier `fw_xwl.js`. Elle récupère la chaîne passée en paramètre et la convertit en objet DOM. Elle retourne un `DOMElement` correspondant au flux XML sérialisé.

Ainsi la méthode de chargement de flux `XMLObjectCookie_loadXML` appelle maintenant directement la fonction `convertStringToDOM` pour tester le flux et éventuellement le sérialiser.

Un exemple est disponible dans la maquette dans la page des Cahiers Navette/Affectation : l'objet `DomElement` est converti en String puis reconverti en objet `DomElement` :

Dans `affectation.js` :

```
// Test des fonctions convertStringToDOM et convertDOMtoString
var chaineXML = fwXml.convertDOMtoString(XMLPage.xmlDoc);
XMLPage.xmlDoc = fwXml.convertStringToDOM(chaineXML);
```

2.6.2 INTERNATIONALISATION DE L'APPLICATION

Dans le cas où un code langue n'existe pas, on le met par défaut en anglais. Dans la méthode `getLibelle` du fichier `jsclient/technique/fw_outils.js`, on recherche la langue stockée dans le cookie (par défaut, celle du navigateur) puis on retourne le libellé de la langue associé (le nom de variable est alors suffixé par les deux lettres représentant la langue). Une récupération d'exception (try catch) permet d'évaluer la variable, et si elle n'existe pas, de récupérer l'erreur et de changer la langue en anglais (suffixe EN) pour cette valeur.

2.6.3 GESTION DE L'ERREUR SESSION INVALIDE EN MODE POPUP

Dans le cas d'une session invalide dans une popup, la redirection vers la page de login est maintenant effectuée. On a modifié la fonction `parseErrorXml(type)` de l'objet `XMLObject` défini dans le fichier javascript `fw_xml.js` du framework. Celle-ci, dans le cas où le type est égal à « popup » et si la session est invalide va ouvrir la page de redirection dans la frame principale et fermer la popup :

```
window.opener.document.location.replace(url+"&idmenu="+textIdmenu);
window.close();
```

2.6.4 LIENS DANS LA JSDoc

Les liens vers les fichiers de constantes, de valeurs par défaut et d'exemples dépendent désormais des variables `url.constValues`, `url.defaultValues` et `url.exemples` du fichier `build.jsdoc` : si ces variables sont non vides, elles doivent indiquer l'url et le nom du fichier associé. Voici les variables définies :

```
url.constValues=${build.includesJSDocRep}/constant-values_A3.tpl
url.defaultValues=${build.includesJSDocRep}/default-values_A3.tpl
url.exemples=${build.includesJSDocRep}/exemples_A3.tpl
```

Avec `build.includesJSDocRep` qui correspond au répertoire de stockage de la documentation du projet.

Dans le fichier `build.xml`, une nouvelle tâche ant a été créée : `initJSDocCondition`. Cette tâche teste si on a des constantes, valeurs par défaut ou exemples et initialise des variables contenant les urls de chacun des

fichiers. Elle est appelée par la tâche de génération de jsdoc qui les transfère au script perl ([jsdoc.pl](#)). Celui-ci passe alors les variables en paramètre lors du lien vers le fichier [overview-frame.tpl](#). Des tests permettent alors d'activer ou non les liens dans la documentation produite.

2.6.5 ACCES AU TEXTE D'UNE BALISE XML

Dans le fichier `fw_xml.js`, une nouvelle fonction globale a été implémentée : `getTextElement` qui prend en paramètre un élément xml et retourne soit la chaîne de caractères qu'il contient, soit null. La récupération du contenu d'une balise diffère selon les navigateurs, cela alourdissait tous les accès d'où cette fonction pour clarifier le code. Cette fonction est très utilisée dans le fichier `fw_formulaire_composant.js` pour les accès aux valeurs dans les flux de données et de paramètres.

Exemple :

```
var myBaliseName = parent.getElements(this.XMLObjectInfos.xmlDoc, "BALISE_NAME");  
//Récupération du contenu de la balise XML : différent selon le navigateur  
var theBaliseName = parent.getTextElement(myBaliseName[0]);
```

2.7 EXEMPLES SUPPLEMENTAIRES IMPLEMENTES

Pas d'exemple supplémentaire non lié à une nouvelle fonctionnalité