Modèle de développement des types d'éléments

Module : Eléments de cours

Les éléments de cours sont des briques éditoriales qui s'approchent de très prés à la “vision métier” du contenu, en fournissant à ce “métier” des modèles pré-formattés.

Le module propose en standard une batterie d'élément adressant les besoins récurrents et classiques d'une édition pédagogique. Dans certains cas, il peut être souhaitable, pour un projet pédagogique spécifique, de créer ses propres éléments de cours, plutôt que de redévelopper des plugins Moodle complets.

Eléments de base d'un type

Un type d'élément de cours est un sous-plugin au sens Moodle. Il dispose de sa version, de ses propres chaines de langues, de ses droits. Il est donc constitué comme un répertoire dans le chemin de base /mod/customlabel/type.

Le contenu obligatoire d'un sous-type est :

- [db]
  - access.php : définit les capacités de contrôle de visibilité du contenu
- [lang]
  - [en]
    - customlabeltype_newtype.php : le fichier de chaines
- customlabel.class.php : la classe d'implémentation
- customlabel.css : la feuille de style de base du widget
- version.php : le fichier de version

Ressources facultatives :

- [pix] : répertoire des images par défaut (> 3.0)

Le répertoire porte le nom du sous plugin qui servira de base aux noms formels dans son code. Dans ce qui suit le nom du plugin est supposé être **newtype**.

Le plugin est représenté par une sous-classe de la classe de base **customlabel_type**

```
require_once $CFG->dirroot.'/mod/customlabel/type/customtype.class.php';

class customlabel_newtype extends customlabel_type {
}
```

Un sous type d'élément de cours doit définir

- un type
- un tableau de champs
- des hooks de traitemement éventuels.

Trois méthodes sont suffisantes pour établir le type d'élément de cours :

Le constructeur de classe

Il définit tous les champs d'informations supportés par ce modèle spécifique d'élément de cours. La classe stocke comme membres le nom du sous-type et un tableau $this→fields de définition des champs.

```
    function __construct($data){
        parent::__construct($data);
        $this->type = 'coursedata';
        $this->fields = array();
```

Les champs sont définis selon le type de données qu'ils demandent à l'utilisateur lors de l'édition de l'étiquette. Les modèles suivants indiquent les différentes possibilités de type de données supportées :

```
        $field = StdClass();
        $field->name = '<fieldname>';
        $field->type = '<fieldtype>';
        $this->fields['<fieldname>'] = $field;
        ... attributs supplémentaires ...
```

Le type du champ de données déterminera comment la donnée sera demandée à l'éditeur du cours. Suivant ce type, des attributs supplémentaires peuvent être acceptés (ou sont requis) pour compléter la description.

Type de champ : Choix simple oui/non (choiceyesno)

```
        $field = new StdClass();
        $field->name = 'choiceyesnoname';
        $field->type = 'choiceyesno';
        $this->fields['choiceyesnoname'] = $field;
```

Est rendu comme une liste déroulante à deux options “Oui” et “Non”.

Type de champ : champ de texte (textfield)

```
        $field = new StdClass;
        $field->name = 'textfieldname';
        $field->type = 'textfield';
```

```
        $field->size = 40;  // optional
        $field->mawlength = 40;  // optional
        $this->fields['textfieldname'] = $field;
```

Produit un simple champ de saisie de texte.

Type de champ : zone de text (htmlarea ou editor)

```
        $field = new Stdclass();
        $field->name = 'textareafieldname';
        $field->type = 'textarea';
        $field->size = 80;
        $this->fields['textareafieldname'] = $field;
```

Produit en sortie une zone de texte simple (textarea) ou un éditeur Whysiwhyg (suivant les réglages généraux du site) pour entrer un texte multi-ligne ou un texte formaté.

Type de champ : Liste déroulante (list)

```
    $field = new StdClass();
    $field->name = 'datafieldname';
    $field->type = 'list';
    $field->options = 'comma,separated,list,of,option,keys';
    $field->multiple = 'multiple';  // adds multiple choice capability
    $this->fields['datafieldname'] = $field;
```

Produit une liste déroulante dont les valeurs sont les clef mentionnées dans l'attribut `options` et les labels visibles les traductions des clefs dans la langue courante. Les traductions des clefs doivent être fournies dans les fichiers de langue du sous-plugin.

Un certain nombre d'attributs optionnels peuvent apporter quelques modifications de comportement pour résoudre des situations approchantes:

```
    $field->straightoptions = true;
```

L'effet de cet attribut est que la liste présentera des valeurs et libellés identiques, construits à partir des clefs données dans l'attribut `options` sans aucune traduction.

Type de champ : Source de données (datasource)

```
  $field = new StdClass();
  $field->name = 'datafieldname';
  $field->type = 'datasource';
  $field->source = 'dbfieldkeyed';  // source mode
  $field->table = 'tablename where to get data from';
  $field->field = 'sourcefieldname';
  $field->select = 'some select clause';
```

```
  $field->multiple = 'multiple';  // adds multiple choice capability
(defaults to : no)
  $field->constraintson = 'level1,level2';
  $field->mandatory = true; // (defaults to : false)
  $this->fields['datafieldname'] = $field;
```

La source de données se présente comme une liste déroulante, mais construite à partir de données dynamiques stockées ailleurs dans Moodle. La source de données peut fournir les clefs et/ou les valeurs de liste. Ce choix est piloté par l'attribut `source` qui peut être :

- **dbfieldkey** : Obtient les labels dans un champ spécifique (attribut `field`) de la table spécifiée (attribut `table`), les clefs sont nécessairement les valeurs de la colonne `id` de la même table. Les résultats peuvent être triés par rapport à un champ de la table (attribut `sortorder`) et ils peuvent être filtrés par une clause SELECT portant sur la même table (attribut `select`).

Exemple :

```
    $field = new StdClass();
    $field->name = 'courseselectorbyid';
    $field->type = 'datasource';
    $field->source = 'dbfieldkey';
    $field->table = 'course';
    $field->field = 'fullname';
    $field->ordering = 'sortorder';
    $field->select = ' visible = 1 ';
    $this->fields['courseselectorbyid'] = $field;
```

- **dbfieldkeyed :** C'est une variante plus flexible que la première qui récupère les options de liste dans une table, mais avec la liberté de choisir à la fois le champ des clefs et le champ des labels de liste.

Exemple :

```
    $field = new Stdclass();
    $field->name = 'courseselectorbyshortname';
    $field->type = 'datasource';
    $field->source = 'dbfieldkeyed';
    $field->table = 'course';
    $field->field = 'fullname';
    $field->key = 'shortname';
    $field->select = ' visible = 1 ';
    $field->ordering = 'sortorder';
    $this->fields['courseselectorbyshortname'] = $field;
```

- **function :** Charge la liste déroulante à partir du résultat d'une fonction PHP qui doit renvoyer un tableau associatif de clefs/valeurs. Il n'est pour l'instant pas possible d'envoyer des paramètres à cette fonction. Cette forme est à utiliser si la récupération des options nécessite une jointure ou un traitement complexe.

```
    $field = new StdClass();
    $field->name = 'selectorbyfunction';
    $field->type = 'datasource';
```

```
    $field->source = 'function';
    $field->file = 'sourcefile/path/from/dirroot';
    $field->function = 'functionname';
    $this->fields['selectorbyfunction'] = $field;
```

Type de champ : Chargement de fichier (filepicker)

```
  $field = new StdClass();
  $field->name = 'datafieldname';
  $field->type = 'filepicker';
  $field->acceptedtypes = 'jpg,png,gif';
  $field->default =
'/theme/mytheme/pix_mod/customlabels/coureheading/default.jpg';
  $field->mandatory = true; // (defaults to : false)
  $this->fields['datafieldname'] = $field;
```

L'attribut `acceptedtypes` vaut par défaut '*'. Ce type de champ permettra à l'enseignant de téléverser une image qui pourra être utilisée dans le template. Il vous faudra probablement “post-traiter” un peu les données pour adapter l'URL et construire une balise IMG, ou directement dans un attribut SRC ou HREF. L'attribut `default` permet de définir une url par défaut si aucun fichier n'a été chargé par l'utilisateur.

Réalisation du contenu

Lorsque l'élément de cours est préparé (lors de son enregistrement), la séquence de production suivante est activée :

- Pre-traitement spécifique (si présent dans le sous-plugin)
- préparation des options de listes
- Récupération des sources de données
- Post-traitement spécifique (si présent dans le sous-plugin)
- Calcul du template

Le pré-traitement peut être utilisé pour apporter des données supplémentaires à l'élément de cours, provenant par exemple de sources externes (ldap, autres bases de données, calcul). Le post-traitement peut être utilisé pour formater les données finales ou calculer des variables de template non fournies par les sources de données ou la description du sous-plugin. Les deux méthodes à surcharger sont :

```
    public function preprocess_data() {
    }
```

```
    public function postprocess_data() {
    }
```

Dans ces méthodes, toutes les données disponibles pour l'élément de cours sont dans la structure `$this→data`

Templates et mise en forme

Les templates sont écrits sous forme de chaines de langue, afin de pouvoir localiser la production de l'élément de cours et permettre une modification plus facile par l'administrateur.

Exemple de template :

```
    $string['template'] = '
        <div class="custombox-commentbox">
            <%%comment%%>
        </div>
        <%if %%readmorecontent%% %>
        <div class="custombox-commentbox readmorelink">
            <a href="javascript:togglecustomstring(\'<%%customid%%>\',
\'Read more...\', \'Read less...\')"><span
id="customctl<%%customid%%>"><%%initialstring%%></span></a>
        </div>
        <div class="custombox-commentbox readmore"
id="custom<%%customid%%>">
            <%%readmorecontent%%>
        </div>
        <%endif %>

        <script type="text/javascript">
        setupcustomstring(\'<%%customid%%>\', \'<%%initiallyvisible%%>\',
\'Read more...\', \'Read less...\');
        </script>
    ';
```

L'injection de variable

Les variables de la structure `$this→data` peuvent être injectées au moyen des balises <varname>. Dans l'exemple ci-dessus, la variable `$this→data→readmorecontent` doit être alimentée si elle a un contenu fourni par l'enseignant.

Syntaxes conditionnelles simples

Pour permettre de résoudre des cas combinatoires simples (alternatives de mise en forme), une syntaxe conditionnelle simple est proposée :

```
<%if %%varname%% %>
<%endif %>
```

Multilinguisme

Les éléments de cours admettent une résolution multilingue, en proposant une version de template

par langue activée dans Moodle. La version anglaise du template est donc obligatoire. De ce fait, les textes des templates peuvent contenir des mentions littérales. La version provenant du fichier de langue correspondant à la langue active sera proposée en sortie. **Toutes les versions linguistiques sont précompilées à l'avance lors de l'enregistrement de l'élément de cours** pour pouvoir être délivrées selon le choix cynamique de la langue par l'utilisateur.

Pour obtenir un multilinguisme total, les enseignants devront également produire le contenu des champs de l'élément de cours avec la syntaxe multilingue de moodle.

RAPPEL : Pour correctement filtrer les éléments de cours qui peuvent contenir du HTML à structure complexe, le filtre “Multilingue amélioré” (filter_multilangenhanced) doit être installé et utilisé en remplacement du filtre mulitilangue standard.

Retour au guide technique - Revenir à l'index du composant - Revenir à l'index des plugins - Revenir au catalogue