Spécification de syntaxe

Moteur de script Moodlescript

Règles de base

La langage MoodleScript se veut un langage simple, facile à lire et à écrire, et qui évite les règles syntaxiques “techno” que l'on trouve dans d'autres principes d'écriture. La syntaxe évitera donc tout recours à des symboles et restera sur un principe de mot clefs explicites afin de garantir la lisibilité naturelle du code.

Une commande moodle sera habituellement composée d'un mot clef, d'arguments fixes, de variables et d'une liste de paramètre/attributs complémentaire. Tous les termes doivent être séparés par au moins un caractère. Une commande commencera toujours par une sémantique d'action (un verbe, ex. ADD, REMOVE, ENROL, BACKUP, etc.) et sera suivie par une alternace de mots-clefs et d'arguments, le tout formant une phrase “lisible” humainement. Certaines commandes pourront accepter une liste supplémentaire d'arguments, notamment pour apporter une liste de valeurs pour un objet à créer par exmeple, ou donnant des clauses conditionnelles pour la destruction d'un objet.

La forme générale d'une commande MoodleScript est donc :

```
VERBE MOTCLEF1{1,n} [ [arg1]{0,n} MOTCLEF1{0,n} ]{0,n}
```

Une commande acceptant une liste complémentaire d'attributs sera de la forme:

```
VERBE MOTCLEF1{1,n} [ [arg1]{0,n} MOTCLEF1{0,n} ]{0,n} **HAVING**
clef1: valeur1
[clef2: valeur2]
...
[clefn: valeurn]
```

Une commande se termine à la première ligne vide rencontrée.

Une commande ne pourra accepter qu'une seule liste d'arguments au plus.

Mots-clefs

Les mots clefs sont des mots à valeur spéciale lorsqu'ils sont positionnés à certains endroits de la commande. Les mots-clefs DOIVENT toujours être écrits en MAJUSCULES.

Les mots clefs sont :

Des verbes (toujours le premier mot d'une commande)

Les verbes sont des mots-clefs à un seul “token” (pas d'espaces) et doivent désigner une action (par ex. ADD ou REMOVE)

Verbes acceptés:

```
ADD, REMOVE, ENROL, BACKUP, MOVE
```

Types d'objets administrables

Les types d'objet désignent des objets “administrables” de moodle. Ils peuvent être des mots simples ou des expressions à plusieurs mots (mais toujours dans un ensemble connu).

```
COURSE, CATEGORY, ENROL METHOD, USER, COHORT, BLOCK, MODULE, etc.
```

Articulations

Les articulations contextuelles sont des petits mots qui indiquent ce que l'on fait des arguments qui les entourent, afin de faciliter la lisibilité “naturelle” de la commande.

```
IN, FOR, TO, IF EXISTS, IF NOT EXISTS
```

Arguments, identifiants et variables

Les arguments sont le plus souvent des identifiants d'objets moodle, des valeurs littérales terminales (qui s'expriment telle qu'écrites) ou éventuellement des appels à des variables globales ^permettant d'obtenir des “valeurs”. La nature et la signification de la valeur peut varier au fil de l'expression, mais se référera toujours à la donnée la plus “logiquement” attendue à cet endroit de la “phase”.

Par exemple, pour une instruction d'inscription, l'expression :

```
ENROL id:33 IN id:3 AS shortname:student USING manual
```

montre 4 attributs qui se réfèrent (successivement) à un Utilisateur, un Cours, un Role et une méthode d'inscription.

Dans le cas où une commande viendrait à créer des ambiguités, la syntaxe devra comporter des “adverbes” propres à lever cette ambiguité.

Identifiants

Lorsque l'expression doit identifier un objet déjà existant dans Moodle, et que plusieurs informations permettent d'identifier cet objet, un discriminateur explicite devra être utilisé pour exprimer la valeur, par exemple. pour un utilisateur, il existe actuellement quatre identifiants possibles: l'identifiant numérique primaire de base de données (id), l'identifiant de connexion (username), le numéro d'identification (idnumber) ou l'email.

Les expressions suivantes sont donc utilisables pour désigner un utilisateur dans moodle :

```
id:33
```

```
username:johndoe
```

```
idnumber:JD@35465
```

```
email:john.doe@gmail.com
```

Techniques spéciales d'identification : utiliser une fonction pour obtenir un identifiant

Dans certains usages, nous souhaiterions que l'identifiant soit fourni par une fonction existante (ou nouvelle) dans l'API moodle, qui s'exécutera dans le contexte courant du moteur. L'identifiant fera alors figurer le préfixe supplémentaire 'func', suivi d'un identifiant de fonction localisé dans son plugin comme suit :

```
idnumber:func:local_ent_installer@get_teacher_cat_idnumber
```

L'expression ci-dessus invoquera la fonction `local_ent_installer_get_teacher_cat_idnumber()` du plugin local `local_ent_installer`, en allant la chercher dans le fichier `locallib.php` (ou `lib.php` par defaut). La fonction devra retourner l'identifiant attendu. Dans l'exemple ci-dessus, la fonction devra retourner le numéro d'identification de l'objet administrable souhaité.

Vous ne pouvez pas passer de paramètres à cet appel, La fonction invoquée devra donc pouvoir fournir la réponse par la seule connaissance du contexte courant, grâce aux variables globales de moodle telles que $USER, $COURSE, etc.

Voici un exemple développé de l'usage d'une telle fonction qui obtient l'identifiant de la catégorie attribuée à un utilisateur lors de la création d'un cours:

```
  /**
   * Provides an uniform scheme for a teacher category identifier.
   * @param object $user a user object. If user is not given will return the
cat identifier of
   * the current user.
   * @return string
   */
  function local_ent_installer_get_teacher_cat_idnumber($user = null) {
      global $USER;

      if (is_null($user)) {
          $user = $USER;
      }

      $teachercatidnum =
strtoupper($user->lastname).'_'.substr(strtoupper($user->firstname), 0, 1).
      $teachercatidnum .= '$'.$user->idnumber.'$CAT';
      return $teachercatidnum;
   }
```

Appelé dans un contexte d'exécution d'un Moodlescript, elle calculera l'identifiant de catégorie pour l'utilisateur “actuellement détenteur” de la session courante, et nous pourrons écrire l'instruction de script suivante qui déplacera le cours “courant” dans la destination souhaitée pour l'utilisateur courant (celui au nom duquel le moteur de script est exécuté):

```
  MOVE COURSE current TO
idnumber:func:local_ent_installer@get_teacher_cat_idnumber
```

Lire plus loin pour la notion de contexte courant et le méta-identifieur “current”.

Arguments littéraux

Les arguments littéraux sont des mots simples ou des chaines de caractère. Dans la version actuelle du moteur, les chaînes ne sont pas délimitée syntaxiquement autrement que par la survenue d'un mot-clef attendu de l'expression. La conséquence est qu'il faudra éviter d'utiliser des littéraux qui utilisent les mots clefs du Moodlescript. C'est une des raisons principales pour laquelle nous avons décidé de forcer l'expression en majuscules de ces mots clefs, car cela diminue le risque de collision avec du texte usuel.

Variables

Nous aurons probablement à injecter dans le script des valeurs provenant de variables contextuelles. Pour cela nous mettrons en place des symboles “non terminaux” qui seront remplacés par leur valeur à l'exécution. La pile d'exécution d'un script peut être chargée au départ avec un contexte global de données qui sera abondé au contexte local qu'utilise une instruction pendant son exécution (le contexte local lui ajoutant ou surchargeant des valeurs). Ces valriables peuvent être placées n'importe où dans la syntaxe, en respectant la forme typique des arguments SQL dans une requête au format Moodle :

```
 :varname
```

Pour être reconaissable, l'appel à la variable DOIT être précédée par au moins un espace.

Une liste complète des variables de contexte disponibles pour les instructions peut être visualisée dans la trace d'exécution par l'instruction spéciale :

```
 LIST GLOBALS
```

Ceci affichera, par ex. dans la console Moodlescript des outils d'administration :

```
 > GLOBAL CONTEXT
 > wwwroot: http://dev.moodle31.fr
 > currentuserid: 2
 > currentusername: admin
 > siteshortname: DEV31
```

Usage en intégration :

Toute invocation d'une pile d'exécution MoodleScript peut être chargée avec un contexte d'entrée de départ, permettant à tout plugin d'y injecter le contexte dont il a besoin pour l'exécution de ses propres scriptlets.

Special keywords (metas)

'**current**'

'current' is a special keyword in place of an expected identifier that will resolve into the nearest current object in the executing environment. F.e, if the expected object is a user identifier, current will resolve to $USER→id. If 'current' addresses a course identifier, it wil usually resolve as $COURSE→id, unless another course id is given to the execution stack by the global context (for plugin developers).

The use of current will simplify scripts run within a known context, by using shorten expressions:

```
ADD ENROL METHOD guest TO current
```

For adding an enrolment method to the current course.

```
ENROL current INTO current AS student
```

For enrolling the current user (the $USER being executing the script) into the current course.

'**last**' or '**first**'

this usually addresses the first available or last available item in the current syntax context. this is used f.e. for blocks location in a region, but might also address any object locaton being sorted with a sortorder attribute.

'**runtime**'

Usually identifiers and variable can be evaluated at parse time or at check time, because they are litterals in the script, or they come from some input or global context. But this is not true in all cases. Lets take an example:

In the following scriptlet:

```
ADD CATEGORY "New category" TO idnumber:EXISTINGCAT HAVING
idnumber: NEWCAT

MOVE COURSE idnumber:SOMECOURSE TO idnumber:NEWCAT
```

We run into an issue because at parse time or at check time, NEWCAT category is not yet created. Thus we must tell the engine that in the second statement, we need the engine waiting the latest

moment to evaulate the identifier to move the course in.

This can be done by the special keywork runtime: and we'll rewrite the scriplet as follows:

```
ADD CATEGORY "New category" TO idnumber:EXISTINGCAT HAVING
idnumber: NEWCAT

MOVE COURSE idnumber:SOMECOURSE TO runtime:idnumber:NEWCAT
```

Using the runtime: special keyword will prevent the parser to evaluate and resolve the identifier, and will store it's initial definition in the handler class. The handler will also NOT try to resolve it at check time, as check time only checks the conditions of execution of all the statements without executing them. At real execution time, the identifier will be resolved to get it's definitive actual value.

Note that 'runtime' variables may raise a real error situation that cannot be recovered or anticipated by the engine and may terminate in a tecnhical failure of the script and the calling process.

Revenir à l'index du composant