

# 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 alternance 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 exemple, 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 ambiguïtés, la syntaxe devra comporter des "adverbes" propres à lever cette ambiguïté.

## 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 défaut). 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 T0  
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 chaînes de caractère. Dans la version actuelle du moteur, les chaînes ne sont pas délimitées syntaxiquement autrement que par la survenue d'un mot-clé 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 variables 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 reconnaissable, 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  
> sitedshortname: 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.

## Mots-clefs spéciaux (metas)

### 'current'

'current' est un mot-clef spécial qui se substitue en lieu et place d'un identifiant d'objet et qui se résoudra par l'objet du type attendu le plus "proche" dans l'environnement d'exécution. Par exemple, si l'identifiant attendu à cet emplacement est un identifiant d'un utilisateur, alors 'current' désignera le plus souvent \$USER→id. Si 'current' désigne un identifiant de cours, alors il se résoudra le plus souvent comme \$COURSE→id. En général, l'exécution d'une création d'un objet imposera la valeur courante du "dernier identifiant connu" par la pile d'exécution, afin que les instructions suivantes puissent naturellement le réutiliser. Ce concept du "dernier objet connu" par le contexte sous-tendra toute la programmation interne des commandes du Moodlescript.

L'utilisation du mot clef 'current' simplifie la compréhension de scripts qui se déroulent dans un contexte connu, en raccourcissant les instructions :

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

Pour ajouter la méthode d'inscription autorisant l'accès des invités dans le dernier cours connu par le contexte.

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

Pour inscrire le dernier utilisateur connu (en général l'utilisateur \$USER pour lequel le script est exécuté) dans le cours courant. (dernier cours créé, dernier cours mis à jour, etc.)

### 'last' ou 'first'

Ceci désignera généralement le premier ou dernier objet connu dans le contexte, lorsque la valeur associée au mot clef désigne un ensemble. Il est actuellement expérimentalement utilisé pour trouver la première zone de blocs disponible dans un format de cours, et sera étendu à d'autres usages où l'interrogation d'une liste ordonnée d'objets est souhaitable.

### 'runtime'

Par défaut, les identifiants et les variables sont évaluées au moment de la compilation du script ou sa validation. car les valeurs sont soit statiques (littéraux), soit proviennent de valeurs chargées *a priori* dans le contexte d'exécution. Mais ce n'est pas toujours le cas. Prenons un exemple :

Dans le scriptlet suivant :

```
ADD CATEGORY "Nouvelle catégorie" TO idnumber:PARENTE HAVING
```

```
idnumber: NOUVELLE
```

```
MOVE COURSE idnumber:UNCOURS TO idnumber:NOUVELLE
```

Il y a de manière évidente un problème à la compilation ou à la validation, car la catégorie désignée par NOUVELLE n'est pas encore créée. Nous devons donc indiquer à la deuxième instruction qu'il faudra attendre lors de l'exécution pour résoudre l'identifiant de la catégorie où le cours doit être déplacé.

Ceci peut être fait en ajoutant le préfixe spécial 'runtime:' et nous réécrivons le script comme suit :

```
ADD CATEGORY "Nouvelle catégorie" TO idnumber:PARENTE HAVING  
idnumber: NEWCAT
```

```
MOVE COURSE idnumber:UNCOURS TO runtime:idnumber:NOUVELLE
```

En utilisant le mot-clef 'runtime:' nous empêchons l'interpréteur syntaxique de résoudre cette valeur, et stockons sa définition initiale (sa référence à remplacer). 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 technical failure of the script and the calling process.

[Revenir à l'index du composant](#)

From:  
<https://docs.activeprolearn.com/> - **Documentation Moodle ActiveProLearn**

Permanent link:  
<https://docs.activeprolearn.com/doku.php?id=local:moodlescript:syntaxspecification>

Last update: **2026/01/13 07:19**

