Syntax specification

MoodleScript evaluable expressions

MoodleScript evaluable expressions, defined from version 2022081700 of the MoodleScript engine, allow writing tests on Moodle data, independently of, or on the contrary in relation to, the user's current situation. A generic syntax has been defined to express many cases involving different data. This syntax defines:

- A set of supported logical operators for combining “evaluable elements”
- The general form of an evaluable element
- The format of “literal” values

The logical operators

There are 4 of them, matching classic logical combinatorics:

- NOT (unary operator, negation)
- AND (binary, logical AND)
- OR (binary, logical OR)
- XOR (binary, exclusive OR)

The processing of logical combinatorics does not currently support nesting, and therefore marking with parentheses. The operator precedence rules are imposed: NOT > AND > OR or XOR.

General form of an evaluable element

An evaluable element is a proposition with at most two operands:

- [Operand 1] <operator> [Operand 2]
- [Operand 1] <operator> (this form is therefore accepted for a one-operand operator).

Two types of operators are defined: a type operating on “data”, such as the evaluation operators '=' or '<'. The operands are then final attribute “values”, comparable arithmetically or textually.

Example:

- `user:username:“harry”:department = “Physics”` (is “harry” tied to the Physics department)

The second type operates on object instances, such as the 'isloggedin' operator. In this case, the operands are references to Moodle object instances and not attribute values.

Example:

- `user:username:“harry” isenrolled course:current` (does “harry” have an active enrolment in the current course)

The complete list of supported “function” operators is given further below.

From the different definitions above, we see that the operands can be of three types:

- A literal scalar value (numeric or textual)
- A reference to an object
- A reference to a scalar attribute of an object (this case is an extension of the previous one)

Scalar values

Must be written between quotes if they are “alone”, i.e. the operand is by itself a scalar value.

Reference to an object

It breaks down into two or three segments that must define a unique instance of a certain type of Moodle business object. An “object type” is used as the first segment, the following segments must name an identifying attribute and set a value of the identifier.

General forms:

```
  [optype]:[identifying attribute]:[identifying value]
```

or

```
  [optype]:[deictic]
```

Examples:

- `user:id:364`
- `user:idnumber:FC12654` (or `user:idnumber:“FC12654”` also accepted)
- `course:shortname:“TRANSFORM 2020-2021”`
- `category:idnumber:“MAGICS”`
- `group:idnumber:“GRP-A”`

Special case, use of deictics:

Deictics are expressions whose value changes depending on the context or time of invocation. We will most often use the keyword “current”, which designates the object corresponding to the current context in which the script is executed.

Example:

- `user:current` designates the user corresponding to `$USER`

- `course:current` designates the course corresponding to `$COURSE`

Reference to a final attribute

A reference to a final attribute is built as an “object” reference followed by an attribute mention. These attributes are most often the “natural” attributes of the object in the database. In some cases however, and to help unify the language, some attribute names can be used to access related indirect data (for example, the custom profile fields of a user account).

General form:

```
[Object ref]:[attribute]
```

Example:

- `user:current:department`
- `course:idnumber:“TR2020”:visible`

Implementation details

Allowed object types

- **user**: A user
- **user_profile_field**: a custom user data field
- **course**: A course
- **category**: A course category
- **cohort**: A cohort
- **group**: A group, in the current course context

Supported operators

- **=**: is equal
- **!=**: is different
- **<** or **⇐**: is less than or less than or equal
- **>** or **>=**: is greater than or greater than or equal
- **~**: contains (regex)
- **!~**: does not contain (regex)

Supported "function" operators

- **hasrolein**: has a role in the context | <userref> hasrolein <courseref|categoryref>
- **isenrolledin**: has an active enrolment in the context | <userref> isenrolledin <courseref|categoryref>
- **hasloggedin**: has logged in at least once | <userref> hasloggedin
- **hascompleted**: has completed a context | <userref> hascompleted <courseref>
- **hasstarted**: has started working (in the sense of completion) | <userref> hasstarted <courseref|categoryref>

- **isincategory**: is a child of the category | <courseref|categoryref> isincategory <categoryref>
- **isincattree**: is in the tree | <courseref|categoryref> isincattree <categoryref>
- **isinsubs**: is in one of the sub-categories | <courseref|categoryref> isinsubs <categoryref>
- **isempty**: is empty | <categoryref|cohortref> isempty
- **isingroup**: is in the group | <userref> isingroup <groupref>

Upcoming objects (prospective)

- **module**: Course module (activity or resource)
- **badge**: Badge

Upcoming "function" operators (prospective)

- **isin**: is in the cohort or the group <userref> isin <cohortref|groupref>
- **hasbadge**: has acquired the badge
- **hascompleted**: extension to the <moduleref> context in addition to the <courseref> context

Upcoming features

- Named expressions: Evaluable expressions associated with a name so they can be used in several places and defined in a unique and centralised way. In anticipation, we note the need for an easily distinguishable syntax to fetch an expression from a catalogue. A proposal may be:

```
::<evaluablerulename>
```

which will clearly distinguish the reference call from the “by members” syntaxes of attribute or object reference calls. The rule name must be a unique token accepting alphanumeric characters and '_' without spaces. The use of extended or accented characters is not recommended.

Thus the simple mention of

```
::myrule1
```

will fetch the complete definition of the rule from the named rules catalogue. The catalogue location is in the tool_moodlescript accessory, the plugin authorised to offer administrator interfaces.

As a prospective, and to encourage the use of named rules, one can envisage passing parameters to a named rule, for example:

```
::myrule1 <param1> "<string param2>"
```

which will offer values to substitute in the named expression in the classic shell variable forms: $1, $2, etc… The following example shows what the writing of such an expression could be:

```
user:idnumber:$1:username ~ ^a
```

For a rule saying “does the username of the user given as parameter by $1 start with 'a'?”

Back to the component index - Back to plugins index - Back to the sets index - Back to catalogue

From:
https://docsen.activeprolearn.com/ - **Documentation Moodle ActiveProLearn**

Permanent link:
https://docsen.activeprolearn.com/doku.php?id=local:moodlescript:expressionsyntaxspecification

Last update: **2026/11/13 20:37**