DekiScript Syntax

This section describes the syntax of DekiScript, including which tokens are valid and how they can be combined.

The language is explained using the extended BNF notation, in which (*x*)* means 0 or more *x*'s, and (*x*)? means an optional *x*. Non-terminals are shown like *non-terminal*, terminals are shown like `terminal`, and tokens are shown like **token**.

Lexical Conventions

Names in DekiScript can be any string of letters, digits, and underscores, not beginning with a digit. Note that any character considered alphabetic by the neutral locale can be used in an identifier. Identifiers are used to refer to variables and table fields.

The following `tokens` are reserved and cannot be used as names: **_**, **nil**, **null**, **true**, **false**, **break**, **continue**, **switch**, **case**, **default**, **foreach**, **if**, **else**, **var**, **typeof**, **in**. DekiScript is not case-sensitive, meaning `page.path` and `PAGE.PATH` are identical.

Comments

You can add comments in Dekiscript by using /* and */ .

BNF Notation

A DekiScript uses the following syntax:

dekiscript ::= *statements*
statements ::= (*assign* (
;
statements)? | *let* (
;
statements)? | *expr* (
;
statements)? | *if-else statements* | *foreach statements* | *switch statements* |
break

(
;
statements)? |
continue
(
;
statements)?|
;
statements)?
assign ::=
var
`name` (
=
expr)? (
,
`name` (
=
expr)?)*
let ::=
let
`name` (
=
|
+=
|
-=
|
***=**
|
/=
|
%=
|
..=
) *expr*
if-else ::=
if
(
expr
)
block (
else
block)?
foreach ::=
foreach

```
(
var
name
in
expr
)
block
switch ::=
switch (
expr )
{
case ( case )*
}

case ::= (
case
expr |
default
) : block
block ::= (
{
statements
}
| assign
;
| let
;
| expr
;
| if-else
;
| foreach
;
)
expr ::= default-value (
?
expr
:
expr )*
null-expr ::= or-expr (
??
null-expr )*
or-expr ::= and-expr (
||
or-expr )*
and-expr ::= equ-expr (
&&
and-expr )*
equ-expr ::= rel-expr ((
!=
|
==
|
!==
|
```

===

) *equ-expr*)*
rel-expr ::= *concat-expr* ((
<

|
<=

|
>=

|
>

) *rel-expr* |
is

`name`)*
concat-expr ::= *add-expr* (
..

concat-expr)*
add-expr ::= *mul-expr* ((
+

|
-

) *add-expr*)*
mul-expr ::= *unary-expr* ((
*****

|
/

|
%

) *mul-expr*)*
unary-expr ::= (
-

|
!

|
typeof

)* *primary*
primary ::= (*literal* | `name` | **(** *expr* **)**) (
.

`name` |
[

expr
]

| *arg-list* | *map*)*
arg-list ::=
(

(*expr* (
,

expr)*)?
)

literal ::= `nil` | `bool` | `number` | `string` | *magic-id* | *map* | *list*
map ::=
{

(*field* (
,

field)*)?
}

```
field ::= ( name | string | number |
(
expr
)
)
:
expr
list ::=
[
( expr (
,
expr )* )?
]

magic-id ::=
@
name
```