Language Elements

Rule Engine · https://2yetis.net/docs/language-elements/

Item	Description
Comments	Specified using ## or //and extend to the end of line, e.g. `## This is a comment` Also specified using `//`, e.g. `// This is a comment` Multiple lines comments are specified using `/*...*/`, e.g. `/* This is a multi-line comment */`
Identifiers / variables	Must start with `a-z, A-Z, _ or $`. Can then be followed by `0-9, a-z, A-Z, _ or $`. e.g. Valid: `var1,_a99,$1` Invalid: `9v,!a99,1$` Variable names are **case-sensitive**, e.g. `var1` and `Var1` are different variables. **NOTE:** JEXL does not support variables with hyphens in them, e.g. `commons-logging // invalid variable name (hyphenated)` is not a valid variable, but instead is treated as a subtraction of the variable `logging` from the variable `commons` JEXL also supports `ant-style` variables, the following is a valid variable name: `my.dotted.var` N.B. the following keywords are reserved, and cannot be used as a variable name or property when using the dot operator: `or and eq ne lt gt le ge div mod not null true false new var do while break continue function return` For example, the following is invalid: `my.new.dotted.var // invalid ('new' is keyword)` In such cases, quoted identifiers or the [ ] operator can be used, for example: `my.'new'.dotted.var my['new'].dotted.var`
Statements	A statement can be the empty statement, the semicolon (;), block, conditional, variable declaration or an expression. Statements are optionally terminated with a semicolon. A single statement or a statement block can be annotated.
Block	A block is simply multiple statements inside curly braces ({, }).

Item	Description
Local variables	Can be defined using the let, const and var keywords; their identifying rules are the same as contextual variables. let declares a local variable (or a parameter) with a lexical block scope. The variable can only be accessed within its definition block and any nested sub-block. This also forbids variable redeclaration within that scope. Note: This emulates Java behavior which differs from ECMAScript. const behaves as let but will prevent the variable from being reassigned by any side effect operator. var declares a variable whose scope is the whole script and allows redefinition. This behavior is altered by the JexlFeature#setLexical(true) that will enforce a lexical scope for all variables akin to let declaration. Basic declaration: let x; Declaration with assignment: const theAnswer = 42; Invalid declaration: var x.y; Local variables they take precedence in resolution over contextual variables. When scripts are created with named parameters, those behave as local variables. Local variables can not use ant-style naming, only one identifier.
Expression	An expression can be the literal, variable, assignment, access operator, function definition, function call, method call or an evaluation operator.
Assignment	Assigns the value of a variable (my.var = 'a value') using a JexlContext as initial resolver. Both beans and ant-ish variables assignment are supported.
Access Operator	Allows to evaluate a property of an object, a value of the collection or an array by using either square brackets or a dotted numeral, e.g. foo.bar will access the bar property of the foo Object. arr1[0] will access the first element of the arr1 array. The safe-access operator foo?.bar shortcuts any null or non-existent references along the navigation path, allowing a safe-navigation free of errors. In the previous expression, if 'foo' is null, the whole expression will evaluate as null. This is an efficient shortcut to defensive expressions such as x?.y?.z that would otherwise be expressed as x? (x.y? (x.y.z ?: null) :null) : null. Properties can also be quoted as in foo.'b a r' and can be dynamic back-quoted interpolation strings as in cal.`${dd.year}-${dd.month}-${dd.day}`. These syntaxes are mixable with safe-access as in foo.'b a r'?.quux or foo?.`${bar}`.quux.