# List of contracts

In order to qualify our tool, we will split the contracts in categories regarding their goal. For each contract, we will describe its purpose, keyword, parameters if needed and assumptions taken. Every contract declaration must begin with the *contract* keyword and then the target specified with *requires* or *suggests*. Note that **":..."** stands for a symbol but it can be replaced by a String (:foo, "foo" and 'foo' are all correct).

## Style issues

This category specifies constraints regarding the style of the code.

### maxLength

| | |
|---|---|
| Description | Check if a method has no more than a given amount of instructions |
| Keyword | contract.requires(:method, :...).maxLength(number) |
| Parameters | • number : number of instructions |
| Assumptions | inst1;inst2 are considered as one instruction |

## Actual bugs or Potential bugs

This category specifies constraints regarding critical constraints that cannot fail or constraints that might be false positive. The difference is made whether the requires or suggests keyword is used.

### overrides|override

| | |
|---|---|
| Description | The method must override the ancestor's method |
| Keywords | contract.requires(:method, :...).overrides(:m_name) |
| Parameters | • m_name : name of the method |
| Assumptions | if the method doesn't exist in any ancestor, the method will be compared to a empty method with no instructions. Please also note that the method checked is the method from the class where the contract is declared, and not the direct parent. |

### has_return|have_return

| | |
|---|---|
| Description | The method must have a return statement |
| Keyword | contract.requires(:method, :...).has_return |
| Parameters | \ |
| Assumptions | short-cut for contract.requires(:...).call(:return) |

**calls|call**

| | |
|---|---|
| Description | The method must make a call of a other given method |
| Keywords | contract.requires(:method, :...).calls\|call(:m_name) |
| Parameters | • m_name : name of the method |
| Assumptions | The call can be made anywhere in the method<br>Be careful, the verification looks for a path where the call is not made, making short-cut. It ignores **if** statements without **else** statement, **while** (because condition can be wrong before first iteration), **case** without **else** statement, and doesn't check if the call made in a *super* call. |

**static_calls|static_call**

| | |
|---|---|
| Description | The method must make a static call of an other given method from a given class |
| Keywords | contract.requires(:method, :...).static_calls(:class_name, : m_name) |
| Parameters | • class_name : name of the class<br>• m_name : name of the method |
| Assumptions | The call can be made anywhere in the method<br>Same warning as previous contract regarding path looking |

**assigns|assign**

| | |
|---|---|
| Description | The method must assign a given variable |
| Keywords | contract.requires(:method, :...).assigns(:f_name) |
| Parameters | • f_name : name of the field to assign |
| Assumptions | The name of the variable given contains possible @ if the contract applies to instance or class variables (:a, :@a, :@@a)<br>Note that if the target is not a method, the assignment must be done outside of methods, else the assignment must be done in the given methods<br>Same warning as previous contract regarding path looking |

**is_private|are_privates**

| | |
|---|---|
| Description | The method must be private |
| Keywords | contract.requires(:method, :...).is_private |
| Parameters | \ |
| Assumptions | \ |

**begins_with_call|begin_with_call**

| | |
|---|---|
| Description | A given statement must be first |
| Keywords | contract.requires.method(:...).begins_with_call(:m_name) |
| Parameters | • m_name : name of the statement |
| Assumptions | the statement is a method call<br>Be careful, the verification looks for a first instruction which is not the wanted call, meaning **while** statements can be short-cut (if condition false before first iteration), **if** or **case** statement without **else** statement will extend the verification to the instruction after the if or case. |

**ends_with_call|end_with_call**

| | |
|---|---|
| Description | A given statement must be last |
| Keywords | contract.requires(:method, :...).ends_with_call(:m_name) |
| Parameters | • m_name : name of the statement |
| Assumptions | Same assumptions as previous contract |

**has_conditional_contracts|have_conditional_contracts**

| | |
|---|---|
| Description | If the first contract returns :pass, second must also pass |
| Keywords | contract.requires(:method, :...).has_conditional_contracts do<br>c_first<br>c_second<br>end |
| Parameters | • c_first : first contract<br>• c_second : second contract |
| Assumptions | The arguments are UContract declaration (contract.requires(:...).call(:...) ) or verification (calls(:...)). In the second case, the verification applies on the have_conditional_contracts contract. Note that we give a block as argument.Also, if one contract given is a full declaration and the target is :subclasses or :hierarchy or :methodsOverriding or :methodAndOverriders or :directSubclasses, the contract will be applied to children even if it is not relevant so be careful. |

**has_disjunctional_contracts|have_disjunctional_contracts**

| | |
|---|---|
| Description | At least one contract must pass |
| Keywords | contract.requires(:method, :...).has_disjunctional_contracts do<br>c_first<br>c_second<br>...<br>end |
| Parameters | • c_first : first contract<br>• c_second : second contract<br>• ... : as many contracts as wanted |
| Assumptions | Same warning on arguments as previous contract. |

**implements|implement**

| | |
|---|---|
| Description | The class implements a given method |
| Keywords | contract.requires(:class).implements(:m_name) |
| Parameters | • m_name : name of the method |
| Assumptions | \ |

**includes_module|include_module**

| | |
|---|---|
| Description | The class includes a given module |
| Keywords | contract.requires(:class).includes_module(:m_name) |
| Parameters | • m_name : name of the module |
| Assumptions | \ |

**extends_module|extend_module**

| | |
|---|---|
| Description | The class extends a given module |
| Keywords | contract.requires(:class).extends_module(:m_name) |
| Parameters | • m_name : name of the module |
| Assumptions | \ |

**matches_table|match_table**

| | |
|---|---|
| Description | The class name match a table name in the DB |
| Keywords | contract.requires(:class).matches_table($c\_name_1$, ...) |
| Parameters | • $c\_name_1$ : name of the table which must be in the DB. You can put any number of name as parameters. |
| Assumptions | if no parameter given, the name of the class targeted is used. |

**has_fields|have_fields**

| | |
|---|---|
| Description | The table in the DB contains the given filed names |
| Keywords | contract.requires(:class).has_fields(t_namet, $f\_name_1$, ...) |
| Parameters | • t_name : name of the table to check.<br>• $f\_name_1$ : name of the field which must be in the table. You can put any number of name as parameters. |
| Assumptions | Put the name of the table might be redundant but we want that the contract is generic so we offer the possibility to set name of the table. |