

# **Advanced Simulation Control**



### Prerequisites: Craft.CASE common using.

There are some cases in which during the simulation the user does not need to decide all the decisions himself because some of them are already decided. Craft.CASE has a unique ability to remember the decision and use the same or the derived decision later.

The following example shows a customer buying either a small thing, or a piece of furniture. The customer has already decided what to buy so the answer to the second pair of questions is redundant (see the picture left).

The standard behaviour of the simulator is that it lets us decide on every condition it meets, even though it might be semantically the same condition all the time. The reason is that the computer can't understand the meaning of human sentences.



Page 1 (total 4)



# Advanced Simulation Control

However, Craft.CASE has an additional module, *Advanced simulation*, through which you can control the simulation. You can "tell" Craft.CASE to remember your decision and to decide on your behalf later. The module uses two properties to reach this behaviour. One property, called *variable-function*, lets Craft.CASE remember your decision. The second property, *condition-function*, lets Craft.CASE decide on the condition. Both of the properties must be defined on transition-start, transition-end and communication lines. They are of type 'Function' and their initial value must be  $\{:C \mid nil\}$ .



These functions, like any other mathematical function, have a result. Here we will use only functions that have result as one of the following: *true*, *false*, *nil*. Inside the condition-function there must be either one of *true*, *false*, *nil*, which is the direct function result, or some more complex statement, that must also have the result *true*, *false* or *nil*.

When the Craft.CASE simulator reaches the condition with the result of its condition-function equal to *nil* (notice that the initial value fulfils this), it lets you decide on the condition, because *nil* means 'not known yet'. If the result of condition-function is *true* instead of the initial *nil*, it doesn't ask you, but it assumes that the condition is fulfilled and goes through this condition. On the other hand if the result is *false*, like in  $\{:C| false\}$ , it also doesn't ask you, but it assumes that the condition is not fulfilled and doesn't go through this condition.

CRAFT.CASE Ltd.

**Prerequisites:** 

Craft.CASE common using

Tel.: +44 (0)20 3287 4580 sales@craftcase.com www.craftcase.com



condition-function			
Result	nil	true	false
Meaning	not known yet	condition is fulfilled	condition is not fulfilled
Simulator	asks	doesn't ask	doesn't ask
Goes through	depends on reply	yes	no

While the condition-functions apply only to the lines with *condition* and help Craft.CASE to decide on the conditions, the variable-functions apply to all the lines and help Craft.CASE to remember the context of past data.

How can Craft.CASE remember the way the simulator already passed through? The answer is *variables*. A variable is a piece of memory in which Craft.CASE can store a value. Each variable has a name in the form C[name]. For our example on the first page we will use two variables: C[small] and C[big]. When we want to store a value in the variable we use an assignment command C[name] := value. In our example we want to store the value *true* in the variables, so we will use assignment commands C[small] := true and C[big] := true. The whole variable-functions will be  $\{:C | C[small] := true\}$  and  $\{:C | C[big] := true\}$ . The first will be set on the transition with 'Wants to buy a small thing' condition and the second will be set on the transition with 'Wants to buy a piece of furniture' condition.



The syntax of functions is very strict, so you must adhere to the rules. Craft.CASE checks the syntax against some mistakes. To check the syntax and to store the function in the condition-function or variable-function property, press Ctrl+Enter immediately after you type the function in the property. You can also press the right mouse over the white box with the function and select *Accept*. If something is wrong, the message Token not expected -> will appear near the mistake. If you want to place more than one assignment commands in the variable-function, end the first command with a full stop sign (.). Don't forget to press Ctrl+Enter in the Project settings window when typing the initial value of both properties.

At the beginning of simulation press the right mouse button over the large white box with a diagram in the simulator, and select from the context menu *Run script -> advanced-simulation -> activate*. Now when the simulator reaches the first pair of conditions it evaluates the condition-functions of both the transitions first. Both functions return *nil* so the simulator asks you which conditions to select. After you select some condition (e.g. small

#### CRAFT.CASE Ltd.

Tel.: +44 (0)20 3287 4580 sales@craftcase.com www.craftcase.com

#### **Prerequisites:**

Craft.CASE common using



# Advanced Simulation Control

thing), its variable-function will evaluate assigning *true* value into the appropriate variable (C[small]:=true). The second variable remains unassigned as the variable-function of the not-selected transition has not been evaluated (C[big] remains unassigned, i.e. its value will be *nil*). When the simulation reaches the second pair of conditions, we want the simulator to use the remembered decision. We will set the condition-functions of the second pair of conditions to  $\{:C| C[small] \text{ is true}\}$  and  $\{:C| C[big] \text{ is true}\}$ .



C[small] is true and C[big] is true are statements. Statements have, similarly to functions, a result, in this case one of true, false or nil. These statements test whether the values stored in appropriate variables are true. If yes, the statement has the result true. If there is something else than true, or the variable has no assigned value (i.e. there is nil), the result is false.

In statements you can use variables (*C[name]*), constants (*true*, *false*, *nil*), predicates (<, <=, =, >=, >, <>, *is*), logic conjunctions (*and*, *or*, *not*) and parentheses. The difference between the '*is*' and '=' predicates is that the '*is*' predicate always has the result of *true* or *false*, while '=' can also have a result of *nil*.

Now when the simulation reaches the second pair of conditions, it evaluates both conditionfunctions first, and depending on their results, it either asks or automatically goes through some branch. If we have previously selected the left branch ("wants to buy a small thing"), the first branch ("bought a small thing") is now true and the second one ("bought a piece of furniture") is false. Simulator doesn't ask but goes through the left branch.

## How to get the Advanced Simulation module

To get our module, download it from http://www.craftcase.com/Page/Modules.aspx

## Look for more information

• The C.C language tutorial can be found in the Resources section of our web page.

CRAFT.CASE Ltd.

Tel.: +44 (0)20 3287 4580 sales@craftcase.com www.craftcase.com **Prerequisites:** 

Craft.CASE common using