|  |
| --- |
| Circle Language Spec: Execution Flow |

## Execution Flow

Execution flow statements are points in a program, that take a decision about: what is the next step to take.

Execution flow is the only thing, that varies the order of execution in a program. It is all about choosing the next step to take.

There are three types of execution flow:

- Conditional execution

- Repetition

- Jumps

*Conditional execution* statements determine the next step to take, based on a condition. The If statement is the most common form of conditional execution.

*Repetition*, more commonly referred to as *loops*, are repetitions of the same code. The For loop is the most common form of repetition. Each repetition changes data, which gives the next repetition a different order of execution or makes the next repetition operate on different data.

A *jump* makes the program jump immediately to a different part of the program. However, sometimes a jump is placed inside a conditional execution statement, basing the jump on a decision after all.

There are multiple forms in which to express jumps, repetitions or conditional execution statements. Each form is explained in a separate article. The following forms of conditional execution statement are supported:

- If

- Select Case (exact value)

- Select Case (split formula)

The following forms of *repetition* are supported:

- For (range)

- For (conditional)

- For Each

- While

- Until

The following forms of *jumps* are supported:

- Normal execution order

- Label, Goto

- Call & Return

- Exit Command

- Exit Loop

- Continue

The last two jumps are only used inside a loop.

In the new computer language, execution flow statements are special commands. They are commands, whose arguments are command references. An execution flow command calls a command passed to it, based on a decision. The passed command references are called *clauses* of the execution flow command.

Even though execution flow commands are really just commands with command references, a call to an execution flow command is often called a *statement*.

Some of the execution flow commands are passed a *condition*. The condition will determine, what command reference is called. The condition is a boolean value. Most likely it is *not* a fixed boolean value. It is more likely to be a boolean variable stored inside an object. The boolean value can *also* be the result of a command. The boolean value can also be the result of a formula, but a formula actually *is* a command. So the condition is either a boolean variable inside an object, or the boolean output of a command.

For the If statement it is alright, when the condition is calculated *once* and the boolean result is passed to the If command as the condition. But in a conditional *loop*, the condition must be re-evaluated on every repetition of the loop. Recalculation of the condition, every time the condition is consulted, can be established by making the condition a reference to a reference to a Boolean, rather than a reference to a specific Boolean object.

So the condition argument refers to a *reference*.

The reference, that is referred to, can perform a calculation before it returns the Boolean. It performs the calculation every time the reference is consulted. How a reference can recalculate the value of an object whenever it is retrieved, is explained by the article *System Objects*.

Commands basically just call more commands. But a few commands do something other than just call other commands. The most basic ones of those are commands, that execute a machine instruction: an operation that is executed by the CPU, the central processing unit of the computer. But directly on top of the machine commands, there are the *execution flow commands*, that control the flow of a program, making the next command to call dependent on a condition. Any other command basically only calls more commands.

### In a Diagram

The concept of execution flow is explained in the article *Execution Flow*. The current article demonstrates its expression in a diagram.

The articles, that will follow, explain the diagram expression separately for each execution flow statement. But to explain their expression in general, the If statement is used as an example here.

An If statement executes a command if a certain condition is met. If the condition is not met, then an alternate command can be run.

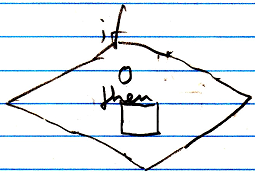
A textual expression of an If statement can look like this:

If Condition Then Command

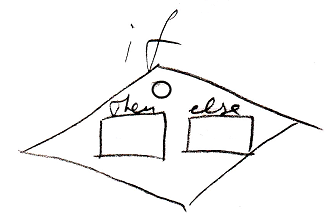
Or:

If Condition Then Command A Else Command B

In a diagram this looks as follows:

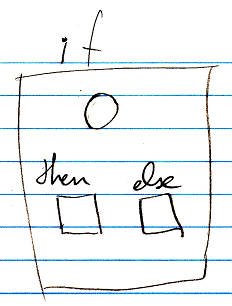


Or:



The circle is the condition. The word *condition* does not have to be shown in the diagram. The Then and the Else are two command references, that are passed to the If command.

The If statement is drawn out like a diamond shape. A diamond shape stands for a *call*. It is a *call* to an execution flow command. The definition of the execution flow commands is part of a system module of execution flow commands. It looks like this:



The definitions of the execution flow commands are squares. But usually you only see *calls* to execution flow commands, with a diamond shape.

The Condition and the Then and Else commands are shown right inside the call. But you are likely to want to see the Then and Else commands defined *outside* the If call.

|  |  |
| --- | --- |
|  |  |

You can do this by defining references to the clauses on a higher level. This will automatically connect the clauses inside the If symbol to the clauses outside the If symbol. A clause inside the If symbol and a clause outside the If symbol are actually both no more than references the same clause. So this will create references to the same clause on multiple levels. Following the rules of automatic containment, references to the same thing are tied together with lines. The symbol on the highest containment level represents the command itself. Putting a reference an a higher level for the sole purpose of elevating the definition of something, up to a level it is not required to be defined in, is called an *esthetic reference*.

See the articles *Esthetic Reference* and *Automatic Containment* for more about those subjects.

It is a coincidence, that a diamond was already used, to express a *decision* in many types of control-flow diagram, that already existed before this new computer language. It turns out, that using the existing syntax for calling commands, passing along command references as parameters, works perfectly for a clear way to express execution flow.