|  |
| --- |
| Circle Language Spec: Execution Control |

## Loops

### Concept

A loop is a form of *execution control*. Execution control is explained in the article *Execution Control*.

Loops, or repetition, are a main form of execution control, where the same code is repeated a number of times. The For loop is the most common form of repetition. On each repetition, data is changed, which gives the next repetition a different order of execution or makes it operate on different data.

A loop can go through a range of values, in case of which it is called a *range* loop. A loop can also stop repeating when a certain condition is met. In that case it is called a *conditional* loop.

The following forms of repetition are supported:

- For (range)

- For (conditional)

- For Each

- While

- Until

Each form is explained in a separate article.

### In a Diagram

Loops are a form of execution control explained in the article *Loops*. The articles that follow only explain their expression in a diagram.

There are four forms of repetition:

- For (range)

- For (conditional)

- For Each

- While

Each form is explained in a separate article. See the articles *For (range) in a Diagram*, *For (conditional) in a Diagram* and *For Each in a Diagram* and While in a Diagram.

### For

#### Concept

The For statement is a form of repetition. Repetition, more commonly know as *loops*, is a kind of execution control statement, explained by the article *Loops*.

In a For statement usually a range of values is gone through. The same code is executed for each of the values.

There are three kinds of For loop:

- For (range)

goes through a contiguous range of values

- For (conditional)

originally intended for going through a contiguous range of values, but more flexible than that.

- For Each

goes through the items of a collection

The three forms will be explained in separate articles.

#### Diagram

The concept of the For loop is already covered by the article *For*. The articles that follow only explains its expression in a diagram.

Each type of For loop has a slight variation in diagram notation. There are three definitions of For commands. They will be covered in the articles *For (range) in a Diagram*, *For (conditional) in a Diagram* and *For Each in a Diagram*.

### For (range)

#### Concept

There are three forms of For loop, as mentioned in the article *For*. This article explains the form of For loop where a contiguous range of values is gone through.

A contiguous range of values, usually numbers, starts at one value and ends at another value. On every loop, the value is increased by one or by a step smaller or langer than that. You can also go through for instance floating point numbers or dates.

The For command takes an initial value (From) as an argument, a last value (Till) and optionally the Step to take on loop. The step is 1 by default, but can also be a smaller step, like 0.5 or a larger step, like 2. The step can also be a negative, like -1, going through numbers in a reversed order.

The For command will also hold a read only parameter Variable, which is the current value of the numbers gone through.

The most important parameter, though, is a command reference to the actual command to loop. This is the Loop parameter, which is a command reference.

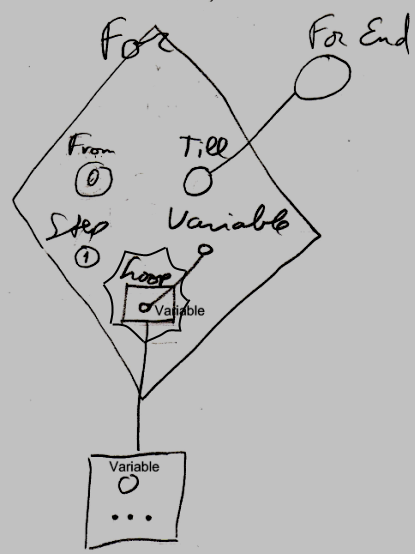
There is also a local object, called the Variable. The variable will hold the current value of the loop. The Variable will be passed to the Loop procedure as a parameter.

The implementation of this kind of For command uses machine instructions to increment the Variable and a conditional jump command to jump back to start another loop and call and return instructions to call the command to loop (or whatever other appropriate instructions forgotten about).

#### Diagram

The concept of the For statement for value ranges is already covered by the article *For (range)*. This article only explains its expression in a diagram.

Below is an example of the diagrammatic expression of a For statement, that goes through a contiguous range of numbers.

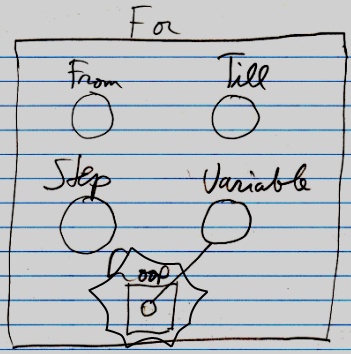


The diamond is a call to the For command. The For loop will go through a range of numbers. The first of the range of numbers is passed to the For command as the From parameter, which is visible inside the diamond as the circle named From. In the example above, the literal value of 0 is filled into the From parameter. This will be the first value of the range. The last value in the range is passed to the For command as the Till parameter. In the example above, the Till parameter is visible inside the diamond as the circle named Till. The Till parameter is a pointer to something defined outside the call to the For command. The Till parameter is pointing to a variable defined outside the call, displayed in the diagram above outside of the diamond, as the circle named For End. The For loop will go through the values 0 till the value indicated by the variable For End. The For loop will increment the current value by 1 on each loop, because the Step argument of the For command is set to the literal value of 1, visible inside the diamond as the circle named Step. The diamond contains another circle, named Variable, which will hold the current value of the range. This current value is passed to the Loop , that will be run multiple times. The Loop procedure is displayed inside the diamond, as the square named Loop. The the Loop procedure is *defined* outside the diamond. The Loop parameter, visible as a square named Loop inside the diamond, points out of the diamond to the larger square just below the diamond, with the ellipsis in it. The ellipsis stands for whatever you fill into it. The larger square below the diamond defines the command that will run multiple times, once for each value in the range. The circle inside the large square is the Variable passed to the Loop procedure. That circle is not tied to the Variable parameter inside the For call, but it is implicitly connected to it, because of the connection between the squares that contain the circles, and the name Variable repeated for each of the circles. (This is due to the rule of *implicit connection through parent.* See the article *Automatic Containment*.)

The Loop parameter is placed inside a nonagon purely for esteatic reasons. It is a single reference to a command, and the nonagon will never contain multiple references to commands, but it is obvious the command in it will be run multiple times. The nonagon stands for multiple, so it is put around the Loop parameter, but it will always contain a single item and you cannot add any more items to it.

In the example above, two of the circles got literal values filled in. One of the circles is tied to something defined outside the diamond. But any of the three From, Till and Step parameters could have had a literal value filled in or could have been pointing to something outside the diamond. The diagram above is just an example. The Loop procedure reference in our example, pointed out of the diamond as well. The loop procedure may as well have been defined right inside the diamond. The Variable of the loop can’t be given a literal value or be set to point to something outside the diamond, because the Variable of the loop is controlled by the For command definition, and can only read.

The *definition* of the For execution control command is part of a system module for execution control commands. The public elements of the definition look like this:



Nothing is filled in yet as the From , Till , Step or Loop . The Variable will be controlled by the For command. It is referenced from the Loop procedure.

### For (conditional)

#### Concept

There are three forms of For loop, as mentioned in the article *For*.

This article explains the form of For loop originally intended for going through a range of values, but more flexible than that. Unlike the *range* For this kind of For is a *conditional* loop.

The initial value of the range is replaced by an initialization procedure, which could do anything, but usually it assigns an initial value to a loop variable.

The last value of the range is replaced by a condition, which could be any condition, but usually it is the condition of the loop variable staying below or equal to the last value of the range.

The step to take going through the range of numbers is replaced by an action, which can be *any command*, but it usually is a procedure, that increments the loop variable with 1.

The For command takes a command reference to the Initialization procedure, which is called once, at the beginning of the loop. It also takes a command reference to the Action, which is a command called at the end of each repetition of the loop. The For command also takes a reference to another command, which is called the Loop command reference, which is the actual loop to repeat. The For command also takes a Boolean Condition, that has to stay True for the loop to keep repeating.

The Condition is usually passed to the For command as a reference to a reference to a Boolean. It needs to be a reference to a reference, because the condition needs to be recalculated every time it is consulted. Why making it a reference to a reference solves that problem is explained in the second last paragraph of the article *Execution Control*, part of which I repeat here:

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*.

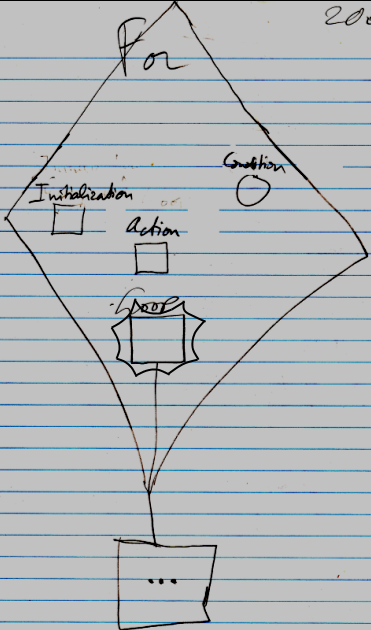
The reference, that is referred to, can perform a calculation before it returns the Boolean. It performs the calculation every time the reference is consulted. How a reference can recalculate the value of an object whenever it is retrieved, is explained by the article *System Interface*.

The implementation of this kind of For command uses a machine instruction to jump back in code to call the Loop again. For the rest it just calls the Loop and the Action command references and evaluates the Condition to determine whether to run the loop again.

#### Diagram

The concept of a conditional For statement is already covered by the article *For (conditional)*. This article only explains its expression in a diagram.

Below is an example of the diagrammatic expression of a conditional For statement.

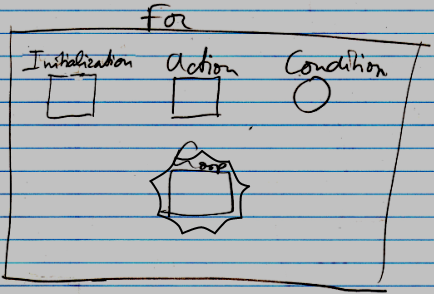


The diamond is a call to the For command. The For command will first call the Initialization procedure reference, displayed inside the diamond as a square called Initialization. The For command will then evaluate the Condition, displayed in the diamond as a circle named Condition. The Condition is a Boolean. When the Condition evaluates to True, the procedure reference Loop will be called, which is displayed inside the diamond as the square named Loop, but this square is tied to a bigger square outside the diamond, delegating the definition of the loop to outside the diamond. After execution of a loop, the Action is executed, which is visible in the diamond as a square named Action. The Action usually increments a loop variable. The loop variable is not defined in the statement. It has to be defined somewhere outside the statement. Then the Condition is evaluated again. If it is True, then the Loop procedure reference is called again. After that the Action is executed again, and the Condition is evaluated again and if it is True, the Loop procedure reference is called again. This happens over and over again, until the Condition will evaluate to False. Then the loop will stop repeating.

The Loop parameter is placed inside a nonagon purely for esteatic reasons. It is a single reference to a command, and the nonagon will never contain multiple references to commands, but it is obvious the command in it will be run multiple times. The nonagon stands for multiple, so it is put around the Loop parameter, but it will always contain a single item and you cannot add any more items to it.

In the example above, the Loop procedure reference was pointing to something outside the diamond, and the other command references were not. The other command references may as well have pointed to a command definition outside of the diamond and the Loop procedure may have been defined right inside the diamond all the same. The diagram above is just an example. The Condition could also have pointed to something outside the diamond.

The *definition* of the For execution control command is part of a system module for execution control commands. The public elements of the definition look like this:



Nothing is filled in yet as the Initialization , Action , Condition or Loop .

### For Each

#### Concept

There are three forms of For loop, as mentioned in the article *For*. This article explains the For Each loop, which goes through the items of a collection. the For Each statement is considered a *range* loop (see *Loops*).

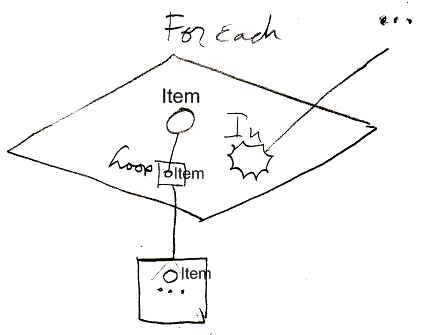
The collection is passed as an argument to the For command. The For command will also be passed a command reference that will be called once for every item in the collection. The command reference parameter is called Loop. The current item of the collection is stored as the Item parameter, that is read only, and controlled by the For command. The Item parameter is passed to the Loop procedure, each time it is called.

The implementation of the For Each loop does not only use plain machine instructions, because it it will be accessing a collection with a more object oriented approach. The looping itself can be controlled with machine instructions, though jumping back to the same call to the Loop procedure for each of the items of the collection.

#### Diagram

The concept of the For Each statement is already covered by the article *For Each*. This article only explains its expression in a diagram.

Below is an example of the diagrammatic expression of a For Each statement.



The diamond is a call to the For Each command. There is a nonagon called In visible inside the diamond. The In parameter can point to any collection to loop through.

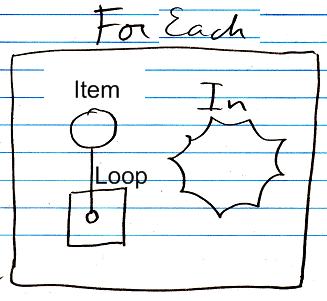
The diamond contains a circle, named Item, which will hold the current item of the collection. The current item is passed to the Loop procedure, that will be run multiple times. The Loop procedure is displayed inside the diamond, as a square named Loop. The Loop procedure is *defined* outside the diamond. The Loop parameter inside the diamond, points out of the diamond to the larger square just below the diamond, with the ellipsis in it. The ellipsis stands for whatever you fill into it. The larger square below the diamond defines the command that will run multiple times, once for each item in the collection. The circle inside the large square is the Item passed to the Loop procedure. That circle is not tied to the Item parameter inside the For call, but it is implicitly connected to it, because of the connection between the squares that contain the circles.

(This is due to the rule of *implicit connection through parent.* See the article *Automatic Containment*.) Officially, the two smaller circles explicitly have to be named Item as well, to not confuse the circles with any other possible circles, that might be defined in the Loop.

In other loops, the Loop parameter was surrounded by an esteatic nonagon, but because the In parameter is already a nonagon, it is already obvious, that the command handles a collection, so a nonagon is not also placed around the Loop parameter.

In the example above, the In collection and the Loop procedure reference were defined outside the diamond. They may as well have been defined right inside the diamond. The diagram above is just an example

The *definition* of the For Each execution control command is part of a system module for execution control commands. The public elements of the definition look like this:



Nothing is filled in yet as the In collection or the Loop procedure reference. The Item parameter will be controlled by the For command. It is referenced from the Loop procedure.

### While

#### Concept

The While loop is a form of execution control. Loops are briefly explained in the article *Loops*. This article explains the While loop.

The While loop keeps repeating the same code as long as a Condition is True.

The Condition is usually passed to the While command as a reference to a reference to a Boolean. It needs to be a reference to a reference, because the condition needs to be recalculated every time it is consulted. Why making it a reference to a reference solves that problem is explained in the second last paragraph of the article *Execution Control*, part of which I repeat here:

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 Interface*.

It needs to be said, that you are not obliged to pass a reference to a reference to a Boolean. It’s just that, in order to have a formula be reevaluated on every consult of the Boolean it must be a reference to a reference. The Boolean can also be just a reference to a Boolean. Then the Boolean has to change by some external force. You can also make it a fixed Boolean value, but then the loop will either not start at all or not end unless you explicitly call Exit Loop, based on some other condition (see *Exit Loop*).

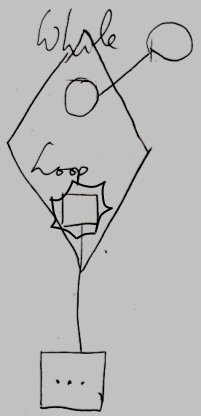
Next to the Condition, the While loop is passed a reference to a command as a parameter. The parameter is called Loop. This command is called repeatedly as long as the Condition returns True.

The implementation of the While loop incorporates an object oriented call to the reference to the reference to the Boolean Condition, and a call to a machine instruction to jump back to the call to the Loop procedure when the Condition returns True.

#### Diagram

The concept of the While loop is already covered by the article *While*. This article only explains its expression in a diagram.

Below is an example of the diagrammatic expression of a While loop.

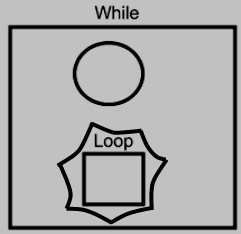


The diamond is a call to the While command. There is a circle inside the diamond, that represents the Boolean Condition. The name Condition is not shown with the circle, because it is the only circle there and it looks so obvious that it is the condition. The circle inside the diamond points to another circle outside the diamond, indicating that the Boolean is defined outside the call to the While statement. The diamond also contains a square called Loop, which is the reference to a command, that will be run as long as the condition is True. The Loop procedure reference points out of the diamond to a command defined outside the call to the While statement.

The Loop parameter is placed inside a nonagon purely for esteatic reasons. It is a single reference to a command, and the nonagon will never contain multiple references to commands, but it is obvious the command in it will be run multiple times. The nonagon stands for multiple, so it is put around the Loop parameter, but it will always contain a single item and you cannot add any more items to it.

In the example above, the Condition and the Loop procedure reference were defined outside the diamond. They may as well have been defined right inside the diamond. The diagram above is just an example.

The *definition* of the While execution control command is part of a system module for execution control commands. The public elements of the definition look like this:



Nothing is filled in yet as the Condition or the Loop procedure reference.

### Until

#### Concept

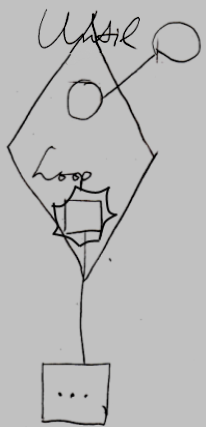
The Until loop is closely related to the While loop. It works exactly the same as the While loop, except that the while loop keeps repeating the same code as long as a condition equals True, and an Until loop keeps repeating the same code as long as a condition equals False. As soon as the condition becomes True the loop will stop.

For the rest of the inner workings of the Until loop, refer to the article While.

#### Diagram

The concept of the Until loop is already covered by the article *Until*. This article only explains its expression in a diagram.

Below is an example of the diagrammatic expression of a Until loop.

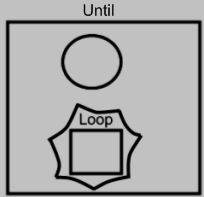


The diamond is a call to the Until command. There is a circle inside the diamond, that represents the Boolean Condition. The name Condition is not shown with the circle, because it is the only circle there and it looks so obvious that it is the condition. The circle inside the diamond points to another circle outside the diamond, indicating that the Boolean is defined outside the call to the Until statement. The diamond also contains a square called Loop, which is a reference to the command, that will be run as long as the condition is False. The Loop command reference points out of the diamond to a command defined outside the call to the While statement.

The Loop parameter is placed inside a nonagon purely for esteatic reasons. It is a single reference to a command, and the nonagon will never contain multiple references to commands, but it is obvious the command in it will be run multiple times. The nonagon stands for multiple, so it is put around the Loop parameter, but it will always contain a single item and you cannot add any more items to it.

In the example above, the Condition and the Loop procedure reference were defined outside the diamond. They may as well have been defined right inside the diamond. The diagram above is just an example.

The *definition* of the Until execution control command is part of a system module for execution control commands. The public elements of the definition look like this:



Nothing is filled in yet as the Condition or the Loop procedure reference.