

# Control Structures

When a macro is called, all its steps are executed sequentially.

However, you can define conditional steps - steps which are only executed if a condition is met. In order to do this, simply give the step a condition property, like in [Chase - Double speed](#):

```
<step condition="Math.IsEqual(Playbacks.Editor.Times.ChaseSpeed, 0.0)">
  ActionScript.SetProperty.Float("Playbacks.Editor.Times.ChaseSpeed", 1.0)
</step>
```

**At the moment (version 10.0 and 10.1) it looks like such conditions are stripped when a macro is copied in Titan. Hence, always move macros with conditions (which has the backdraw that you can have a macro only on one handle.**

This step is only executed if its condition is met - and the condition is a function written in double quotes: [Math.IsEqual](#) takes two values as arguments and returns true if both values are equal. In total, if the property `Playbacks.Editor.Times.ChaseSpeed` equals 0.0 then this step is performed and sets this property to 1.0.

Another mechanism is also related to control structures albeit strictly it is just juggling with booleans: simple toggle logic, see [Timecode - Enable/Disable](#):

```
<step>ActionScript.SetProperty.Boolean("Timecode.Enabled",
!Timecode.Enabled)</step>
```

This simply negates a variable, effectively turning it into its reciprocal value: if the variable 'Timecode.Enabled' is true it will be set to false, and vice versa.

There are also some other syntax versions:

```
<step>
{
  if(Math.IsNotEqual(Playbacks.Editor.Times.ChaseSpeed, 0.0)) {
    ** Do Something **
  } else {
    ActionScript.SetProperty.Float("Playbacks.Editor.Times.ChaseSpeed",
1.0);
  }
}
</step>
```

and as of version 10, it is possible to write conditions in a more modern way:

```
<step>
{
  if(Playbacks.Editor.Times.ChaseSpeed == 0.0) {
    ActionScript.SetProperty.Float("Playbacks.Editor.Times.ChaseSpeed",
1.0);
  }
}
```

```
}  
}  
</step>
```

## further readings

- [Introduction to macros](#)
- [Console and simulator](#) - how actions on the consoles are described
- [Recorded vs. coded macros](#) - both kinds: Country, AND Western
- [Macro file format](#) - what to observe when creating macro files
- [Macro Folders](#) - where exactly are the macro files stored
- [Deploying macros](#) - how to import a macro file into Titan
- [XML format](#) - a veeery basic introduction into the format macro files are written in
- [The Syntax of Functions](#) - understanding how functions are described in general
- [Control Structures](#) - conditions and other means to control the flow
- [Action and Menus](#) - when a menu needs to be toggled in addition to the action
- [Step Pause](#) - a little delay might sometimes be helpful
- [Active Binding](#) - highlighting a macro handle as active
- [Namespaces](#) - a way to keep order of the functions, properties and other stuff
- [Datatypes](#) - numbers, words, yes & no: the various types of values
- [Properties list](#) - the affected system variables of Titan
- [Function list](#) - the functions mentioned in this wiki
- [Examples list](#) - all the contributed macros. And where is yours?

2017/10/13 15:12 · icke\_siegen

From:  
<https://avosupport.de/wiki/> - **AVOSUPPORT**

Permanent link:  
[https://avosupport.de/wiki/macros/control\\_structures?rev=1511520768](https://avosupport.de/wiki/macros/control_structures?rev=1511520768)

Last update: **2017/11/24 10:52**

