Creating Parameters

You need parameters to connect the script module with controls on the macro page and to save the script module's state with the program. Before you can connect your script module with controls on a macro page, you must specify the parameters that you want to use in your script by calling the function defineParameter for each of them. Once a parameter is defined, it is shown in the **Parameter List**. From this list, you can then connect it with a control on the macro page. When you save the program, the parameters that you defined for the script module are saved with it.

The function defineParameter also creates a global variable that represents the value of the parameter in the script. You can use this global variable like any other variable in the script (see Parameter vs. Global Variables for details).

On this page:

Related pages:

- defineParameter

Defining Parameters

defineParameter(name)

You define a parameter by calling the function defineParameter with at least its name as the first argument. This name serves as name for the parameter in the **Parameter List** and as name for the global variable that represents the parameter in the script. The additional arguments of defineParameter are optional and can be used to change the characteristics of the parameter (see Parameter Characteristics for details). If no further arguments are defined, the parameter will be a floating point value in the range from 0 to 100.

Parameters vs. Global Variables

The function defineParameter creates a global variable that represents the value of the parameter in the script. It should be noted that:

- The rules for the naming and scope of global variables also apply for parameters.
- You can change the value of a parameter by assigning a new value to the corresponding global variable.
- The parameters that you defined for your script module are saved with the program, as opposed to global variables, which are not saved automatically.

The following example shows that parameters can be used just like global variables. After the parameter Scale has been defined, it is used to replace the note-on velocity. The value of Scale is changed by assigning the value of the last incoming MIDI controller to it.

Example 1

```
-- change the parameter Scale through MIDI controller and use its value to replace the note-on velocity

-- initialize variables
min = 0
max = 100
maxVel = 127
defVel = 100
default = defVel / maxVel * max
maxCC = 127

-- define Scale with the previous variables
defineParameter("Scale", nil, default, min, max)

-- use the value of Scale to replace the note-on velocity
function onNote(event)
    event.velocity = maxVel * Scale / max
    postEvent(event)
end

-- change the value of Scale through the last incoming MIDI controller
function onController(event)
    Scale = event.value / maxCC * max
end
```

Parameter Characteristics

How a parameter behaves depends on its characteristics. You determine the characteristics of a parameter with the arguments of defineParameter. To create a parameter with specific characteristics, the arguments must be set in the order in which they are shown in the following syntax examples.

Numeric

defineParameter(name, longName, default, min, max, increment, changeCallback)

Creates a numeric parameter. The `default` argument defines the value that the parameter will default to. The `min` and `max` arguments define the value range of the parameter. The `increment` argument defines the step size of the parameter. The arguments `default`, `min`, `max` and `increment` can be any integer or floating point value. How many digits are shown behind the decimal point for a value string of a parameter is determined by the value of the `increment` argument. For example:

Value	Description
`increment = 1`	The parameter will be an integer value and its value string will display no digits behind the decimal point.
`increment = 0.001`	The parameter will be a floating point value and its value string will display three digits behind the decimal point.
`increment = 0`	The parameter will be a floating point value and its value string will display two digits behind the decimal point.

The automatic formatting of a value can be overridden with the `format` argument. See Additional Named Arguments for more details.

Indexed String Array

defineParameter(name, longName, default, strings, changeCallback)

Creates a parameter with integer indices that have a text representation given by the string values of an array. The `default` argument defines the index that the parameter will default to. The `strings` argument must be an array with string values starting with index 0 or 1.

Boolean

defineParameter(name, longName, bool, changeCallback)

Creates a boolean parameter. The `bool` argument also defines the default value of the parameter.

String

defineParameter(name, longName, string, changeCallback)

Creates a parameter with a string value. You can change the string by assigning a new string value to the parameter.

Table

defineParameter(name, longName, table, changeCallback)

Creates a parameter with a table as value. The `name` argument of the parameter also defines the name of the table. You can access the values of the table using the regular methods, e.g., dot notation.

By Parameter Definition

defineParameter(name, longName, parameterDefinition, changeCallback)

Creates a parameter with the behavior of the specified parameterDefinition. You can use this to clone the behavior of existing parameters.

By Named Arguments

defineParameter { name = "p", longName = "param", default = 0, min = 0, max = 100, increment = 0.01, onChanged = callback, type = "float", format = "%.2f", readOnly = false, writeAlways = false, automatable = true, persistent = true }

Creates a parameter by named arguments. The only argument to the function is a table with the key/value pairs that define the parameter. The additional keys type, format, readOnly, writeAlways, automatable and persistent give you control over more advanced features. They can only be set with named arguments. See Defining Parameters by Named Arguments for more details.

Example 2

```
-- showcase different parameters

-- initialize variables
maxVelocity = 127

-- change callback of parameter "Name"
function nameChanged()
    print("name changed to", Name) --print the value of the parameter
end

-- initialize parameters
defineParameter("Scale", nil, 100) -- parameter with default 100 and range 0 to 100
defineParameter("Offset", nil, 0, -100, 100, 1) -- bipolar parameter with integer steps
defineParameter("Pan", nil, 0, -100, 100, 0.1) -- bipolar parameter with 0.1 steps
defineParameter("Mode", nil, 1, { "Off", "Normal", "Hyper" }) -- indexed string array
defineParameter("Enable", "Enable Filter", true) -- switch with long name
defineParameter("Label", nil, "untitled", nameChanged) -- string parameter
defineParameter("Intervals", nil, { 0, 4, 7 }) -- table parameter
defineParameter("Volume", nil, this.parent:getParameterDefinition("Level")) --[[ parameter with the same
behavior
                                                                                as the "Level" parameter
                                                                                of the parent layer ]]
-- use the parameters Scale and Intervals to play a chord with fixed velocity
function onNote(event)
    fixedVelocity = maxVelocity * Scale / 100
    local id1 = playNote(event.note + Intervals[1], fixedVelocity)
    local id2 = playNote(event.note + Intervals[2], fixedVelocity)
    local id3 = playNote(event.note + Intervals[3], fixedVelocity)
end
```

Parameter Change Callback

The change callback is only called if the value of the parameter was changed from the user interface, e.g., by adjusting the corresponding control on the macro page, or by calling setParameter. It is not called if the value was changed through assigning a value from inside the script. The following example revisits Example 1 to demonstrate this:

Example 3

```
-- change the parameter Scale through MIDI controller and use its value to replace the note-on velocity
-- the current value of Scale is printed only if changed from UI, e.g., go to the Parameter List to adjust
Scale

-- initialize variables
min = 0
max = 100
maxVel = 127
defVel = 100
default = defVel / maxVel * max
maxCC = 127

-- this callback function will only be called if you adjust Scale from the UI
function valueChanged()
    print("Value of Scale changed to:", Scale)
end

-- define Scale with the previous variables
defineParameter("Scale", nil, default, min, max, valueChanged)

-- use the value of Scale to replace the note-on velocity
function onNote(event)
    event.velocity = maxVel * Scale / max
    postEvent(event)
end

-- change the value of Scale through the last incoming MIDI controller
function onController(event)
    -- assigning a value to Scale will not call the callback function
    Scale = event.value / maxCC * max
end
```

Change Callback with Anonymous Function

In Example 2 the function nameChanged is declared before the parameter is defined. This is necessary for defineParameter in order to detect that the argument nameChanged is a function. If you want to declare the callback function after defining the corresponding parameter, you must call the callback function within an anonymous function. As the name suggests, an anonymous function is a function without a name.

Example 4

```
-- define string parameter
defineParameter("Name", nil, "untitled", function() nameChanged() end) --[[ if nameChanged is called inside
                                                                              an anonymous function, it can
                                                                              be declared after
defineParameter ]]

-- change callback of parameter "Name"
function nameChanged()
    print("name changed to", Name) -- print the value of the parameter
end
```

Defining Parameters By Named Arguments

When calling defineParameter with several arguments, the arguments are matched by their position and the associated values are passed on to the function. For this reason, the arguments of defineParameter must match the exact order and position when calling the function. Alternatively, you can set the arguments with the keys and values of a table. This method of passing arguments and values to a function is called *named arguments*.

Named arguments have the advantage that they can be set in any order you want and that optional or additional arguments can be left out without destroying the predefined order and position of the arguments to that function. The following example shows the parameters from Example 2 created with named arguments.

Example 5

```
-- different parameters created with named arguments

-- change callback of parameter "Name"
function nameChanged()
    print("name changed to", Name) --print the value of the parameter
end

-- parameter with default 100 and range 0 to 100
defineParameter{
        name = "Scale",
        default = 100
}

-- bipolar parameter with integer steps
defineParameter{
        name = "Offset",
        default = 0,
        min = -100,
        max = 100,
        increment = 1,
}

-- bipolar parameter with 0.1 steps
defineParameter{
        name = "Pan",
        default = 0,
        min = -100,
        max = 100,
        increment = 0.1,
}

-- indexed string array
defineParameter{
        name = "Mode",
        default = 1,
        strings = { "Off", "Normal", "Hyper" },
}

-- switch with long name
defineParameter{
        name = "Enable",
        longName = "Enable Filter",
        default = true,
}

-- string parameter
defineParameter{
        name = "Label",
        default = "untitled",
        onChanged = nameChanged,
}

-- table parameter
defineParameter{
        name = "Intervals",
        default = { 0, 4, 7 },
}
```

Creating a parameter by ParameterDefinition is not supported when using named arguments.

Additional Named Arguments

If you create a parameter by named arguments, you get access to these additional arguments:

type	The value type of the parameter (integer, float, boolean, string, variant, or envelope). The type must match the default and increment arguments.	string, optional

format	Formats the value string of a float value using the provided arguments. Only the format specifiers for float values are supported, i. e., e, E, f, g, or G. Other format specifiers are not supported. This overrides any automatic formatting from the `increment` argument.	string, optional
readOnly	The parameter can only be changed from the script if this is set to `true`. The argument defaults to `false` if no value is set.	bool, optional
writeAlways	A parameter does not call its change callback if its value is set without being changed. Set this to `true` if you want to guarantee that the change callback of the parameter is called. The argument defaults to `false` if not set.	bool, optional
automatable	Set this to `false` if you do not want the parameter to be automated. The argument defaults to `true` if not set.	bool, optional
persistent	The parameter will not be restored from the VST preset if this is set to `false`. The argument defaults to `true` if not set.	bool, optional

The arguments readOnly, writeAlways and automatable are helpful if you have a parameter that is used only for indication, but not for entering values.

Example 6

```
-- parameter that is read only, not automatable and not persistent
defineParameter {
    name = "deltaTime",
    longName = "Delta Time",
    default = 0,
    min = 0,
    max = 2^31,
    type = "float",
    format = "%.3f ms",
    readOnly = true,
    automatable = false,
    persistent = false,
}

-- measure the time between subsequent notes
function onNote(event)
    postEvent(event)
    t2 = getTime()
    if t1 then
        deltaTime = t2 - t1
    end
    t1 = t2
end
```