Layer

The Layer class inherits all properties and methods of the Element class.

On this page:

Class Hierarchy

- AudioFile
- Element
  - Bus
  - Effect
  - Instance
  - Layer
    - Program
  - MidiModule
  - ModulationMatrixRow
  - Slot
  - Zone
- Event
- LoadProgress
- ParameterDefinition

Element

Element Class, findChildren, getChild, getParameter, getParameterDefinition, getParameterNormalized, hasParameter, removeFromParent, setName, setParameter, setParameterNormalized

Constructors

Layer Constructor

Layer()

Description

Constructor to create a new Layer object.

Available in: Controller.

Return Values

Returns a new Layer object.

Example

```
-- This function creates different types of objects in the Program Tree.
-- The objects in the Program Tree do not have a name. You will see only their icons.
function createProgram()
  local inst = this.program.instance
  local prg = Program()
  local bus = Bus()
  prg:appendBus(bus)
  inst:setProgram(prg, 1)
  local layer = Layer()
  prg:appendLayer(layer)
  layer:appendZone(Zone())
  local mm = MidiModule('MIDI Player')
  layer:appendMidiModule(mm)
  local fx = Effect('Distortion')
  bus:appendEffect(fx)
end

createProgram()
```

Jump to Top

Methods

appendBus

appendBus(bus)

Description

Function to add a bus in the specified destination layer. The destination layer is determined by its Layer object. For example, `this.parent` specifies the parent layer of the script module as destination layer. The bus to be added is determined by its Bus object. You can use getBus or findBusses to determine the bus. The new bus will be added behind the existing busses. To insert a bus at a specific position in the destination layer, use insertBus instead.

An Element object can only have one parent. It cannot be child of multiple parents. Therefore, an Element object that you retrieved from the running plug-in instance must be removed before it can be inserted again. The Element objects that you retrieve through loadPreset or loadPresetAsync can be inserted freely, because these functions create a copy of the Element objects when reading them.

Available in: Controller.

Arguments

bus	The Bus object of the bus that you want to append.	Bus

Example

To explore the following script:

1. Download Program.vstpreset.
2. Drag the preset on the MediaBay to import it to the user folder for VST presets.
3. Create an empty program and add a script module.
4. Paste the script into the text editor of the script module and execute the script.

```
-- append bus from Program.vstpreset into the current program

-- get the file path for user VST presets
path = getUserPresetPath()

-- load VST preset
loadedProgram = loadPreset(path.."/Program/Program.vstpreset")

-- get the first bus from the loaded program
bus = loadedProgram:getBus()

-- append bus
if bus then
    this.program:appendBus(bus)
end
```

Jump to Top

appendLayer

appendLayer(layer)

Description

Function to add a layer in the specified destination layer. The layer to be added and the destination layer are both determined by their Layer objects. You can use getLayer or findLayers to determine the layer to be added. For example, `this.parent` defines the parent layer of the script module as destination layer. The new layer will be added behind the existing layers. To insert a layer at a specific position in the destination layer, use insertLayer or insertLayerAsync instead.

An Element object can only have one parent. It cannot be child of multiple parents. Therefore, an Element object that you retrieved from the running plug-in instance must be removed before it can be inserted again. The Element objects you retrieve through loadPreset or loadPresetAsync can be inserted freely, because these functions create a copy of the Element objects when reading them.

Available in: Controller.

Arguments

layer	The Layer object of the layer that you want to append.	Layer

Example

To explore the following script:

1. Download Program.vstpreset.
2. Drag the preset on the MediaBay to import it to the user folder for VST presets.
3. Create an empty program and add a script module.
4. Paste the script into the text editor of the script module and execute the script.

```
-- append layer from Program.vstpreset into the current program

-- get the file path for user VST presets
path = getUserPresetPath()

-- load VST preset
loadedProgram = loadPreset(path.."/Program/Program.vstpreset")

-- get the first layer from the loaded program
layer = loadedProgram:getLayer ()

-- append layer
if layer then
    this.program:appendLayer(layer)
end
```

Jump to Top

appendLayerAsync

appendLayerAsync(layer, callback)

Description

Function to add a layer in the specified destination layer using a separate, parallel thread. Appending a layer in a separate thread can be necessary if the layer is too big to be added in a short time. The layer to be inserted and the destination layer are both determined by their Layer objects. You can use getLayer or findLayers to determine the layer to be inserted. For example, `this.parent` determines the parent layer of the script module as destination layer. The new layer will be added behind the existing layers. To insert a layer at a specific position in the destination layer, use insertLayer or insertLayerAsync instead. The function returns a LoadProgress object that can be used to monitor the load progress. After the layer is added, the callback function is called. The callback function gets the LoadProgress object as default argument.

An Element object can only have one parent. It cannot be child of multiple parents. Therefore, an element object that you retrieved from the running plug-in instance must be removed before it can be inserted again. The element objects you retrieve through loadPreset or loadPresetAsync can be inserted freely, because these functions create a copy of the Element objects when reading them.

Available in: Controller.

Arguments

layer	The Layer object of the layer that you want to append.	Layer
callback	Callback function that is called when the layer is added. The callback function gets the LoadProgress object as argument.	function, optional

Return Values

Returns a LoadProgress object.

Example

```
-- start with an empty program, remove all existing layers
layers = this.parent:findLayers()

if layers then
   for i, layer in ipairs(layers) do
       this.parent:removeLayer(layer)
   end
end

-- table with layer presets from Skylab
layerPresets = {
   { name = "Ambient Pad 01", path = "vstsound://724ACB205EFF46F885735D1B216C37AD/.AppData/Steinberg
/Skylab/Sub Presets/Layer Presets/Ambient Pads/Ambient Pad 01.vstpreset" },
   { name = "Ambient Pad 02", path = "vstsound://724ACB205EFF46F885735D1B216C37AD/.AppData/Steinberg
/Skylab/Sub Presets/Layer Presets/Ambient Pads/Ambient Pad 02.vstpreset" },
   { name = "Ambient Pad 03", path = "vstsound://724ACB205EFF46F885735D1B216C37AD/.AppData/Steinberg
/Skylab/Sub Presets/Layer Presets/Ambient Pads/Ambient Pad 03.vstpreset" },
   { name = "Ambient Pad 04", path = "vstsound://724ACB205EFF46F885735D1B216C37AD/.AppData/Steinberg
/Skylab/Sub Presets/Layer Presets/Ambient Pads/Ambient Pad 04.vstpreset" },
}
-- create table with the preset names
function getPresetNames()
   presetNames = {}
   for i, preset in ipairs(layerPresets) do
       presetNames[i] = preset.name
   end
end

getPresetNames()

-- remove the old layer after the new one was added
function removeOldLayer(progressInfo)
   local newPreset = progressInfo.root
   if oldPreset then
       this.parent:removeLayer(oldPreset)
       print(oldPreset.name.." removed.")
   end
   oldPreset = newPreset
end

-- append the preset in a separate thread
function appendNewLayer(progressInfo)
   if progressInfo.root then
       this.parent:appendLayerAsync(progressInfo.root, removeOldLayer)
       print("Appending "..progressInfo.root.name.."...")
   end
end

-- load the preset in a separate thread
function onSelectPresetChanged()
   progress = 0
   progressInf = loadPresetAsync(layerPresets[SelectPreset].path, appendNewLayer)
   print("Loading "..layerPresets[SelectPreset].name.."...")
end

-- define a parameter for selecting the preset to be loaded
defineParameter("SelectPreset", "Select Preset", 1, presetNames, onSelectPresetChanged)

-- monitor the progress with onIdle
progress = 1
function onIdle()
   if progress < 1 then
       progress = progressInf.progress
       print("Progress: "..(progressInf.progress * 100).."%")
   end
end
```

Jump to Top

appendMidiModule

appendMidiModule(module)

Description

Function to add a MIDI module in the specified destination layer. The destination layer is determined by its Layer object. For example, `this.parent` specifies the parent layer of the script module as destination layer. The MIDI module to be added is determined by its MidiModule object. You can use getMidiModule or findMidiModules to determine the desired MIDI module. The new MIDI module will be added behind the existing MIDI modules. To insert a MIDI module at a specific position in the destination layer, use insertMidiModule instead.

An Element object can only have one parent. It cannot be child of multiple parents. Therefore, an Element object that you retrieved from the running plug-in instance must be removed before it can be inserted again. The Element objects you retrieve through loadPreset or loadPresetAsync can be inserted freely, because these functions create a copy of the Element objects when reading them.

Available in: Controller.

Arguments

module	The MidiModule object of the MIDI module that you want to append.	MidiModule

Example

To explore the following script:

1. Download Program.vstpreset.
2. Drag the preset on the MediaBay to import it to the user folder for VST presets.
3. Create an empty program and add a script module.
4. Paste the script into the text editor of the script module and execute the script.

```
-- append MIDI module from Program.vstpreset into the current program

-- get the file path for user VST presets
path = getUserPresetPath()

-- load VST preset
loadedProgram = loadPreset(path.."/Program/Program.vstpreset")

-- get the first MIDI module from the loaded program
module = loadedProgram:getMidiModule()

-- append MIDI module
if module then
    this.program:appendMidiModule(module)
end
```

Jump to Top

appendZone

appendZone(zone)

Description

Function to add a zone in the specified destination layer. The destination layer is determined by its Layer object. For example, `this.parent` specifies the parent layer of the script module as destination layer. The zone to be added is determined by its Zone object. You can use getZone or findZones to determine the zone. The new zone will be added behind the existing zones. To insert a zone at a specific position in the destination layer, use insertZone instead.

An Element object can only have one parent. It cannot be child of multiple parents. Therefore, an Element object that you retrieved from the running plug-in instance must be removed before it can be inserted again. The Element objects you retrieve through loadPreset or loadPresetAsync can be inserted freely, because these functions create a copy of the Element objects when reading them.

Available in: Controller.

Arguments

zone	The Zone object of the zone that you want to append.	Zone

Example

To explore the following script:

1. Download Program.vstpreset.
2. Drag the preset on the MediaBay to import it to the user folder for VST presets.
3. Create an empty program and add a script module.
4. Paste the script into the text editor of the script module and execute the script.

```
-- append zone from Program.vstpreset into the current program

-- get the file path for user VST presets
path = getUserPresetPath()

-- load VST preset
loadedProgram = loadPreset(path.."/Program/Program.vstpreset")

-- get the first zone from the loaded program
zone = loadedProgram:getZone()

-- append zone
if zone then
    this.program:appendZone(zone)
end
```

Jump to Top

findBusses

findBusses(recursive, nameOrFilterFunction)

Description

Function to find busses in the specified Element object. For example, `this.parent` specifies the parent of the script module as the Element object to be searched in. If recursive is set to `true`, subelements will also be searched. The function returns an array with the Bus objects of the found busses. Particular busses can be searched by name or through a filter function. If searching by name, findBusses accepts only the Bus objects that match the specified name. The filter function uses the Bus object of each bus as argument. Only those Bus objects that return `true` for the search criteria defined in the filter function will be accepted by findBusses. Without a name or filter function the Bus objects of all busses in the searched Element obects will be returned.

Available in: Controller, Processor.

Arguments

recursive	If set to `false`, only the specified Element object will be searched. If set to `true`, subelements will also be searched. The default is `false`.	boolean
nameOrFilterFunction	The name of the busses searched for or a filter function. Only the Bus objects that match the name or return `true` for the search criteria of the filter function will be accepted. Set this to `nil` to deactivate any name filter or search criteria.	string or function, optional

Return Values

Returns an array with the Bus objects of the found busses.

Example

```
-- find all busses and print their names
busses = this.program:findBusses(true)

if busses[1] then
    for i, bus in ipairs(busses) do
        print(bus.name)
    end
else
    print("Could not find any busses!")
end
```

Jump to Top

findEffects

findEffects(recursive, nameOrFilterFunction)

Description

Function to find effects in the specified Element object. For example, `this.parent` specifies the parent of the script module as Element object to be searched in. To specifiy a bus to be searched in, use getBus or findBusses. If recursive is set to `true`, subelements will also be searched. The function returns an array with the Effect objects of the found effects. Particular effects can be searched by name or through a filter function. If searching by name, findEffects accepts only the Effect objects that match the specified name. The filter function uses the Effect object of each effect as argument. Only those Effect objects that return `true` for the search criteria defined in the filter function will be accepted by findEffects. Without a name or filter function the Effect objects of all effects in the searched Element objects will be returned.

Available in: Controller, Processor.

Arguments

recursive	If set to `false`, only the specified Element object will be searched. If set to `true`, subelements will also be searched. The default is `false`.	boolean
nameOrFilterFunction	The name of the effects searched for or a filter function. Only the Effect objects that match the name or return `true` for the search criteria of the filter function will be accepted. Set this to `nil` to deactivate any name filter or search criteria.	string or function, optional

Return Values

Returns an array with the Effect objects of the found effects. Returns an empty table if no effects are found.

Example

```
-- find all effects and print their names
effects = this.program:findEffects(true)

if effects[1] then
    for i, effect in ipairs(effects) do
        print(effect.name)
    end
else
    print("Could not find any effects!")
end
```

Jump to Top

findLayers

findLayers(recursive, nameOrFilterFunction)

Description

Function to find layers in the specified layer. For example, `this.parent` specifies the parent layer of the script module as the layer to be searched in. If recursive is set to `true`, sublayers will also be searched. The function returns an array with the Layer objects of the found layers. Particular layers can be searched by name or through a filter function. If searching by name, findLayers accepts only the Layer objects that match the specified name. The filter function uses the Layer object of each layer as argument. Only those Layer objects that return `true` for the search criteria defined in the filter function will be accepted by findLayers. Without a name or filter function the Layer objects of all layers in the searched layers will be returned.

Available in: Controller, Processor.

Arguments

recursive	If set to `false`, only the current layer will be searched. If set to `true`, sublayers will also be searched. The default is `false`.	boolean
nameOrFilterFunction	The name of the layers searched for or a filter function. Only the Layer objects that match the name or return `true` for the search criteria of the filter function will be accepted. Set this to `nil` to deactivate any name filter or search criteria.	string or function, optional

Return Values

Returns an array with the Layer objects of the found layers. Returns an empty table if no layers are found.

Example

```
-- find all layers and print their names
layers = this.program:findLayers(true)

if layers[1] then
    for i, layer in ipairs(layers) do
        print(layer.name)
    end
else
    print("Could not find any layers!")
end
```

Jump to Top

findMidiModules

findMidiModules(recursive, nameOrFilterFunction)

Description

Function to find MIDI modules in the specified layer. For example, `this.parent` specifies the parent layer of the script module as the layer to be searched in. If recursive is set to `true`, sublayers will also be searched. The function returns an array with the MidiModule objects of the found MIDI modules. Particular MIDI modules can be searched by name or through a filter function. If searching by name, findMidiModules accepts only the MidiModule objects that match the specified name. The filter function uses the MidiModule object of each MIDI module as argument. Only those MidiModule objects that return `true` for the search criteria defined in the filter function will be accepted by findMidiModules. Without a name or filter function the MidiModule objects of all MIDI modules in the searched layers will be returned.

Available in: Controller, Processor.

Arguments

recursive	If set to `false`, only the current layer will be searched. If set to `true`, sublayers will also be searched. The default is `false`.	boolean
nameOrFilterFunction	The name of the MIDI modules searched for or a filter function. Only the MidiModule objects that match the name or return `true` for the search criteria of the filter function will be accepted. Set this to `nil` to deactivate any name filter or search criteria.	string or function, optional

Return Values

Returns an array with the MidiModule objects of the found MIDI modules. Returns an empty table if no MIDI modules are found.

Example

```
-- find all MIDI modules and print their names
modules = this.program:findMidiModules(true)

if modules[1] then
    for i, module in ipairs(modules) do
        print(module.name)
    end
else
    print("Could not find any MIDI modules!")
end
```

Jump to Top

findZones

findZones(recursive, nameOrFilterFunction)

Description

Function to find zones in the specified layer. For example, `this.parent` defines the parent layer of the script module as the layer to be searched in. If recursive is set to `true`, sublayers will also be searched. The function returns an array with the Zone objects of the found zones. Particular zones can be searched by name or through a filter function. If searching by name, findZones accepts only the Zone objects that match the specified name. The filter function uses the Zone object of each zone as argument. Only those Zone objects that return `true` for the search criteria defined in the filter function will be accepted by findZones. Without a name or filter function the Zone objects of all zones in the searched layers will be returned.

Available in: Controller, Processor.

Arguments

recursive	If set to `false`, only the current layer will be searched. If set to `true`, sublayers will also be searched. The default is `false`.	boolean
nameOrFilterFunction	The name of the zones searched for or a filter function. Only the Zone objects that match the name or return `true` for the search criteria of the filter function will be accepted. Set this to `nil` to deactivate any name filter or search criteria.	string or function, optional

Return Values

Returns an array with the Zone objects of the found zones. Returns an empty table if no zones are found.

Example

```
-- find all zones and print their names
zones = this.program:findZones(true)

if zones[1] then
    for i, zone in ipairs(zones) do
        print(zone.name)
    end
else
    print("Could not find any zones!")
end
```

Jump to Top

getBus

getBus(nameOrPosition)

Description

Function to retrieve the Bus object of a bus in the specified Element object. For example, `this.parent` specifies the parent of the script module as the Element object to be searched in. This function does not search in subelements. A particular bus can be searched by name or position. The position is the number indexing the busses in the specified Element object. If several busses share the same name, only the first match will be returned. If no argument is set, the function returns the first bus it finds.

Available in: Controller, Processor.

Arguments

nameOrPosition	The name or position of the bus. Set this to `nil` to deactivate the search filter.	string or number, optional

Return Values

Returns the Bus object of the found bus. Returns `nil` if no bus is found.

Example

```
-- locate the first bus in the program and print its name
bus = this.program:getBus()

if bus then
    print(bus.name)
else
    print("Could not find a bus!")
end
```

Jump to Top

getLayer

getLayer(nameOrPosition)

Description

Function to retrieve the Layer object of a layer in the specified layer. For example, `this.parent` specifies the parent layer of the script module as the layer to be searched in. The function does not search in sublayers. A particular layer can be searched by name or position. The position is the number indexing the layers in the specified layer. If several layers share the same name, only the first match will be returned. If no argument is set, the function returns the first layer it finds.

Available in: Controller, Processor

Arguments

nameOrPosition	The name or position of the layer. Set this to `nil` to deactivate the search filter.	string or number, optional

Return Values

Returns the Layer object of the found layer. Returns `nil` if no layer is found.

Example

```
-- locate the first layer in the program and print its name
layer = this.program:getLayer()

if layer then
    print(layer.name)
else
    print("Could not find a layer!")
end
```

Jump to Top

getMidiModule

getMidiModule(nameOrPosition)

Description

Function to retrieve the MidiModule object of a MIDI module in the specified layer. For example, `this.parent` defines the parent layer of the script module as the layer to be searched in. This function does not search in sublayers. A particular MIDI module can be searched by name or position. The position is the number indexing the MIDI modules in the specified layer. If several MIDI modules share the same name, only the first match will be returned. If no argument is set, the function returns the first MIDI module it finds.

Available in: Controller, Processor.

Arguments

nameOrPosition	The name or position of the MIDI module. Set this to `nil` to deactivate the search filter.	string or number, optional

Return Values

Returns the MidiModule object of the found MIDI module. Returns `nil` if no MIDI module is found.

Example

```
-- locate the first MIDI module in the program and print its name
module = this.program:getMidiModule()

if module then
    print(module.name)
else
    print("Could not find a MIDI module!")
end
```

Jump to Top

getZone

getZone(nameOrPosition)

Description

Function to retrieve the Zone object of a zone in the specified layer. For example, `this.parent` defines the parent layer of the script module as the layer to be searched in. This function does not search in sublayers. A particular zone can be searched by name or position. The position is the number indexing the zones in the specified layer. If several zones share the same name, only the first match will be returned. If no argument is set, the function returns the first zone it finds.

Available in: Controller, Processor.

Arguments

nameOrPosition	The name or position of the zone. Set this to `nil` to deactivate the name filter.	string or number, optional

Return Values

Returns the Zone object of the found zone. Returns `nil` if no zone is found.

Example

```
-- get the first zone in the program and print its name
zone = this.program:getZone()

if zone then
    print(zone.name)
else
    print("Could not find a zone!")
end
```

Jump to Top

insertBus

insertBus(bus, position)

Description

Function to insert a bus at the specified position in the destination layer. The bus to be inserted is determined by its Bus object. You can use getBus or findBusses to determine the bus. The destination layer is determined by its Layer object. For example, `this.parent` sets the parent layer of the script module as destination layer. The position is the number indexing the existing busses in the destination layer. The new bus will be inserted before the specified position. To add the bus at the end, use appendBus instead.

An Element object can only have one parent. It cannot be child of multiple parents. Therefore, an Element object you retrieved from the running plug-in instance must be removed before it can be inserted again. The Element objects that you retrieve through loadPreset or loadPresetAsync can be inserted freely, because these functions create a copy of the Element objects when reading them.

Available in: Controller.

Arguments

bus	The Bus object of the bus that you want to insert.	Bus
position	The position where the bus is inserted.	number

Example

To explore the following script:

1. Download Program.vstpreset.
2. Drag the preset on the MediaBay to import it to the user folder for VST presets.
3. Create an empty program and add a script module.
4. Paste the script into the text editor of the script module and execute the script.

```
-- insert bus from Program.vstpreset into the current program

-- get the file path for user VST presets
path = getUserPresetPath()

-- load VST preset
loadedProgram = loadPreset(path.."/Program/Program.vstpreset")

-- get the first bus from the loaded program
bus = loadedProgram:getBus()

-- insert bus
if bus then
    this.program:insertBus(bus, 1)
end
```

Jump to Top

insertLayer

insertLayer(layer, position)

Description

Function to insert a layer at a specific position in a destination layer. The layer to be inserted and the destination layer are both determined by their Layer objects. You can use getLayer or findLayers to determine the layer to be inserted. For example, `this.parent` determines the parent layer of the script module as destination layer. The position is the number indexing the existing layers in the destination layer. The new layer will be inserted before the specified position. To add the layer at the end, use appendLayer or appendLayerAsync instead.

An Element object can only have one parent. It cannot be child of multiple parents. Therefore, an Element object that you retrieved from the running plug-in instance must be removed before it can be inserted again. The Element objects you retrieve through loadPreset or loadPresetAsync can be inserted freely, because these functions create a copy of the Element objects when reading them.

Available in: Controller.

Arguments

layer	The Layer object of the layer that you want to insert.	Layer
position	The position where the layer is to bei inserted.	number

Example

To explore the following script:

1. Download Program.vstpreset.
2. Drag the preset on the MediaBay to import it to the user folder for VST presets.
3. Create an empty program and add a script module.
4. Paste the script into the text editor of the script module and execute the script.

```
-- insert layer from Program.vstpreset into the current program

-- get the file path for user VST presets
path = getUserPresetPath()

-- load VST preset
loadedProgram = loadPreset(path.."/Program/Program.vstpreset")

-- get the first layer from the loaded program
layer = loadedProgram:getLayer ()

-- insert layer
if layer then
    this.program:insertLayer(layer, 1)
end
```

Jump to Top

insertLayerAsync

insertLayerAsync(layer, position, callback)

Description

Function to insert a layer at a specified position in a destination layer using a separate, parallel thread. Inserting a layer in a separate thread can be necessary if the layer is too big to be inserted in a short time. The layer to be inserted and the destination layer are both determined by their Layer objects. You can use getLayer or findLayers to determine the layer to be inserted. For example, `this.parent` determines the parent layer of the script module as destination layer. The position is the number indexing the existing layers in the destination layer. The new layer will be inserted before the specified position. To add the layer at the end, use appendLayer or appendLayerAsync instead. The function returns a LoadProgress object that can be used to monitor the load progress. After the layer is inserted, the callback function is called. The callback function gets the LoadProgress object as default argument.

An Element object can only have one parent. It cannot be child of multiple parents. Therefore, an Element object that you retrieved from the running plug-in instance must be removed before it can be inserted again. The Element objects you retrieve through loadPreset or loadPresetAsync can be inserted freely, because these functions create a copy of the Element objects when reading them.

Available in: Controller.

Arguments

layer	The Layer object of the layer that you want to insert.	Layer
position	The position where the layer is to be inserted.	number
callback	Callback function that is called when the layer is inserted. The callback function gets the LoadProgress object as argument.	function, optional

Return Values

Returns a LoadProgress object.

Example

```
-- start with an empty program, remove all existing layers
layers = this.parent:findLayers()

if layers then
  for i, layer in ipairs(layers) do
      this.parent:removeLayer(layer)
  end
end

-- table with layer presets from Skylab
layerPresets = {
  { name = "Ambient Pad 01", path = "vstsound://EB86867EFF8E44FEA8FE366F676E25BE/.AppData/Steinberg/Skylab
/Sub Presets/Layer Presets/Ambient Pads/Ambient Pad 01.vstpreset" },
  { name = "Ambient Pad 02", path = "vstsound://EB86867EFF8E44FEA8FE366F676E25BE/.AppData/Steinberg/Skylab
/Sub Presets/Layer Presets/Ambient Pads/Ambient Pad 02.vstpreset" },
  { name = "Ambient Pad 03", path = "vstsound://EB86867EFF8E44FEA8FE366F676E25BE/.AppData/Steinberg/Skylab
/Sub Presets/Layer Presets/Ambient Pads/Ambient Pad 03.vstpreset" },
  { name = "Ambient Pad 04", path = "vstsound://EB86867EFF8E44FEA8FE366F676E25BE/.AppData/Steinberg/Skylab
/Sub Presets/Layer Presets/Ambient Pads/Ambient Pad 04.vstpreset" },
}

-- create table with the preset names
function getPresetNames()
  presetNames = {}
  for i, preset in ipairs(layerPresets) do
    presetNames[i] = preset.name
  end
end

getPresetNames()

-- remove the old layer after the new one was added
function removeOldLayer(progressInfo)
  local newPreset = progressInfo.root
  if oldPreset then
    this.parent:removeLayer(oldPreset)
    print(oldPreset.name.." removed.")
  end
  oldPreset = newPreset
end

-- insert the preset in a separate thread
function insertNewLayer(progressInfo)
  if progressInfo.root then
    this.parent:insertLayerAsync(progressInfo.root, 1, removeOldLayer)
    print("Inserting "..progressInfo.root.name.."...")
  end
end

-- load the preset in a separate thread
function onSelectPresetChanged(layerPreset)
  loadPresetAsync(layerPreset.path, insertNewLayer)
  print("Loading "..layerPreset.name.."...")
end

-- define a parameter for selecting the preset to be loaded
defineParameter("SelectPreset",  "Select Preset",  1, presetNames, function() onSelectPresetChanged
(layerPresets[SelectPreset]) end)
```

Jump to Top

insertMidiModule

insertMidiModule(module, position)

Description

Function to insert a MIDI module at the specified position in the determined destination layer. The MIDI module to be inserted is determined by its MidiModule object. You can use getMidiModule or findMidiModules to determine the desired MIDI module. The destination layer is determined by its Layer object. For example, `this.parent` determines the parent layer of the script module as destination layer. The position is the number indexing the existing MIDI modules in the destination layer. The new MIDI module will be inserted before the specified position. To add the MIDI module at the end, use appendMidiModule instead.

An Element object can only have one parent. It cannot be child of multiple parents. Therefore, an Element object that you retrieved from the running plug-in instance must be removed before it can be inserted again. The Element objects you retrieve through loadPreset or loadPresetAsync can be inserted freely, because these functions create a copy of the Element objects when reading them.

Available in: Controller.

Arguments

module	The MidiModule object of the MIDI module that you want to insert.	MidiModule
position	The position where the MIDI module is inserted.	number

Example

To explore the following script:

1. Download Program.vstpreset.
2. Drag the preset on the MediaBay to import it to the user folder for VST presets.
3. Create an empty program and add a script module.
4. Paste the script into the text editor of the script module and execute the script.

```
-- insert MIDI module from Program.vstpreset into the current program

-- get the file path for user VST presets
path = getUserPresetPath()

-- load VST preset
loadedProgram = loadPreset(path.."/Program/Program.vstpreset")

-- get the first MIDI module from the loaded program
module = loadedProgram:getMidiModule()

-- insert MIDI module
if module then
    this.program:insertMidiModule(module, 1)
end
```

Jump to Top

insertZone

insertZone(zone, position)

Description

Function to insert a zone at the specified position in the determined layer. The zone to be inserted is determined by its Zone object. You can use getZone or findZones to determine the desired zone. The destination layer is determined by its Layer object. For example, `this.parent` determines the parent layer of the script module as destination layer. The position is the number indexing the existing zones in the destination layer. The new zone will be inserted before the specified position. To add the zone at the end, use appendZone instead.

An Element object can only have one parent. It cannot be child of multiple parents. Therefore, an Element object that you retrieved from the running plug-in instance must be removed before it can be inserted again. The Element objects you retrieve through loadPreset or loadPresetAsync can be inserted freely, because these functions create a copy of the Element objects when reading them.

Available in: Controller.

Arguments

zone	The Zone object of the zone that you want to insert.	Zone
position	The position where the zone is inserted.	number

Example

To explore the following script:

1. Download Program.vstpreset.
2. Drag the preset on the MediaBay to import it to the user folder for VST presets.
3. Create an empty program and add a script module.
4. Paste the script into the text editor of the script module and execute the script.

```
-- insert zone from Program.vstpreset into the current program

-- get the file path for user VST presets
path = getUserPresetPath()

-- load VST preset
loadedProgram = loadPreset(path.."/Program/Program.vstpreset")

-- get the first zone from the loaded program
zone = loadedProgram:getZone()

-- insert zone
if zone then
    this.program:insertZone(zone, 1)
end
```

Jump to Top

removeBus

removeBus(busOrPosition)

Description

Function to remove a bus from the specified layer. For example, `this.parent` specifies the parent layer of the script module as the layer that contains the bus. The bus to be removed is determined by its Bus object or its position. You can use getBus or findBusses to determine the Bus object. The position is the number that indexes the busses in the specified layer.

Available in: Controller.

Arguments

busOrPosition	The Bus object or the position of the bus to be removed.	Bus or number

Example

```
-- remove all busses from the program

-- find the busses in the program
busses = this.program:findBusses(true)

-- and remove them
for i, bus in ipairs(busses) do
    this.parent:removeBus(bus)
end
```

Jump to Top

removeLayer

removeLayer(layerOrPosition)

Description

Function to remove a layer from the specified layer. For example, `this.parent` specifies the parent layer of the script module as the layer that contains the layer to be removed. The layer is determined by its Layer object or its position. You can use getLayer or findLayers to determine the Layer object. The position is the number indexing the layers within the specified layer.

Available in: Controller.

Arguments

layerOrPosition	The Layer object or the position of the layer to be removed.	Layer or number

Example

```
-- remove all layers from the program

-- find the layers in the program
layers = this.program:findLayers(true)

-- and remove them
for i, layer in ipairs(layers) do
    layer.parent:removeLayer(layer)
end
```

Jump to Top

removeMidiModule

removeMidiModule(moduleOrPosition)

Description

Function to remove a MIDI module from the specified layer. For example, `this.parent` specifies the parent layer of the script module as the layer that contains the MIDI module. The MIDI module to be removed is determined by its MidiModule object or its position. You can use getMidiModule or findMidiModules to determine the MidiModule object. The position is the number that indexes the MIDI modules in the specified layer.

Available in: Controller.

Arguments

moduleOrPosition	The MidiModule object or the position of the MIDI module to be removed.	MidiModule or number

Example

```
-- remove all MIDI modules from the program except the script module

-- find the MIDI modules in the program
modules = this.program:findMidiModules(true)

-- and remove them
for i, module in ipairs(modules) do
        if module ~= this then -- exclude script module
            module.parent:removeMidiModule(module)
        end
end
```

Jump to Top

removeZone

removeZone(zoneOrPosition)

Description

Function to remove a zone from the specified layer. For example, `this.parent` specifies the parent layer of the script module as the layer that contains the zone. The zone to be removed is determined by its Zone object or its position. You can use getZone or findZones to determine the Zone object. The position is the number that indexes the zones in the specified layer.

Available in: Controller.

Arguments

zoneOrPosition	The Zone object or position of the zone to be removed.	Zone or number

Example

```
-- remove all zones from the program

-- find the zones in the program
zones = this.program:findZones(true)

-- and remove them
for i, zone in ipairs(zones) do
    zone.parent:removeZone(zone)
end
```

Jump to Top

addQCAssignment

addQCAssignment(qc, element, nameOrID, scope)

Description

Function to add a quick control assignment to the specified layer and quick control. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control that you want to edit. The quick control assignment will be added to the quick control with the index stated by the `qc` argument. The arguments `element` and `nameOrID` specify the parameter to be connected. The scope determines the part of the program that will be affected by the quick control assignment. You specify the scope by setting the `scope` argument to the Element object that corresponds to the desired part of the program.

The index of the quick controls starts counting from 1. QC 1 to QC 8 have index 1 to 8. Sphere H, Sphere V and Mod Wheel have index 9, 10 and 11.

Available in: Controller.

Arguments

qc	The index of the quick control to which the assignment will be added.	number
element	The Element object of the parameter to be connected.	Element
nameOrID	The name or ID of the parameter.	string or number
scope	The Element object that will be affected by the quick control assignment.	Element

Example

```
-- assign the octave parameter of the zone to the first quick control of the script module's parent layer
layer = this.parent
zones = layer:findZones(true)

layer:addQCAssignment(1, zones[1], "Pitch.Octave", layer)
```

Jump to Top

removeQCAssignment

removeQCAssignment(qc, assignment)

Description

Function to remove a quick control assignment from the specified layer and quick control. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control with the assignment to be removed. The `assignment` argument is the index of the quick control assignment to be removed. The indices of the quick controls and the assignments both start counting from 1.

Available in: Controller.

Arguments

qc	The index of the quick control with the assignment to be removed.	number
assignment	The index of the quick control assignment to be removed.	number

Example

```
-- remove all quick control assignments from the specified layer and qc

function clearQC(layer, qc)
	local assignments = layer:getNumQCAssignments(qc)
	if assignments > 0 then
		for i = assignments, 1, -1 do
			layer:removeQCAssignment(qc, i)
		end
		print(assignments.." assignments have been removed from '"..layer.name.."', QC "..qc..".")
	else
		print("No assignments found on '"..layer.name.."', QC "..qc..".")
	end
end

clearQC(this.parent, 1)
```

Jump to Top

getNumQCAssignments

getNumQCAssignments(qc)

Description

Function to retrieve the number of assignments of a quick control on the specified layer. For example, `this.parent` defines the parent layer of the script module as the layer with the desired quick control. The `qc` argument is the index of the quick control with the requested assignments.

The index of the quick controls starts counting from 1. QC 1 to QC 8 have the index values 1 to 8. Sphere H, Sphere V and Mod Wheel have the index values 9, 10 and 11.

Available in: Controller.

Arguments

qc	The index of the quick control.	number

Return Values

Returns the number of assignments of the specified layer and quick control.

Example

```
-- print the number of assignments of the first quick control on the parent layer
layer = this.parent
qc = 1

print("Number of assignments on '"..layer.name.."', QC "..qc..": "..layer:getNumQCAssignments(qc).."."")
```

Jump to Top

getQCAssignmentParamId

getQCAssignmentParamId(qc, assignment)

Description

Function to retrieve the parameter ID of the parameter that is connected to the specified quick control assignment. The quick control assignment is determined by the Layer, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment with the requested parameter. The indices of the quick controls and the assignments both start counting from 1.

Available in: Controller.

Arguments

qc	The index of the quick control.	number
assignment	The index of the quick control assignment.	number

Return Values

Returns the parameter ID of the parameter connected to the specified quick control assignment.

Example

```
-- print the parameter ID of the parameter connected to the qc assignment
layer = this.parent
qc = 1
assignment = 1

paramID = layer:getQCAssignmentParamId(qc, assignment)

print("Parameter ID of '"..layer.name.."', QC "..qc.."", assignment "..assignment..": "..paramID..".")
```

Jump to Top

getQCAssignmentScope

getQCAssignmentScope(qc, assignment)

Description

Function to retrieve the element object that is set as scope for the specified quick control assignment. The quick control assignment is determined by the Layer object of the layer, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment with the requested scope. The indices of the quick controls and the assignments both start counting from 1.

Available in: Controller.

Arguments

qc	The index of the quick control.	number
assignment	The index of the quick control assignment.	number

Return Values

Returns the element object that is set as scope for the specified quick control assignment.

Example

```
-- print the scope for the qc assignment
layer = this.parent
qc = 1
assignment = 1

scope = layer:getQCAssignmentScope(qc, assignment).name -- use only the name of the returned element

print("Scope of '"..layer.name.."', QC "..qc.."", assignment "..assignment..": "..scope..".")
```

Jump to Top

getQCAssignmentMin

getQCAssignmentMin(qc, assignment)

Description

Function to retrieve the minimum value of the specified quick control assignment. The quick control assignment is determined by the Layer, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment with the requested minimum value. The indices of the quick controls and the assignments both start counting from 1.

Available in: Controller.

Arguments

qc	The index of the quick control.	number
assignment	The index of the quick control assignment.	number

Return Values

Returns the minimum value of the specified quick control assignment.

The value range is always 0 to 100 %, even if the mode of the quick control assignment is set to Relative or Switch Relative.

Example

```
-- print the minimum value of the qc assignment as displayed in HALion
layer = this.parent
qc = 1
assignment = 1

qcType = layer:getQCAssignmentType(qc, assignment)
qcMin = layer:getQCAssignmentMin(qc, assignment)

-- convert to bipolar range if the qc is of the type relative or switch relative
if (qcType == QCAssignmentType.relative or qcType == QCAssignmentType.switchRelative) then
    qcMin = qcMin * 2 - 100
end

print("Minimum value of '"..layer.name.."', QC "..qc..", assignment "..assignment..": "..qcMin..".")
```

Jump to Top

getQCAssignmentMax

getQCAssignmentMax(qc, assignment)

Description

Function to retrieve the maximum value of the specified quick control assignment. The quick control assignment is determined by the Layer, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment with the requested maximum value. The indices of the quick controls and the assignments both start counting from 1.

Available in: Controller.

Arguments

qc	The index of the quick control.	number
assignment	The index of the quick control assignment.	number

Return Values

Returns the maximum value of the specified quick control assignment.

The value range is always 0 to 100 %, even if the mode of the quick control assignment is set to Relative or Switch Relative.

Example

```
-- print the maximum value of the qc assignment as displayed in HALion
layer = this.parent
qc = 1
assignment = 1

qcType = layer:getQCAssignmentType(qc, assignment)
qcMax = layer:getQCAssignmentMax(qc, assignment)

-- convert to bipolar range if the qc is of the type relative or switch relative
if (qcType == QCAssignmentType.relative or qcType == QCAssignmentType.switchRelative) then
    qcMax = qcMax * 2 - 100
end

print("Maximum value of '"..layer.name.."', QC "..qc.."", assignment "..assignment..": "..qcMax..".")
```

Jump to Top

getQCAssignmentCurve

getQCAssignmentCurve(qc, assignment)

Description

Function to retrieve the curve value of the specified quick control assignment. The quick control assignment is determined by the Layer object, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment with the requested curve value. The indices of the quick controls and the assignments both start counting from 1.

Available in: Controller.

Arguments

qc	The index of the quick control.	number
assignment	The index of the quick control assignment.	number

Return Values

Returns the curve value of the specified quick control assignment.

The value range is -100 % to +100 %.

Example

```
-- print the curve value of the qc assignment
layer = this.parent
qc = 1
assignment = 1

qcCurve = layer:getQCAssignmentCurve(qc, assignment)

print("Curve value of '"..layer.name.."', QC "..qc..", assignment "..assignment..": "..qcCurve..".")
```

Jump to Top

getQCAssignmentMode

getQCAssignmentMode(qc, assignment)

Description

Function to retrieve the mode that is set for the specified quick control assignment. The quick control assignment is determined by the Layer object, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment with the requested mode. The indices of the quick controls and the assignments both start counting from 1.

Available in: Controller.

Arguments

qc	The index of the quick control.	number
assignment	The index of the quick control assignment.	number

Return Values

Returns the mode that is set for the specified quick control assignment as a number. The mode can be determined via names or indices. See Quick Control Assignment Modes for details.

Example

```
-- print the mode of the qc assignment
layer = this.parent
qc = 1
assignment = 1

qcMode = layer:getQCAssignmentMode(qc, assignment)

if qcMode == QCAssignmentMode.absolute then
    qcModeName = "Absolute"
elseif qcMode == QCAssignmentMode.relative then
    qcModeName = "Relative"
elseif qcMode == QCAssignmentMode.switch then
    qcModeName = "Switch"
elseif qcMode == QCAssignmentMode.switchRelative then
    qcModeName = "Switch Relative"
end

print("Mode of '"..layer.name.."', QC "..qc.., assignment "..assignment..": "..qcModeName..".")
```

Jump to Top

getQCAssignmentBypass

getQCAssignmentBypass(qc, assignment)

Description

Function to retrieve the bypass state of the specified quick control assignment. The quick control assignment is determined by the Layer object, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment with the requested bypass state. The indices of the quick controls and the assignments both start counting from 1.

Available in: Controller.

Arguments

qc	The index of the quick control.	number
assignment	The index of the quick control assignment.	number

Return Values

Returns the bypass state of the specified quick control assignment as boolean value.

Example

```
-- print the bypass state of the qc assignment
layer = this.parent
qc = 1
assignment = 1

qcBypass = layer:getQCAssignmentBypass(qc, assignment)

print("Bypass of '"..layer.name.."', QC "..qc..", assignment "..assignment..": "..tostring(qcBypass)..".")
```

Jump to Top

setQCAssignmentParamId

setQCAssignmentParamId(qc, assignment, paramID)

Description

Function to set the parameter ID for connecting the corresponding parameter to the specified quick control assignment. The quick control assignment is determined by the Layer object, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment. The indices of the quick controls and the assignments both start counting from `1`. The `paramID` argument selects the parameter to be connected with the quick control assignment.

Available in: Controller

Arguments

qc	The index of the quick control.	number
assignment	The index of the quick control assignment.	number
paramID	The ID of the parameter to be connected.	number

Example

```
-- connect the coarse parameter of the zone to the specified quick control assignment
layer = this.parent
zones = layer:findZones(true)
zone = zones[1]
qc = 1
assignment = 1
coarseParamID = zone:getParameterDefinition("Pitch.Coarse").id

layer:setQCAssignmentParamId(qc, assignment, coarseParamID)
```

Jump to Top

setQCAssignmentScope

setQCAssignmentScope(qc, assignment, scope)

Description

Function to set the scope for the specified quick control assignment. The quick control assignment is determined by the Layer object, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment. The indices of the quick controls and the assignments both start counting from 1. The scope is defined by the Element object that you assign to the `scope` argument.

Available in: Controller.

Arguments

qc	The index of the quick control.	number
assignments	The index of the quick control assignment.	number
scope	The Element object to be used as scope.	Element

Example

```
-- set the scope to the first zone that was found
layer = this.parent
zones = layer:findZones(true)
zone = zones[1]
qc = 1
assignment = 1

layer:setQCAssignmentScope(qc, assignment, zone)
```

Jump to Top

setQCAssignmentMin

setQCAssignmentMin(qc, assignment, min)

Description

Function to set the minimum value of the specified quick control assignment. The quick control assignment is determined by the Layer object, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment. The indices of the quick controls and the assignments both start counting from 1. The `min` argument sets the minimum value of the quick control assignment.

Available in: Controller.

Arguments

qc	The index of the quick control.	number
assignment	The index of the quick control assignment.	number
min	The minimum value to be set.	number

The value range of the minimum value is always 0 to 100 %, even if the type of the quick control assignment is set to Relative or Switch Relative.

Example

```
-- set the minimum value of the quick control assignment depending on the type
layer = this.parent
qc = 1
assignment = 1

qcType = layer:getQCAssignmentType(qc, assignment)

if (qcType == QCAssignmentType.relative or qcType == QCAssignmentType.switchRelative) then
    qcMin = 25
else
    qcMin = 0
end

layer:setQCAssignmentMin(qc, assignment, qcMin)
```

Jump to Top

setQCAssignmentMax

setQCAssignmentMax(qc, assignment max)

Description

Function to set the maximum value of the specified quick control assignment. The quick control assignment is determined by the Layer object, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment. The indices of the quick controls and the assignments both start counting from 1. The `max` argument sets the maximum value of the quick control assignment.

Available in: Controller

Arguments

qc	The index of the quick control.	number
assignment	The index of the quick control assignment.	number
max	The maximum value to be set.	number

The value range of the maximum value is always 0 to 100 %, even if the type of the quick control assignment is set to Relative or Switch Relative.

Example

```
-- set the maximum value of the quick control assignment depending on the type
layer = this.parent
qc = 1
assignment = 1

qcType = layer:getQCAssignmentType(qc, assignment)

if (qcType == QCAssignmentType.relative or qcType == QCAssignmentType.switchRelative) then
    qcMax = 75
else
    qcMax = 100
end

layer:setQCAssignmentMax(qc, assignment, qcMax)
```

Jump to Top

setQCAssignmentCurve

setQCAssignmentCurve(qc, assignment, curve)

Description

Function to set the curve value of the specified quick control assignment. The quick control assignment is determined by the Layer object, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment. The indices of the quick controls and the assignments both start counting from 1. The `curve` argument sets the curve value of the quick control assignment.

Available in: Controller.

Arguments

qc	The index of the quick control.	number
asignment	The index of the quick control assignment.	number
curve	The curve value in the range -100 % to +100 %.	number

Example

```
-- set the curve of the quick control assignment to -100 %
layer = this.parent
qc = 1
assignment = 1

layer:setQCAssignmentCurve(qc, assignment, -100)
```

Jump to Top

setQCAssignmentMode

setQCAssignmentMode(qc, assignment, mode)

Description

Function to set the mode of the specified quick control assignment. The quick control assignment is determined by the Layer object, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment. The indices of the quick controls and the assignments both start counting from 1. The `mode` argument sets the mode of the quick control assignment.

Available in: Controller

Arguments

qc	The index of the quick control.	number
asignment	The index of the quick control assignment.	number
mode	The mode to be set. It can be determined via names or indices. See Quick Control Assignment Modes for details.	enum or number

Example

```
-- set the mode of the quick control assignment to absolute and adjust min and max to full range
layer = this.parent
qc = 1
assignment = 1

layer:setQCAssignmentType(qc, assignment, QCAssignmentType.absolute)
layer:setQCAssignmentMin(qc, assignment, 0)
layer:setQCAssignmentMax(qc, assignment, 100)
```

Jump to Top

setQCAssignmentBypass

setQCAssignmentBypass(qc, assignment, bypass)

Description

Function to set the bypass state of the specified quick control assignment. The quick control assignment is determined by the Layer object, the index of the quick control and the index of the assignment itself. For example, `this.parent` defines the parent layer of the script module as the layer that contains the quick control. The `qc` argument is the index of the quick control and the `assignment` argument is the index of the assignment. The indices of the quick controls and the assignments both start counting from 1. The `bypass` argument sets the bypass state of the quick control assignment.

Available in: Controller.

Arguments

qc	The index of the quick control.	number
assignment	The index of the quick control assignment.	number
bypass	The bypass state to be set.	boolean

Example

```
-- bypass the specified quick control assignment
layer = this.parent
qc = 1
assignment = 1

layer:setQCAssignmentBypass(qc, assignment, true)
```

Jump to Top