**MXwendler**
**Javascript Interface Description Version 2.3**

This document describes the MXWendler (MXW) Javascript Command Interface. You will learn how to control MXwendler through the Javascript interface. We will describe parts of the hierarchical control approach and how to create interactive Javascripts in MXWendler.

05.12, valid for builds > 1900

# Contents



## What is Javascript?

Javascript, also known as ECMA Script is a standardized interpreted programming language. It is mainly used in webbrowsers to control web pages and test user input before sending.

## Why could i control MXWendler using Javascript?

Although there are many ways to control MXWendler with a lot of I/O devices, beginning with the mouse, the keyboard, DMX dongles and MIDI devices, and there is a broad automation layer in MXWendler using events and playlists, some desired behaviour is simply too complex to be offered through a GUI element. **Javascript allows you to program any animation** and behaviour sequence. Additionally, Javascript allows you to **emit values to I/O devices like DMX or MIDI** ( please check your version for this functionality ) and set various internal states in media, eg. set variables and exchange text strings in SWF files.

## Where do i enter Javascript Scripts?

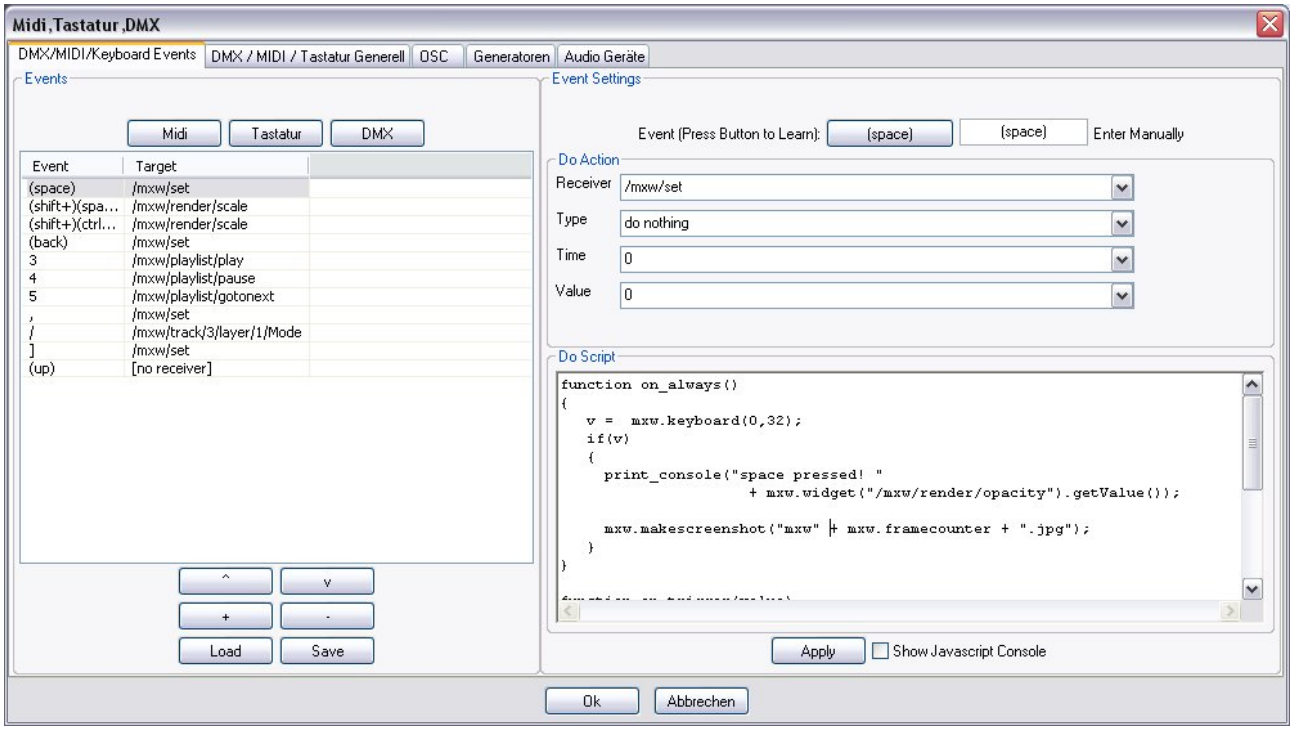
Scripts reside always close to the events. Scripts have three entry points:

```
void on_trigger ( triggervalue )
{
      //
      // this function will be called
      // once after an event occurred
      // and before drawing the frame
      // triggervalue is the normalized
      // value that the mapped control
      // device emits:
      // keyboard ( press/release )    [1:0]
      // midi 0 .. 127                 [0 .. 1]
      // dmx 0 .. 255                  [0 .. 1]
}

float on_repeat(void)
{
      //
      // this function will be called
      // the next frame after on_trigger
      // has been called.
      // this function will be called
      // repeatedly as long as  the function
      // returns a positive value
      // this enables you to create functions that
      // last for a certain time: if you want to
      // create a javascript fade for eg. 10 seconds,
      // create a variable outside this function,
      // give it a value of eg. 250 in on_trigger
      // decrement it by one in on_repeat and return it
}

void on_always(void)
{
      //
      // this function will be called
      // once each frame regardless
      // of any inner or outer state.
      // be careful with this function,
      // the only way to turn it off
      // is to remove this script and
      // reload the event map
}
```

## Example script: called once on trigger, repeat for 10 seconds

1. Create an event for the space bar
2. Copy and paste this script into the script area of an event.
3. Close events, confirm changes
4. Open javascript console ( ctrl+shift+j )
5. Trigger the space bar

```
var a;
function on_trigger( triggervalue )
{
      a = 250;
      print_console("on_trigger called with " + triggervalue);
}

function on_repeat()
{
      print_console("on_repeat called with " + a--);
      return a;
}
```



## Which interface elements can i control using Javascript?

You can control any element you can adress using the standard hierarchical MXWendler access pattern. To access MXWendler elements, you access the static object "mxw" - it is an object like the javascript Math object, which is always available. To faciliate debugging, a script-wide function "print_console" is available, too.

```
function on_trigger( triggervalue )
{
	print_console("on_trigger called with " + triggervalue);
}
```

The object "mxw" offers the following members and properties:

```
function on_trigger(value)
{
	// print various states
	print_console("current time " + mxw.millis );

	print_console("current frame width " + mxw.width );

	print_console("current frame height " + mxw.height );

	print_console("current output width " + mxw.outwidth );

	print_console("current output height " + mxw.outheight );

	print_console("current framecounter " + mxw.framecounter );

	//
	// read-access to IO devices
	print_console("current dmx value at channel 413 "
			+ mxw.dmx(413)
			);
	print_console("current midi value device 1 channel 10 "
			+ mxw.midi(1,10)
			);
	print_console("current keyboard value for (shift)+(space) "
			+ mxw.keyboard(1,32)
			);

	//
	// make a screenshot
	print_console("make a screenshot named screen.jpg "
			+ mxw.screenshot("screen.jpg")
			);


	//
	// access widgets and control and get/set values
	print_console("access opacity control of main render output "
			+ mxw.widget("/mxw/render/opacity")
			);
```

```
        print_console("main render opacity value is "
                        + mxw.widget("/mxw/render/opacity").getValue()
                        );

        print_console("main render opacity: set 0.5 value");
        mxw.widget("/mxw/render/opacity").setValue(0.5);

    }
```

## Example Script 1: Print Frame Number, Make Screenshot, Start Clip

This script will:

- always print the current frame number
- on trigger, create a screenshot
- 2 seconds after triggering load the set with number 10

```
var load_set_at;

function on_always()
{
	//
	// read current frame and print to console
	print_console("current frame: " +  mxw.framecounter );
}

function on_trigger(value)
{
	//
	// create screenshot with name like "mxw_20125.jpg"
	makescreenshot("mxw_" + mxw.millis + ".jpg");

	//
	// store the triggering point in time,
	// add time distance ( 2000 msec = 2 seconds )
	load_set_at = mxw.millis + 2000;
}

function on_repeat()
{
	//
	// this function will be called
	// each frame after on_trigger as
	// log as it returns a value higher
	// than 0
	// we calc the time distance and
	// use the result to trigger
	// loading a set

	ret = load_set_at - mxw.millis;

	if(ret>0)
	{
		//
		// we are early, return value is positive,
		// we will be called again
		return ret;
	}
	else
	{
		//
		// ret is lower 0, more than
		// two seconds since trigger
		// so start action
```

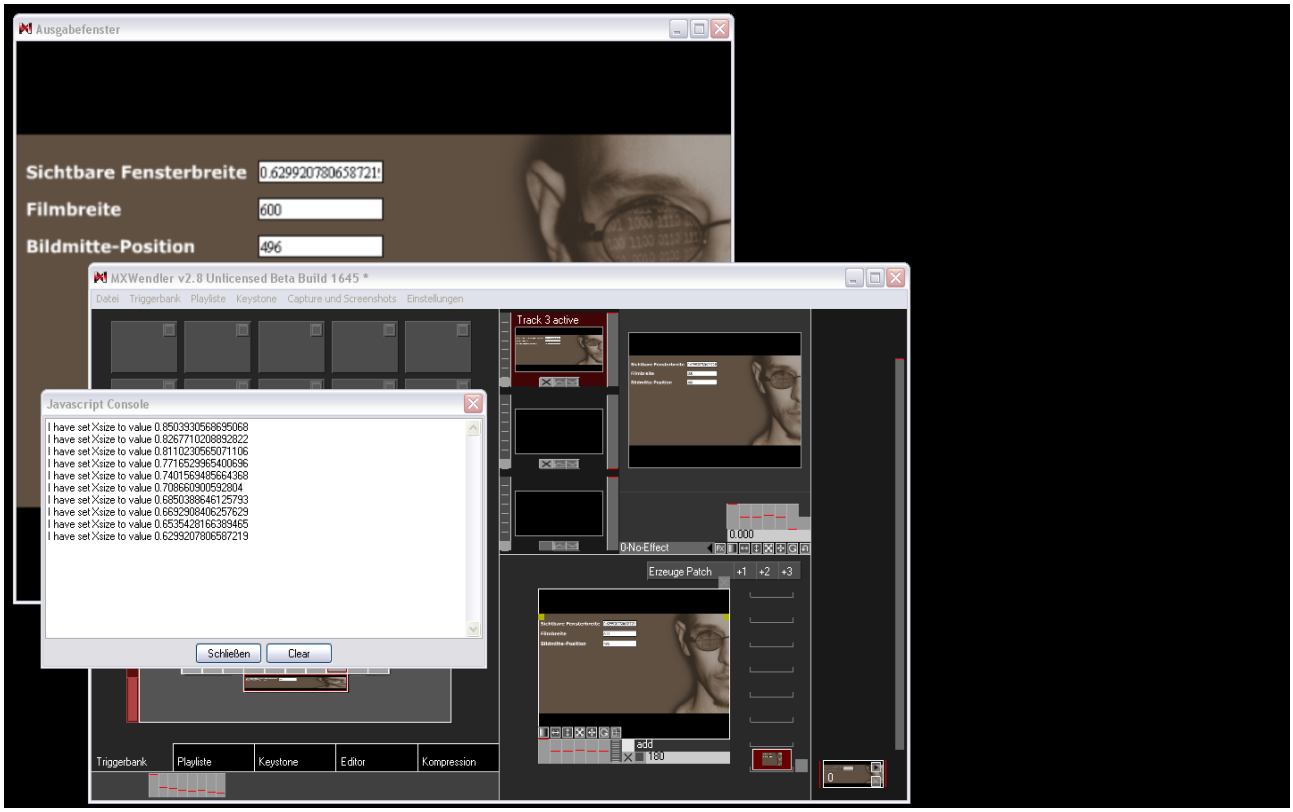
```
            //
            // let js resolve widget, then
            // send value as command
            mxw.widget("/mxw/set").setValue(10);

            //
            // return 0: we are done
            return 0;
        }

    }
```

## Example Script 2: Reading/Setting SWF Flash Media Variables

This script will:

- check for a certain clip
- check for its media
- set a variable
- read that variable and print it to the console

```
//
// only on_trigger is needed
function on_trigger(value)
{
   //
   // get reference of clip
   // do *not* store this value, it is
   // higly dynamic and may change each frame!
   w = mxw.widget("/mxw/track/active/layer/active/clip");

   //
   // always check for existence
   if(w)
   {
      //
      // get media ref. Do *not* store this value, too.
      m = w.media();
       if(m)
       {
          //
          // the name of the flash variable.
          // you have to set this name in your swf
          // creation tool.
          var swfvar = "Xsize";

          //
          // set the value, always passed as string.
          // you can set numeric values here or pass
          // names, titles, etc.
          m.setVariable(swfvar,value);

          //
          // get the value, always returned as string
          print_console( "I have set "+ swfvar + " to value " +
             m.getVariable(swfvar)
          );
       }
   }
}
```




*Setting SWF Variables through the Javascript Interface*

## Static functions

Static functions can be inserted at any place in the script field. They do not rely on any objects.

```
void print_console( "hello world" )
```

Purpose

Debugging help: print a value to the javascript console.

Arguments

String

Return value

None

Example

```
function on_tigger(value)
{
      print_console("on_trigger called with value " + val);
}
```

```
void on_trigger ( triggervalue )
```

Purpose

Called by the main application whenever a triggering event happened

Arguments

A float value representing the current value of the triggering event.

Keyboards emit either 1 or zero, [1:0],

Midi values 0 .. 127 are scaled to [0 .. 1],

DMX values 0 .. 255 are scaled to [0 .. 1] as well

Return value

None

Example

```
function on_trigger(val)
{
      print_console("on_trigger called with value " + val);
}
```

```
float on_repeat(void)
```

Purpose

Called by the main application on each frame a) after on_trigger has been called once and b) as long as the function returns a positive value.
With this function you can keep the scipt execution alive after an event happened.

Arguments

None

Return value

Float. If the value is greater than zero, the function will be called on the next frame again

Example

```
function on_repeat()
{
print_console("this will be called once after on_trigger");
return 0;
}
```

```
void on_always(void)
```

Purpose

Called by the main application on each frame after on_trigger has been called. This is a convenience function as it is exactly the same as on_repeat() returning a positive value.

Arguments

None

Return value

None.

Example

```
function on_always()
{
     print_console("
          this will be called on each frame once\
          on_trigger() has been called.\
          There is no way to stop execution until\
          the event is redefined or the application is\
          stopped. ");
}
```

## Objects

There are certain objects available for MXW javascript scripting. They provide access to application specific functions and values.

There is one static object, "mxw" that does not need special initialization and is always available. It is the accessor to state values ( framecounter etc. ) and active objects, like clips and the like. There are two other object types, widget and media. A "widget" object represents a unique widget which is available in the User Interface. A "media" object represents the media which is loaded in a clip.

### mxw

Purpose

A static object representing the main application. Does not have to be initialized.

Functions

```
// will return the dmx value of channel xy
mxw.dmx( channel );

// will send the dmx value to channel xy
mxw.send_dmx( channel, value );

// will return the midi value of device x channel y
mxw.midi( device,channel );

// will send the midi value to device x channel y
mxw.send_midi( device,channel,value );

// will return the keyboard value of key (char) in
// state 0/1/2/3 (none/shift/ctrl/shft+ctrl)
mxw.keyboard( state, char );

// 1. servername, ip "192.168.0.2"
// 2. portnumber
// 3. Adresspattern
// 4. d: dbl, i:int, c:string, b:blob
// 5... arguments. Up to 15 arguments are allowed
// each argument must have a type letter in arg 4
mxw.send_osc( "localhost", 7000, "/mxw/reply", "ddicc",
                "2.0", "223349.335", "11", "hello" "world );

// will make a screenshot and save it as screen.jpg.
// the scrrenshot will have the current render size
mxw.screenshot("screen.jpg");

// will return a widget object corresponding to the
// identifying string. This object can be used for further
// actions and queries
mxw.widget("/mxw/render/opacity");
```

Properties

```
// milliseconds since software start
mxw.millis

// current gui frame width
mxw.width

// current gui frame height
mxw.height

// current output frame width
mxw.outwidth

// current output frame height
mxw.outheight

// framenumber since software start
mxw.framecounter
```

## widget

### Purpose

This javascript object is a proxy to a corresponding object in the aplication gui. It can be used for further action and queries. The object must be initialized using mxw.widget() and should not be stored due to the dynamic nature of the user interface

### Functions

```
bool widget.isvalid();
```

Returns true if the widget object has been initialized properly. Note that the corresponding real widget may have been deleted meanwhile, so it is good style to resolve the widget at each access ( resolving is very fast )

Good:

```
mxw.widget("/mxw/...").isvalid();
```

Bad:

```
w = mxw.widget("/mxw/...");
( .. some frames later )
w.isvalid(); // access stored reference
```

```
float widget.getValue();
```

Returns the current value of the widget in a 0 .. 1 range.

```
void widget.setValue(float);
```

Sets the current value of the widget to the specified value.

```
media widget.media();
```

Returns the current media of the widget for further actions and queries. Only successful for clips and preloads.

```
void widget.animate
            ( initvalue, incomingvalue, duration, delta, type);
```

Triggers an animation for the widgets value. This behaves exactly as the triggerevents. Type must be set as an integer, corresponding to the available behaviour types ( pass value, go to and loop etc ) Note that sometimes some values may not be used ( eg. incomingvalue when using 'random' type ) but some dummy values must be passed always.

### Properties

None

## media

### Purpose

This javascript object is a proxy to a corresponding media object in the aplication gui. It can be used for further action and queries. The object must be initialized using mxw.widget("/path/to/clip").media() and should not be stored due to the dynamic nature of the user interface

### Functions

```
string media.getMediaName();
```

Returns the full path and filename of the media.

```
string media.getVariable(string "var_name" );
```

Returns the current value of the specified variable. Only useful for swf media.

```
void widget.setVariable(string "var_name", string "new_val");
```

Sets the current value of the specified variable. Only useful for swf media.

### Properties

None

## grabber

### Purpose

This javascript object is a proxy to a corresponding DMX or Artnet grabber object in the keystone area. To get a handle to a grabber object, name the grabber in the keystone setup and get the handle with

```
mxw.grabber(„grabbername“)
```

### Functions

```
float getGrabSizeX();
float getGrabSizeY();
```

Returns the size of the grabber in pixels

```
int[] getData();
```

Returns a data array with the grabbed values. Can be sent as a blob by OSC, see mxw.send_osc for details.

### Properties

None