V21.13.0 | 15th October 2021

IWS 21 User Guide |

V21.13.0 | 15th October 2021 | IME | ISE | Pak | Wotja

Powerful Music Engine Scripting

***Note*: "Generators" is the name used in Wotja 21, for what was referred to as "Voices" in older versions.**

Wotja 21 contains a powerful scripting engine which, while entirely optional to use. It is especially useful when creating "Adaptive Music".

The Wotja Script Editor is accessed from Wotja as follows:

- Mix Scripts: Wotja Music Edit Mode > Mix Properties > Script
- Cell Scripts: Wotja Music Edit Mode > Cell Properties > Script
- Generator Scripts: Wotja Mix Edit Mode > Generator Network Editing Panel > Generator (AKA Generator) > Scripting

The Wotja Script Sandbox is accessed from Wotja as follows:

- Wotja Music Edit Mode > Action Menu > Script Sandbox

Wotja Scripting is aimed at generative music fans who want to "hack" Wotja, coders and hackers who want to play with music, and anybody who uses Wotja and wants to push the outer limits of what Wotja can do out of the box.

Tip for iOS Users: To get a "standard iOS" text editor with copy/paste etc, just select Action menu > Edit... You can then use normal iOS operations to copy/paste etc.

The Wotja Script language is based on the widely used ECMAScript scripting langauge (ECMAScript E5/E5.1). ECMAScript (also referred to commonly by the name JavaScript), is very powerful, fast and easy to learn. ECMAScript is a very popular embedded scripting langage, and is widely used in the web, gaming and multimedia worlds. Wotja adds various Wotja-specific functions to the language. These extensions are outlined later in this document; we call them *"Event Handler Script Functions"* or *"Triggers"*. We give you lots of examples to get you started.

Wotja Scripts and Wotja Scripting (hereafter just "scripts", scripting etc.) can be used in two separate contexts:

Wotja Scripts and Wotja Scripting (hereafter just "scripts", scripting etc.) can be used in two separate contexts:

- **Event Handler Script Functions (*"Triggers"*)**

  The most common way to use scripting in Wotja is to use the Script Editor Window accessed from the IME Generator Scripts and Cell Scripts and Mix Scripts cells. This allows you to create Event Handler Script Functions (*"Triggers"*), which are small bits of code in the widely used ECMAScript language. The specially named Event Handler Script Functions in these scripts are triggered when various events happen when a mix is playing. Using Event Handler Script Functions allows you to tell the Generator or Mix Objects to behave in very powerful ways while the mix is playing.

- **Sandbox Scripts**

  You can use the Wotja Sandbox script window to experiment with Script code. You can show the Wotja Sandbox from the main Mix screen (press the Action menu button, and select the *Script Sandbox* item).

Why would you want to use Scripting?

Are you a generative music fan who wants to see what your own custom code can achieve when leveraging a hugely powerful app such as Wotja? Are you code hacker who want to play with music? Are you a Wotja expert who wants to push the outer limits of what Wotja can do out of the box?

With careful use of Wotja's Event Handler Script Functions, you can do a lot of cool things, while turning Wotja into a custom hyperinstrument.

You can, if you wish, create Event Handler Script Functions to run in response to the following events:

- start of playback
- every bar
- when a note is composed by the IME
- end of playback
- in response to external MIDI CC via e.g. keyboard controller
- in response to external MIDI Note on/off via e.g. keyboard controller

Every script can manipulate a large number of things in Wotja in real-time. Some examples: manipulating music rules, playing with mutation, altering patterns; and what have you.

By creating code to respond to external MIDI events, you can in turn manipulate various properties of Wotja, such as auto-chording parameters, rule settings etc.

User Interface Updates

If you change a property with a script function such as *imeParameterSet*, the user interface doesn't always update to reflect your call; you'd need to refresh the table that displays your updated parameter, to see the change in the user interface. We hope to be able to improve this behaviour at some future point.

If you change a parameter with the user interface, and that parameter is later changed by a script; the latter script call will override your user interface change. Don't get confused by this!

If you have a script which copies a parameter value at start, and then resets it when the mix stops; where you've in between changed the parameter through the user interface; the latter script call will override your user interface change. Don't get confused by this!

Latency and Timing

Latency and Timing

The IME composes events a little bit in advance of when you'll hear them. This is to ensure that the notes are composed, in time for them to be picked-up by the ISE; the system also does the audio processing required by the ISE a little ahead of time before you actually hear it. Added together, this means that script code run in response to, say, *onImeBar* won't always appear precisely synchronized to what you're listening to!

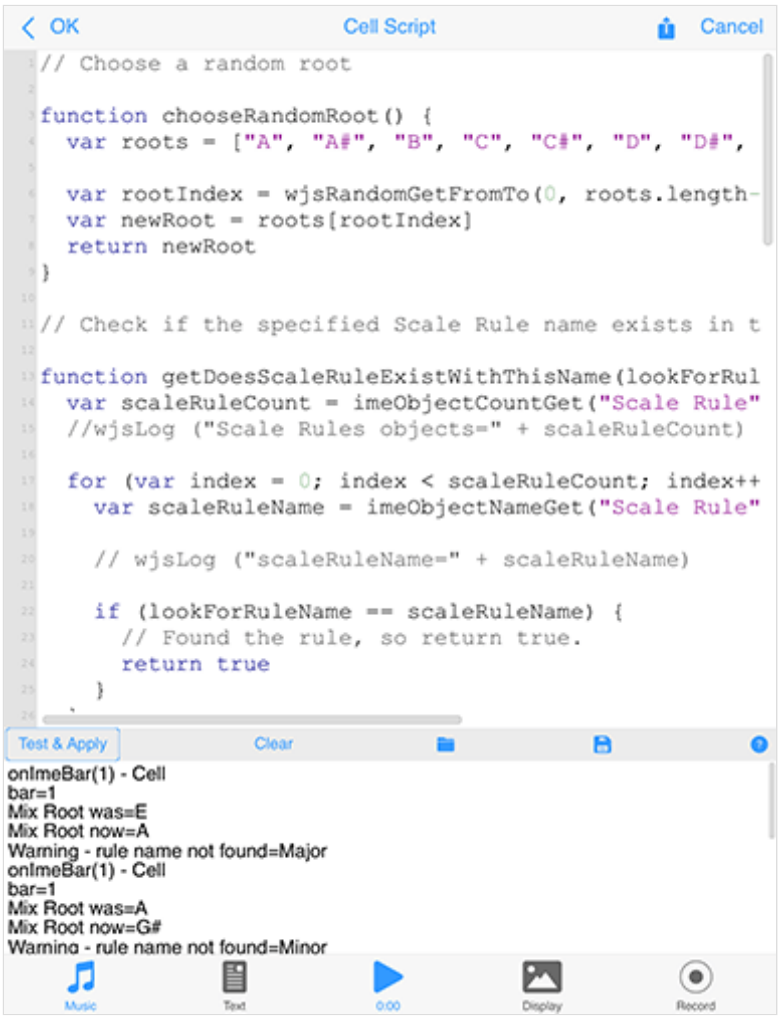
Infinite Loops

Please be careful not to implement so-called *Infinite Loops* in your script code, as that might cause the Wotja application to crash.

Recursion (recursive function calls)

Please be careful not to write script code that results in a lot of *Recursion*, as that might cause the Wotja application to crash.

Scripting Interface



The Wotja Script Editor Window

When you decide to edit a *Mix Script* or *Cell Script* or *Generator Script* parameter, or if you select the *Script Sandbox* menu item, you are presented with the *Script Editor* window. It supports syntax-colored code editing and a console window at the bottom lets you capture logs from your scripts.

- Mix/Cell/Generator Script Editor
  In this mode, the top panel allows you to edit Wotja Script Functions associated with your Mix or Cell or Generator..
- Script Sandbox
  In this mode, it is used to create and run scripts that aren't saved, and that don't access Mix properties. It is there for you to experiment with Wotja Script!

The top panel allows you to edit Wotja Script Functions associated with your object.

The panel can contain as many functions as you want, or as few as you want. Leave it blank if you don't want to use any scripting.

Tip for iOS Users: To get a "standard iOS" text editor with copy/paste etc, just select Action menu > Edit... You can then use normal iOS operations to copy/paste etc.

Here is an example with just one trigger script function:

```
// Just one function!

function onImeBar(bar) {
  wjsLog ("onImeBar: Bar number=" + bar)
}
```

Here is an example with three trigger script functions:

```
// Three functions!

function onImeStart() {
  wjsLog ("onImeStart!")
}

function onImeBar(bar) {
  wjsLog ("onImeBar: Bar number=" + bar)
}

function onImeStop() {
  wjsLog ("onImeStop!")
}
```

Test & Apply (only for Generator Script or Cell Script or Mix Script)

Pressing this button will compile any Wotja Script that you type in the large text area at the top of the window, and if the script looks valid, it will send the script to be stored within the Mix. It will take effect immediately (if your Mix is running).

Run Script (only for Script Sandbox)

Pressing this button will compile any Wotja Script that you type in the large text area at the top of the window, and display any results in the bottom panel. Use *Run Script* to quickly check your script for obvious syntax errors.

Note that any changes you make here aren't saved within the mix.

Open

Pressing the *Open* button displays the Wotja Scripts Window, where you can select a Wotja Script file from either built-in examples, or from ones you've saved yourself (which have a .wotjascript extension). Your saved custom scripts are listed in the top section; you can save your own custom scripts in this section, by using the Save button.

Save

Pressing the *Save* button lets you save the contents of your Script editor window to a document with the .wotjascript extension. You can then easily find and re-open that Script using the Open button (your saved custom scripts are listed in the top section).

Help

Pressing the *Help* button displays help on Wotja Scripting system!

Back (<)

Pressing the *Back (<)* button will close the screen. If you're editing a Generator Script or Cell Script or Mix Script, you'll first be prompted to save or discard any changes you might have made.

Action

Pressing the *Action* button will show the following popup menu (a Pro Feature Set is required to use them):

- *Edit... (iOS)*: Allows you to edit in a "standard iOS" text editor with standard iOS copy/paste etc.
- *Clear Script*: Select this to clear the script.
- *Export to Clipboard (iOS)*: Exports script to clipboard.
- *Import from Clipboard (iOS)*: Imports script from clipboard.

Cancel

Pressing the *Cancel* button will discard your changes.

Test Results

Any test results are displayed in the *Test Results* area at the bottom.

Clear

Pressing the *Clear* button in the bottom area, quickly erases the text in the bottom panel.

See Also

- Mix... Scripts

- Mix - Scripts
- Cell - Scripts
- Generator - Scripts
- IWS 20 User Guide
- Event Handler Script Functions
- Script Function Reference

Event Handler Script Functions (*"Triggers"*)

Triggers can be assigned to most Objects, including the Cell and Generator Objects.

You can enter these scripts using the following parameter groups / views:

- Mix - Scripts
- Cell - Scripts
- Generator - Scripts

Logging output to the Script console with *wjsLog*

The *wjsLog* function may be used to write text to the Script Console; this is very useful for debugging your scripts!

Wotja Scripts Window

The Wotja Scripts window lets you select from a number of built-in scripts that are provided by Intermorphic.

There are simple implementations of each of the Event Triggers functions for Generator Scripts or Cell Scripts or Mix Scripts, and there are a large number of *Cookbook* scripts for both Generator and Mix. The *Cookbook* scripts which are often quite detailed, and which are designed to help you get started with writing your own scripts.

When in this Window, select the item you wish to use in the Script Editor.

The top section lists any Scripts that you might have saved yourself; you can press the "Trash Can" icon to delete your user script.

Generator - Scripts

The Generator Scripts parameter allows you to embed small Event Handler Script Functions, which are small bits of code in the widely used JavaScript language, that are triggered when various events happen when the Generator is playing. Specially named Event Handler Script Functions in these scripts are triggered when various events happen when a Generator is playing.

Start

The onImeStart Event Handler Script Function is called once at the start of the Generator's owning cell, when that cell starts playing.

The function has no parameters.

```
function onImeStart() {
  wjsLog ("Generator start!")
}
```

Bar

The onImeBar Event Handler Script Function is called at the start of every bar while the mix is playing in its cell, starting from 1.

The function has one parameter:

- *bar* the bar number, starting from 1.

```
function onImeBar(bar) {
  wjsLog ("Generator Bar number=" + bar)
}
```

Stop

The onImeStop Event Handler Script Function is called once at the end of the Generator's owning cell, just as that Cell stops playing.

The function has no parameters.

```
function onImeStop() {
  wjsLog ("Mix stop!")
}
```

Composed

The onImeGeneratorComposed Event Handler Script Function is called when the Generator composes a note. Use this to emit MIDI CC events and what have you.

The function has five parameters:

- *generatorIndex* the generator index, starting from 0.
- *noteon* a value indicating if this is a note on event - true means note on, false means note off.
- *channel* the MIDI Channel, from 0 to 15 (MIDI channel is actually displayed in the IME Network screen as 1 to 16)
- *pitch* the MIDI pitch of the composed note, from 0 to 127.
- *velocity* the MIDI velocity of the composed note, from 1 to 127.

```
function onImeGeneratorComposed(generatorIndex, noteon, channel, pitch, velocity) {
  wjsLog ("Generator Composed index=" + generatorIndex + ", noteon=" + noteon + ", cha
```

```
}
```

User Controller Changed

The onImeGeneratorUserController Event Handler Script Function is called when either of the Generator's User Controllers output value changes.

The function has three parameters:

- *generatorIndex* the generator index, starting from 0.
- *controllerIndex* the zero-based index of the controller; 0 means User Controller 1, 1 means User Controller 2.
- *value* the controller's new value, from 0 to 127.

```
function onImeGeneratorUserController(generatorIndex, controllerIndex, value) {
  wjsLog ("onImeGeneratorUserController, generatorIndex=" + generatorIndex + ", contro
}
```

MIDI In CC

The onMIDIInCC Event Handler Script Function is called whenever a MIDI CC event is received by the MIDI Input device.

The function has three parameters:

- *channel* the MIDI Channel, from 0 to 15 (MIDI channel is actually displayed in the IME Network screen as 1 to 16)
- *cc* the CC event identifier, from 0 to 127.
- *value* the value of the CC event, from 0 to 127.

Note that your script will *only* respond to MIDI In CC Events that target the MIDI Channel that your generator is attached to.

```
function onMIDIInCC(channel, cc, value) {
  wjsLog ("Generator onMIDIInCC channel=" + channel + ", " +  cc + ", " +  value)
}
```

MIDI In Note

The onMIDIInNote Event Handler Script Event Handler Script Function is called whenever a MIDI Note On or Off event is received by the MIDI Input device.

The function has four parameters:

- *channel* the MIDI Channel, from 0 to 15 (MIDI channel is actually displayed in the IME Network screen as 1 to 16)

- *noteon* a value indicating if this is a note on event - true means note on, false means note off.
- *pitch* the MIDI pitch of the note event, from 0 to 127.
- *velocity* the velocity of the note event, from 0 to 127.

Note that your script will *only* respond to MIDI In Note Events that target the MIDI Channel that your generator is attached to.

```
function onMIDIInNote(channel, noteon, pitch, velocity) {
  wjsLog ("Generator onMIDIInNote channel=" + channel + ", noteon=" + noteon + ", " +
}
```

See Also

- IWS 20 User Guide
- Event Handler Script Functions
- Script Function Reference
- Wotja as a Hyperinstrument

Cell - Scripts

The Cell Scripts parameters allow you to embed small Event Handler Script Functions, which are small bits of code in the widely used JavaScript language, that are triggered when various events happen when the Cell is playing. Using Event Handler Script Functions. These scripts allow you to tell the Cell to behave in very powerful ways while it is playing.

Start

The onImeStart Event Handler Script Function is called once at the start of the Mix's owning cell, when that cell starts playing.

The function has no parameters.

```
function onImeStart() {
  wjsLog ("Mix start!")
}
```

Bar

The onImeBar Event Handler Script Function is called at the start of every bar while the mix is playing, starting from 1.

The function has one parameter:

- *bar* the bar number, starting from 1.

```
function onImeBar(bar) {
  wjsLog ("Bar number=" + bar)
}
```

Stop

The onImeStop Event Handler Script Function is called once at the end of the Mix's owning cell, just as that Cell stops playing.

The function has no parameters.

```
function onImeStop() {
  wjsLog ("Mix stop!")
}
```

MIDI In CC

The onMIDIInCC Event Handler Script Function is called whenever a MIDI CC event is received by the MIDI Input device.

The function has three parameters:

- *channel* the MIDI Channel, from 0 to 15 (MIDI channel is actually displayed in the IME Network screen as 1 to 16)
- *cc* the CC event identifier, from 0 to 127.
- *value* the value of the CC event, from 0 to 127.

```
function onMIDIInCC(channel, cc, value) {
  wjsLog ("Mix onMIDIInCC channel" + channel + ", " +  cc + ", " +  value)
}
```

MIDI In Note

The onMIDIInNote Event Handler Script Event Handler Script Function is called whenever a MIDI Note On or Off event is received by the MIDI Input device.

The function has four parameters:

- *channel* the MIDI Channel, from 0 to 15 (MIDI channel is actually displayed in the IME Network screen as 1 to 16)
- *noteon* a value indicating if this is a note on event - true means note on, false means note off.
- *pitch* the MIDI pitch of the note event, from 0 to 127.
- *velocity* the velocity of the note event, from 0 to 127.

```
function onMIDIInNote(channel, noteon, pitch, velocity) {
  wjsLog ("Mix MIDI in note=" +  noteon + ", " +  channel + ", " +  pitch + ", " +  ve
}
```

See Also

- IWS 20 User Guide
- Event Handler Script Functions
- Script Function Reference
- Wotja as a Hyperinstrument

Mix - Scripts

The Mix Scripts parameters allow you to embed small Event Handler Script Functions, which are small bits of code in the widely used JavaScript language, that are triggered when various events happen when the mix is playing. Using Event Handler Script Functions allows you to tell the Mix to behave in very powerful ways while it is playing.

Start

The onImeStart Event Handler Script Function is called once at the start of the Mix playing.

The function has no parameters.

```
function onImeStart() {
  wjsLog ("Mix start!")
}
```

Bar

The onImeBar Event Handler Script Function is called at the start of every bar while the mix is playing, starting from 1.

The function has one parameter:

- *bar* the bar number, starting from 1.

```
function onImeBar(bar) {
  wjsLog ("Bar number=" + bar)
}
```

Stop

The onImeStop Event Handler Script Function is called once, just as the Mix stops playing.

The function has no parameters.

```
function onImeStop() {
  wjsLog ("Mix stop!")
}
```

MIDI In CC

The onMIDIInCC Event Handler Script Function is called whenever a MIDI CC event is received by the MIDI Input device.

The function has three parameters:

- *channel* the MIDI Channel, from 0 to 15 (MIDI channel is actually displayed in the IME Network screen as 1 to 16)
- *cc* the CC event identifier, from 0 to 127.
- *value* the value of the CC event, from 0 to 127.

```
function onMIDIInCC(channel, cc, value) {
  wjsLog ("Mix onMIDIInCC channel" + channel + ", " +  cc + ", " +  value)
}
```

MIDI In Note

The onMIDIInNote Event Handler Script Event Handler Script Function is called whenever a MIDI Note On or Off event is received by the MIDI Input device.

The function has four parameters:

- *channel* the MIDI Channel, from 0 to 15 (MIDI channel is actually displayed in the IME Network screen as 1 to 16)
- *noteon* a value indicating if this is a note on event - true means note on, false means note off.
- *pitch* the MIDI pitch of the note event, from 0 to 127.
- *velocity* the velocity of the note event, from 0 to 127.

```
function onMIDIInNote(channel, noteon, pitch, velocity) {
  wjsLog ("Mix MIDI in note=" +  noteon + ", " +  channel + ", " +  pitch + ", " +  ve
}
```

See Also

- IWS 20 User Guide
- Event Handler Script Functions
- Script Function Reference
- Wotja as a Hyperinstrument

Scripting Objects

Objects and Parameters

The following table lists the various Objects and their associated parameters together with how those objects/parameters are available to Wotja Scripts, showing: the object name; the parameter group / view (where shown to a user); the displayed name (where shown to a user); the parameter name (as supplied to functions and which is always unique for a given object); and the range of legal values that you may supply.

Object: *"Generator" (AKA Generator)*

Parameter Group / View	Displayed Name	Script Parameter Name	Notes
Generator - Basics	Name	"Generator"	*Generator name*
	Mute	"Mute"	Yes or No
	Patch	"Patch"	Value in range 0 to 127
	Use Patch?	"Use Patch?"	Yes or No
	MIDI Channel	"MIDI Channel"	Value in range 0, 16
	Generator Type	"Generator Type"	"Rhythmic", "Ambient", "Follower", "Repeater", "Patterns", "Listening"
	Pitch	"Pitch"	Value in range 0 to 127
	Pitch Range	"Pitch Range"	Value in range 11, 127
	Phrase Length	"Phrase Length"	Value in range 1, 256; also in Generator - Ambient
	Phrase Length Range	"Phrase Length Range"	Value in range 0, 256; also in Generator - Ambient
	Phrase Gaps	"Phrase Gaps"	Value in range 0, 256; also in Generator - Ambient
	Phrase Gaps Range	"Phrase Gaps Range"	Value in range 0, 256; also in Generator - Ambient
	Note Rest %	"Phrase Note Rest %"	Value in range 0, 100; also in Generator - Ambient

Generator - Ambient	Units	"Ambient Units"	"Seconds (thousandths of a)", "Beats (60ths of a)", "Full seconds"
	Duration	"Ambient Duration"	Value in range 0, 32000
	Duration Range	"Ambient Duration Range"	Value in range 0, 32000
	Gap Minimum	"Ambient Gap Min"	Value in range 0, 32000
	Gap Range	"Ambient Gap Range"	Value in range 0, 32000
Generator - Follower	Follow Generator	"Follow Named Generator"	*Generator name*
	Percent	"Follow Percent"	Value in range 0, 100
	Strategy	"Follow Strategy"	"Chordal Harmony", "Interval Within Scale Rule", "Semitone Shift"
	Units	"Follow Delay Unit"	"Seconds (thousandths of a)", "Beats (60ths of a)", "Full seconds"
	Delay	"Follow Delay"	Value in range 0, 32000
	Delay Range	"Follow Delay Range"	Value in range 0, 32000
	Shift/Interval	"Follow Shift/Interval"	Value in range -60, +60
	S/I Range	"Follow Shift/Interval Range"	Value in range -60, +60
Generator - Repeater	Generator	"Repeat Specific Generator"	*Generator name*
	Percent	"Repeat Bars Percent"	Value in range 0, 100
	Bars	"Repeat For Bars"	Value in range 1, 100
	History Range	"Repeat Bar History Range"	Value in range 0, 100
	History	"Repeat Bar History"	Value in range 1, 100
	Bars Range	"Repeat For Bars Range"	Value in range 0, 100
Generator - Patterns	Patterns	"Patterns"	*Pattern String*
	Use Percent	"Patterns Use Percent"	Value in range 0, 100
	Mutation Factor	"Mutation factor"	Value from 0 to 1000 (corresponding to 0 to 100 percent)
	Bars Between	"Mutate No. Bars"	Value in range 0, 100
	Bars Range	"Mutate No. Bars Range"	Value in range 0, 100
	Mutate Rhythm?	"Mutation of Rhythm"	Yes or No
	Meter	"Meter"	"1:4", "2:4", "3:4", "4:4", "5:4", "6:4", "7:4", "8:4", "1:8", "2:8", "3:8", "4:8", "5:8", "6:8", "7:8", "8:8", "9:8", "10:8", "11:8", "12:8"
Generator - Text to Music	Text Pattern Enabled	"Text Pattern Enabled"	Yes or No

	Text Pattern Text	"Text Pattern Text"	*TTM Text String*
	Text Pattern Phrase Length	"Text Pattern Phrase Length"	Value in range 1 to 16
	Text Pattern Phrase Range	"Text Pattern Phrase Range"	Value in range 0 to 16
	Text Pattern Gaps	"Text Pattern Gaps"	Value in range 0 to 16
	Text Pattern Gaps Range	"Text Pattern Gaps Range"	Value in range 0 to 16
	Text Pattern Interval	"Text Pattern Interval"	Value in range 0 to 36
	Text Pattern Interval Range	"Text Pattern Interval Range"	Value in range 0 to 36
	Text Pattern Improvise After Tune	"Text Pattern Improvise After Tune"	Yes or No
	Text Pattern Variation	"Text Pattern Variation"	Value in range 0 to 100
	Text Pattern Repeats	"Text Pattern Repeats"	Value in range 0 to 16
	Text Pattern Repeats Range	"Text Pattern Repeats Range"	Value in range 0 to 16
	Text Pattern Tune Start At Index	"Text Pattern Tune Start At Index"	Value in range 0 to 100 No
	Text Pattern Tune Length Override	"Text Pattern Tune Length Override"	Value in range 0 to 100 No
	Text Pattern Display	"Text Pattern Display"	Yes or No
Generator - Chords	Depth	"Chord Depth"	Value in range 1, 32
	Pitch Offset	"Chord Pitch Offset"	Value in range -60, +60
	Delay	"Chord Delay"	Value in range 0, 32000
	Depth %	"Chord Depth Percent"	Value in range 0, 100
	Delay Unit	"Chord Delay Unit"	"Seconds (thousandths of a)", "Beats (60ths of a)", "Quantized Beats (60ths of a)"
	Velocity Factor	"Chord Velocity Factor"	Value in range -100, +100
	Delay Range	"Chord Delay Range"	Value in range 0, 32000
	Depth Range	"Chord Depth Range"	Value in range 0, 32
	Strategy	"Chord Strategy"	"Chordal Harmony", "Interval Within Scale Rule", "Semitone Shift"
	Shift/Interval	"Chord Shift/Interval"	Value in range -60, +60
	S/I Range	"Chord Shift/Interval Range"	Value in range -60, +60

Generator - Rules	Harmony Rules	"Harmony Rules"	*Rule name*
	Next Note Rules	"Next Note Rules"	*Rule name*
	Rhythm Rules	"Rhythm Rules"	*Rule name*
	Scale Rules	"Scale Rules"	*Rule name*
	Harmonize?	"Harmonize?"	Yes or No
	Voice Root	"Voice Root"	"A", "A#", "Ab" etc.
Generator - Comments	Copyright	"Copyright"	*Copyright text*
	Notes	"Notes"	*Notes text*
Generator - Arcticulation	Minimum	"Articulation Minimum"	Value in range 0, 100
	Range	"Articulation Range"	Value in range 0, 100
	Change	"Articulation Change"	Value in range 0, 100
	Change Range	"Articulation Change Range"	Value in range 0, 100
Generator - Controllers	Damper/Hold (64)	"Damper/Hold (64)"	Value in range -1, 127
	Harmonic Content (71)	"Harmonic Content (71)"	Value in range -1, 127
	Reverb (91)	"Reverb (91)"	Value in range -1, 127
	Chorus (93)	"Chorus (93)"	Value in range -1, 127
	Damper Release	"Damper Release"	Yes or No
	Portamento (84)	"Portamento (65)"	Value in range -1, 127
	MIDI Channel Sharing	"MIDI Channel Reallocation"	Yes or No
Generator - User Controller 1	MIDI CC	"User Controller 1 Midi Command"	Value in range 0 to 127
	Mode	"User Controller 1 Mode"	"-1 - Off", "0 - Random Drift", "1 - LFO (Min-Max-Min)", "2 - LFO (Max-Min-Max)", "3 - Sawtooth (Min-Max)", "4 - Sawtooth (Max-Min)"
	Minimum	"User Controller 1 Minimum"	Value in range 0, 127
	Range	"User Controller 1 Range"	Value in range 0, 127
	Change	"User Controller 1 Change"	Value in range 0, 127
	Change Range	"User Controller 1 Change Range"	Value in range 0, 127
	Update	"User Controller 1 Update"	Value in range 0, 10000
	Update Range	"User Controller 1 Update Range"	Value in range 0, 10000

	Update Units	"User Controller 1 Update Unit"	"Seconds (thousandths of a)", "Full seconds"
	Beat Cycle Length	"User Controller 1 Beat Cycle Length"	Value in range 0, 32000
	Phase Shift%	"User Controller 1 Phase Shift"	Value in range 0, 100
Generator - User Controller 2	MIDI CC	"User Controller 2 Midi Command"	Value in range 0 to 127
	Mode	"User Controller 2 Mode"	"-1 - Off", "0 - Random Drift", "1 - LFO (Min-Max-Min)", "2 - LFO (Max-Min-Max)", "3 - Sawtooth (Min-Max)", "4 - Sawtooth (Max-Min)"
	Minimum	"User Controller 2 Minimum"	Value in range 0, 127
	Range	"User Controller 2 Range"	Value in range 0, 127
	Change	"User Controller 2 Change"	Value in range 0, 127
	Change Range	"User Controller 2 Change Range"	Value in range 0, 127
	Update	"User Controller 2 Update"	Value in range 0, 10000
	Update Range	"User Controller 2 Update Range"	Value in range 0, 10000
	Beat Cycle Length	"User Controller 2 Beat Cycle Length"	Value in range 0, 32000
	Update Units	"User Controller 2 Update Unit"	"Seconds (thousandths of a)", "Full seconds"
	Phase Shift%	"User Controller 2 Phase Shift"	Value in range 0, 100
Generator - Micro Note Delay	Delay Range	"Micro Note Delay Range"	Value in range 0, 1000
	Delay Change	"Micro Note Delay Change"	Value in range 0, 1000
	Delay Offset	"Micro Note Delay Offset"	Value in range -1000, +1000
Generator - Micro Pitch	Bend Sensitivity	"Pitch Bend Sensitivity"	Value in range 0, 24
	Pitch Bend Offset	"Pitch Bend Offset"	Value in range -8192, +8191
	Pitch Range	"Micro Pitch Range"	Value in range 0, 8191
	Pitch Change	"Micro Pitch Change"	Value in range 0, 1000
	Pitch Update	"Micro Pitch Update"	Value in range 0, 10000
	Update Range	"Micro Pitch Update Range"	Value in range 0, 10000
Generator - Micro Vol Env	Range	"Micro Volume Range"	Value in range 0, 127
	Change	"Micro Volume Change"	Value in range 0, 127
	Update	"Micro Volume Update"	Value in range 0, 1000
	Update Range	"Micro Volume Update Range"	Value in range 0, 10000

Generator - Note to MIDI CC Mapping	CC for Note On?	"MIDI CC instead of Note?"	Yes or No
	Note On CC	"MIDI CC Note On Value"	Value from 0 to 127
	CC for Velocity?	"MIDI CC for Note On Velocity?"	Yes or No
	Velocity CC	"MIDI CC Note On Velocity"	Value from 0 to 127
	CC for Off?	"MIDI CC for Note Off?"	Yes or No
	Note Off CC	"MIDI CC Note Off Value"	Value from 0 to 127
Generator - User Envelope 1 (Volume)	MIDI CC	"User Envelope 1 MIDI CC"	Value from 0 to 127
	Enabled?	"User Envelope 1 Enabled"	Yes or No
	Envelope	"Volume"	*Envelope*
Generator - User Envelope 2 (Pan)	MIDI CC	"User Envelope 2 MIDI CC"	Value from 0 to 127
	Enabled?	"User Envelope 2 Enabled"	Yes or No
	Envelope	"Pan (10)"	*Envelope*
Generator - Envelope - Velocity	Velocity	"Velocity"	*Envelope*
Generator - Envelope - Velocity Range	Velocity Range	"Velocity Range"	*Envelope*
Generator - Envelope - Velocity Change	Velocity Change	"Velocity Change"	*Envelope*
Generator - Envelope - Velocity Change Range	Velocity Change Range	"Velocity Change Range"	*Envelope*

Object: *"Scale Rule"*

Parameter Group / View	Displayed Name	Script Parameter Name	Notes
Scale Rule	Value	"Value"	

Object: *"Harmony Rule"*

Parameter Group / View	Displayed Name	Script Parameter Name	Notes
Harmony Rule	Value	"Value"	

Object: *"Rhythm Rule"*

Parameter Group / View	Displayed Name	Script Parameter Name	Notes

Parameter Group / View	Displayed Name	Script Parameter Name	Notes
Rhythm Rule	Value	"Value"	

Object: *"Next Note Rule"*

Parameter Group / View	Displayed Name	Script Parameter Name	Notes
Next Note Rule	Value	"Value"	

Object: *"Cell"*

Parameter Group / View	Displayed Name	Script Parameter Name	Notes
Generative Cell Rules	Scale Rules	"Scale Rules"	
	Harmony Rules	"Harmony Rules"	
	Next Note Rules	"Next Note Rules"	
Cell Meter	Meter	"Meter"	"1:4", "2:4", "3:4", "4:4", "5:4", "6:4", "7:4", "8:4", "1:8", "2:8", "3:8", "4:8", "5:8", "6:8", "7:8", "8:8", "9:8", "10:8", "11:8", "12:8"

Object: *"Mix"*

Parameter Group / View	Displayed Name	Script Parameter Name	Notes
Mix Rules	Scale Rules	"Scale Rules"	
	Harmony Rules	"Harmony Rules"	
	Next Note Rules	"Next Note Rules"	
Mix	Mix Root	"Mix Root"	"A", "A#", "Ab" etc.
	Tempo	"Tempo"	Value in range 1, 400

Synth Preset List

The following table all Synth Presets available to *iseFxNetworkPresetSet* in the current version of Wotja. The list was created using Wotja Script and the *iseFxNetworkAllPresetNamesGet* function.

```
sounds/Bass/Bass fat swept.tg
sounds/Bass/Bass filter sweep.tg
sounds/Bass/Bass pluck soft.tg
sounds/Bass/Bass plucked bright.tg
sounds/Bass/Bass pulser.tg
sounds/Beats/Beats ch10 eperc wt.tg
sounds/Beats/Beats ch10 imdrums wt.tg
sounds/Beats/Beats ch10 ogglegacy wt.tg
sounds/Bell/Bell bright.tg
sounds/Bell/Bell combo.tg
sounds/Bell/Bell glass felt.tg
sounds/Bell/Bell glass tri.tg
sounds/Bell/Bell synth bright.tg
```

```
sounds/Bell/Bell synth soft.tg
sounds/Drone/Drone aeolian spinner.tg
sounds/Drone/Drone airier.tg
sounds/Drone/Drone buzz chorus low.tg
sounds/Drone/Drone buzz chorus sweep.tg
sounds/Drone/Drone cosmic 1.tg
sounds/Drone/Drone cosmic 2.tg
sounds/Drone/Drone fizz 2.tg
sounds/Drone/Drone fizz.tg
sounds/Drone/Drone fm sweep variation.tg
sounds/Drone/Drone fuzz.tg
sounds/Drone/Drone gentle harmonic shifts.tg
sounds/Drone/Drone gentle shifts pan.tg
sounds/Drone/Drone index range.tg
sounds/Drone/Drone lpf ebb blurred.tg
sounds/Drone/Drone mild peril.tg
sounds/Drone/Drone motor.tg
sounds/Drone/Drone perpetual motion.tg
sounds/Drone/Drone piano orch wash.tg
sounds/Drone/Drone slow phase morph.tg
sounds/Drone/Drone slow phase pan.tg
sounds/Drone/Drone slow phase strings.tg
sounds/Drone/Drone soft chorus sweep.tg
sounds/Drone/Drone strings octave.tg
sounds/Drone/Drone strings.tg
sounds/Drone/Drone subtle offset pulse.tg
sounds/Drone/Drone subtle pulse.tg
sounds/Drone/Drone synth soft.tg
sounds/Drone/Drone voxaa.tg
sounds/Drone/Drone warm fizzer.tg
sounds/Drone/Drone wide chorus low.tg
sounds/Drone/Drone wide subtle additive.tg
sounds/Drone/Drone wind in wire.tg
sounds/E-piano/E-Piano 2.tg
sounds/E-piano/E-Piano blend 2.tg
sounds/E-piano/E-Piano blend.tg
sounds/E-piano/E-Piano bowed.tg
sounds/E-piano/E-Piano bright.tg
sounds/E-piano/E-Piano shimmer.tg
sounds/E-piano/E-Piano soft.tg
sounds/E-piano/E-Piano space.tg
sounds/E-piano/E-Piano.tg
sounds/Lead/Lead bob.tg
sounds/Lead/Lead dsynth soft warm.tg
sounds/MO1-Lead/Brassy.tg
sounds/MO1-Lead/Breathy chime.tg
sounds/MO1-Lead/Bright FM keys 1.tg
sounds/MO1-Lead/Chime bar 1.tg
sounds/MO1-Lead/Felt mallet chime.tg
sounds/MO1-Lead/Filter warble.tg
sounds/MO1-Lead/FM keys 1.tg
sounds/MO1-Lead/FM keys 2.tg
sounds/MO1-Lead/FM keys 3.tg
sounds/MO1-Lead/FM keys 4.tg

sounds/MO1-Lead/FM keys 5.tg
```

```
sounds/MO1-Lead/FM keys 6.tg
sounds/MO1-Lead/FM pluck.tg
sounds/MO1-Lead/Harmonic scan 1.tg
sounds/MO1-Lead/Inverse harmonic scan.tg
sounds/MO1-Lead/Mellow FM keys 1.tg
sounds/MO1-Lead/Perc hi-lo.tg
sounds/MO1-Lead/Pluck octaves.tg
sounds/MO1-Lead/Rattle plate.tg
sounds/MO1-Lead/Resonant filter sweep.tg
sounds/MO1-Lead/Ringing sine.tg
sounds/MO1-Lead/Slight vocal.tg
sounds/MO1-Lead/Soft bright lead.tg
sounds/MO1-Lead/Soft filter sweep.tg
sounds/MO1-Lead/Soft hi-lo.tg
sounds/MO1-Lead/Soft perc tine.tg
sounds/MO1-Lead/Soft pluck bright tail.tg
sounds/MO1-Lead/Staccato supersaws hard attack.tg
sounds/MO1-Lead/Staccato supersaws.tg
sounds/MO1-Lead/Supersaws hard attack.tg
sounds/MO1-Lead/Supersaws.tg
sounds/MO1-Lead/Tuned noise.tg
sounds/MO1-Lead/WTable scan 1.tg
sounds/MO1-Lead/WTable scan 2.tg
sounds/MO1-Lead/WTable scan 3.tg
sounds/MO1-Lead/WTable scan 4.tg
sounds/MO1-Lead/WTable scan 5.tg
sounds/MO1-Lead/WTable scan 6.tg
sounds/Multi/Multi 3 op fm template.tg
sounds/Multi/Multi piano strings.tg
sounds/Multi/Multi wavetable crossfade template.tg
sounds/Noise/Noise mains hum.tg
sounds/Noise/Noise on the beach.tg
sounds/Noise/Noise ring mod high bell.tg
sounds/Noise/Noise tuned breezes.tg
sounds/Pad/Pad bowed.tg
sounds/Pad/Pad brass metal shimmer.tg
sounds/Pad/Pad brass soft.tg
sounds/Pad/Pad brassy.tg
sounds/Pad/Pad coda mod.tg
sounds/Pad/Pad epiano.tg
sounds/Pad/Pad fizz grower.tg
sounds/Pad/Pad fizz.tg
sounds/Pad/Pad hummer.tg
sounds/Pad/Pad outer space.tg
sounds/Pad/Pad pipe shimmer.tg
sounds/Pad/Pad pitched metal.tg
sounds/Pad/Pad side trails vib.tg
sounds/Pad/Pad soft chev 2.tg
sounds/Pad/Pad soft easel.tg
sounds/Pad/Pad soft ep 1.tg
sounds/Pad/Pad soft ep 2.tg
sounds/Pad/Pad soft for chording.tg
sounds/Pad/Pad soft tremolo.tg
sounds/Pad/Pad space park 1.tg

sounds/Pad/Pad space park 2.tg
```

```
sounds/Pad/Pad space park 3.tg
sounds/Pad/Pad space scape.tg
sounds/Pad/Pad space shimmer.tg
sounds/Pad/Pad space wind.tg
sounds/Pad/Pad synth 1.tg
sounds/Pad/Pad synth 2.tg
sounds/Pad/Pad unresolved.tg
sounds/Pad/Pad warm chev 1.tg
sounds/Pad/Pad warm soft.tg
sounds/Perc/Perc beat sync noise.tg
sounds/Piano/Piano blend bright.tg
sounds/Piano/Piano blend upright.tg
sounds/Piano/Piano brass.tg
sounds/Piano/Piano damped bright.tg
sounds/Piano/Piano damped long 1.tg
sounds/Piano/Piano damped long 2.tg
sounds/Piano/Piano damped.tg
sounds/Piano/Piano upright long.tg
sounds/Piano/Piano upright wide.tg
sounds/Pluck/Pluck guitar bell bright.tg
sounds/Pluck/Pluck guitar blend edge.tg
sounds/Pluck/Pluck guitar blend ep.tg
sounds/Pluck/Pluck guitar blend osc.tg
sounds/Pluck/Pluck guitar blend soft.tg
sounds/Pluck/Pluck guitar blend steel.tg
sounds/Pluck/Pluck guitar bowed.tg
sounds/Strings/Strings build.tg
sounds/Strings/Strings distant.tg
sounds/Strings/Strings slow.tg
sounds/Strings/Strings tremolo.tg
sounds/Synth/Synth beat sync filter blip.tg
sounds/Synth/Synth dsynth warm 2.tg
sounds/Synth/Synth fizz pulse.tg
sounds/Synth/Synth fizz soft.tg
sounds/Synth/Synth organ electric.tg
sounds/Synth/Synth pwm.tg
sounds/Synth/Synth saw phase offset.tg
sounds/Synth/Synth slow vowel morph.tg
sounds/Synth/Synth soft lead.tg
sounds/Synth/Synth subtractive template 1.tg
sounds/Synth/Synth subtractive template.tg
sounds/Synth/Synth swept harmonics.tg
sounds/Synth/Synth wavetable simple vibrato.tg
sounds/Units/Unit dsynth.tg
sounds/Units/Unit osc.tg
sounds/Units/Unit particle.tg
sounds/Units/Unit wavetable.tg
```

Effect Preset List

The following table all Effect Presets available to *iseFxNetworkPresetSet* in the current version of Wotja. The list was created using Wotja Script and the *iseFxNetworkAllPresetNamesGet* function.

```
fx/AllInOne/Combi chorus delay reverb 2 fxm
```

```
fx/AllInOne/Combi chorus delay reverb 2.fxm
fx/AllInOne/Combi chorus delay reverb.fxm
fx/AllInOne/Combi chorus reverb.fxm
fx/AllInOne/Combi compressor chorus delay eq.fxm
fx/AllInOne/Combi compressor chorus delay reverb eq.fxm
fx/AllInOne/Combi delay chorus reverb.fxm
fx/AllInOne/Combi delay eq.fxm
fx/AllInOne/Combi delay reverb eq.fxm
fx/AllInOne/Combi eq chorus reverb.fxm
fx/AllInOne/Combi filter delay 2.fxm
fx/AllInOne/Combi filter delay reverb muted.fxm
fx/AllInOne/Combi filter delay shift.fxm
fx/AllInOne/Combi filter delay.fxm
fx/AllInOne/Combi filter eq reverb.fxm
fx/AllInOne/Combi filter shifter.fxm
fx/AllInOne/Combi lfo filter chorus reverb.fxm
fx/AllInOne/Combi lfo filter chorus.fxm
fx/AllInOne/Combi mix eq compressor reverb.fxm
fx/AllInOne/Combi reverb eq.fxm
fx/AllInOne/Combi reverb lfo filter.fxm
fx/Amp/- Amp default.fxm
fx/Amp/Amp quad.fxm
fx/Chorus/- Chorus default.fxm
fx/Chorus/Chorus deep.fxm
fx/Chorus/Chorus double.fxm
fx/Chorus/Chorus fast flange.fxm
fx/Chorus/Chorus feedback.fxm
fx/Chorus/Chorus light.fxm
fx/Chorus/Chorus medium.fxm
fx/Chorus/Chorus strong.fxm
fx/Compressor/- Compressor default.fxm
fx/Compressor/Compressor basic.fxm
fx/Compressor/Compressor low end boost.fxm
fx/Compressor/Compressor mild.fxm
fx/Compressor/Compressor smooth.fxm
fx/Compressor/Compressor tight.fxm
fx/Delay/- Delay default.fxm
fx/Delay/Delay bounce.fxm
fx/Delay/Delay double sync.fxm
fx/Delay/Delay doubling.fxm
fx/Delay/Delay echoer.fxm
fx/Delay/Delay filtered mono.fxm
fx/Delay/Delay filtered sync pan.fxm
fx/Delay/Delay long.fxm
fx/Delay/Delay resonator.fxm
fx/Delay/Delay simple.fxm
fx/Delay/Delay slapback.fxm
fx/Delay/Delay tight slapback.fxm
fx/Distortion/- Distortion default.fxm
fx/Distortion/Distortion bren filter.fxm
fx/Distortion/Distortion fuzz drive.fxm
fx/Distortion/Distortion fuzz.fxm
fx/Distortion/Distortion mild.fxm
fx/EQ/- Equaliser default.fxm
fx/EQ/Equaliser hi cut.fxm
fx/EQ/Equaliser lo cut.fxm
```

```
fx/EQ/Equaliser mid.fxm
fx/Filter/- Filter default.fxm
fx/Filter/Filter 2Hz autowah.fxm
fx/Filter/FIlter 4Hz auto wah.fxm
fx/Filter/Filter 4Hz autowah.fxm
fx/Filter/Filter dirty.fxm
fx/Filter/Filter hi cut.fxm
fx/Filter/Filter high fast sweep.fxm
fx/Filter/Filter parallel bandpass 2 bar sweep.fxm
fx/Filter/Filter slow auto wah.fxm
fx/Filter/Filter slow sweep hi q.fxm
fx/Filter/Filter subtle sweep.fxm
fx/Filter/Filter telephone.fxm
fx/Overdrive/- Overdrive default.fxm
fx/Overdrive/Overdrive extreme.fxm
fx/Overdrive/Overdrive harmonics.fxm
fx/Overdrive/Overdrive hot.fxm
fx/Overdrive/Overdrive serious.fxm
fx/Overdrive/Overdrive warm.fxm
fx/Pan/Pan auto.fxm
fx/Pan/Pan shimmer.fxm
fx/Reverb/- Reverb default.fxm
fx/Reverb/Reverb 5 second.fxm
fx/Reverb/Reverb 9 second.fxm
fx/Reverb/Reverb basement.fxm
fx/Reverb/Reverb bright hall.fxm
fx/Reverb/Reverb bright plate.fxm
fx/Reverb/Reverb bright small room.fxm
fx/Reverb/Reverb drum hallway.fxm
fx/Reverb/Reverb drum studio.fxm
fx/Reverb/Reverb light ambience.fxm
fx/Reverb/Reverb medium room.fxm
fx/Reverb/Reverb resonant metallic.fxm
fx/Reverb/Reverb ringing 6 second.fxm
fx/Reverb/Reverb slapback room.fxm
fx/Reverb/Reverb smooth 10 second.fxm
fx/Volume/Volume 10 second duck.fxm
fx/Volume/Volume 4hz gate pan.fxm
fx/Volume/Volume 4hz gate.fxm
fx/Volume/Volume swell.fxm
fx/Volume/Volume tremolo fast.fxm
fx/Volume/Volume tremolo mono 2xbpm sync.fxm
fx/Volume/Volume tremolo pan.fxm
fx/Volume/Volume tremolo slow.fxm
fx/Volume/Volume tremolo stereo 2xbpm sync.fxm
```

See Also

- IWS 20 User Guide
- Event Handler Script Functions
- Wotja as a Hyperinstrument

Wotja Script Utility functions

Wotja Script Utility functions all start with the *wjs* prefix

```
wjsLog ()
```

The function *wjsLog* may be used to display text in the Script Console; this is very useful for debugging your scripts!

Example:

```
function onImeStart() {
  wjsLog("Hello World!")
}
```

```
wjsGetRandom ()
```

This function returns a random integer.

The function has no parameters.

Example:

```
var value = wjsGetRandom ()
```

```
wjsRandomGetFromTo (minimum, maximum)
```

This function returns a random integer in the range of the two supplied values.

The function has two parameters:

- *minimum* the minimum value that can be returned
- *maximum* the maximum value that can be returned

Example:

```
function onImeStart() {
  // Get a random integer between 0 and 50 inclusive.
  var value = wjsRandomGetFromTo (0, 50)
  wjsLog("random value=", value)
}
```

Mix related functions

These functions are intended to be use from Mix Scripts. Mix related functions all start with the *mix* prefix.

```
mixCellObjectCountGet (columnIndex, trackIndex, objectType)
```

Returns the number of objects of the specified type, in the current Wotja Mix.

The function has the following parameters:

- *columnIndex* the column index of interest; from 0 to 3; 0 is the first cell on the left, 3 is the last cell on the right
- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix
- *objectType* the type of the object of interest, i.e. one of *"Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule"*.

Example:

```
function testFunction() {
  var columnIndex = 0
  var trackIndex = 0
  // Example usage:
  var scaleRuleCount = mixCellObjectCountGet(columnIndex, trackIndex, "Scale Rule")
  print("Mix cell(" + columnIndex + "," + trackIndex + ") : scale rule count = " + sca
}
```

```
mixCellObjectIndexGet (columnIndex, trackIndex, objectType, objectIndex)
```

Returns the index of the named object, which will be greater than or equal to 0 if the named object is found; the function will return -1 if not found.

The function has the following parameters:

- *columnIndex* the column index of interest; from 0 to 3; 0 is the first cell on the left, 3 is the last cell on the right
- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix
- *objectType* the type of the object of interest, i.e. one of "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule".
- *objectName* the name of the object of interest

Example:

```
function testFunction() {
  var columnIndex = 0
  var trackIndex = 0
  // Example usage:
  var scaleRuleCount = mixCellObjectCountGet(columnIndex, trackIndex, "Scale Rule")
  print("Mix cell(" + columnIndex + "," + trackIndex + ") : scale rule count = " + sca
  var scaleRuleName = mixCellObjectNameGet(columnIndex, trackIndex, "Scale Rule", 0)
  print("Mix cell(" + columnIndex + "," + trackIndex + ") : scale rule 0 has name = " +
  var indexOfScaleRuleName = mixCellObjectIndexGet(columnIndex, trackIndex, "Scale Rul
  print("Mix cell(" + columnIndex + "," + trackIndex + ") : scale rule with name=" + s

}
```

```
mixCellObjectNameGet (columnIndex, trackIndex, objectType, objectIndex)
```

Returns the name of the specified object.

The function has the following parameters:

- *columnIndex* the column index of interest; from 0 to 3; 0 is the first cell on the left, 3 is the last cell on the right
- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix
- *objectType* the type of the object of interest, i.e. one of "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule".

- *objectIndex* the index of the object of interest, starting at 0 for the first object.

Example:

```
function testFunction() {
  var columnIndex = 0
  var trackIndex = 0
  // Example usage:
  var scaleRuleCount = mixCellObjectCountGet(columnIndex, trackIndex, "Scale Rule")
  print("Mix cell(" + columnIndex + "," + trackIndex + ") : scale rule count = " + sca
  var scaleRuleName = mixCellObjectNameGet(columnIndex, trackIndex, "Scale Rule", 0)
  print("Mix cell(" + columnIndex + "," + trackIndex + ") : scale rule 0 has name = "
}
```

```
mixCellObjectParameterGet (columnIndex, trackIndex, objectType, objectIndex, parameterI
```

Returns the value of the parameter, for the specified object index of the specified object type, in the playing mix.

The function has the following parameters:

- *columnIndex* the column index of interest; from 0 to 3; 0 is the first cell on the left, 3 is the last cell on the right
- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix
- *objectType* the type of the object of interest, i.e. one of "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule".
- *objectIndex* the index of the object of interest, starting at 0 for the first object. If the objectType is "Cell", or "Mix", you do not provide this parameter.
- *parameterName* the parameter of interest.

Example:

```
function testFunction() {
  // Example usage:
  var columnIndex = 0
  var trackIndex = 0
  var generatorIndex = 0
  var generatorValue = mixCellObjectParameterGet(columnIndex, trackIndex, "Generator",
  print("generatorValue = " + generatorValue)
  mixCellObjectParameterSet(columnIndex, trackIndex, "Generator", generatorIndex, "Sca
```

```
  generatorValue = mixCellObjectParameterGet(columnIndex, trackIndex, "Generator", gen
  print("generatorValue = " + generatorValue)
  var cellValue = mixCellObjectParameterGet(columnIndex, trackIndex, "Cell", "Scale Ru
  print("cellValue = " + cellValue)
}
```

```
mixCellObjectParameterSet (columnIndex, trackIndex, objectType, objectIndex, parameterI
```

Sets the parameter to the the supplied value, for the specified object index of the specified object type, in the playing mix.

The function has the following parameters:

- *columnIndex* the column index of interest; from 0 to 3; 0 is the first cell on the left, 3 is the last cell on the right
- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix
- *objectType* the type of the object of interest, i.e. one of *"Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule"*.
- *objectIndex* the index of the object of interest, starting at 0 for the first object. If the objectType is "Cell", or "Mix", you do not provide this parameter.
- *parameterName* the parameter of interest.
- *newValue* the new parameter value to use.
  Special behaviour for Rules: if you set *newValue* to be the name a built-in rule (e.g. Scale Rule "All Scale Major") for the parameter value, the parameter value will be set magically to that of the corresponding rule type. Otherwise, you'd need to set the rule value in the normal way, with a string looking something like *"1 0 1 1 0 1 0 1 0 1"*

Example:

```
function testFunction() {
  // Example usage:
  var columnIndex = 0
  var trackIndex = 0
  var generatorIndex = 0
  var generatorValue = mixCellObjectParameterGet(columnIndex, trackIndex, "Generator",
  print("generatorValue = " + generatorValue)
  mixCellObjectParameterSet(columnIndex, trackIndex, "Generator", generatorIndex, "Sca
  generatorValue = mixCellObjectParameterGet(columnIndex, trackIndex, "Generator", gen
  print("generatorValue = " + generatorValue)
  var cellValue = mixCellObjectParameterGet(columnIndex, trackIndex, "Cell", "Scale Ru
```

```
  print("cellValue = " + cellValue)

  mixCellObjectParameterSet(columnIndex, trackIndex, "Cell", "Scale Rules", "Mix\nDefa
  cellValue = mixCellObjectParameterGet(columnIndex, trackIndex, "Cell", "Scale Rules"
  print("cellValue = " + cellValue)
}
```

```
mixColumnLockGet ()
```

Returns the currently locked column index: if no column is currently locked, this returns -1 otherwise, returns a value from 0 to 3

The function has no parameters

Example:

```
function testFunction() {
  var columnLockedIndex = mixColumnLockGet()
  print ("columnLockedIndex = " + columnLockedIndex)
}
```

```
mixColumnLockSet (columnIndex)
```

Sets column lock (or not); if columnIndex is 0 to 3, this sets the column lock on; if columnIndex is -1, this turns the column lock off

The function has the following parameters:

- *columnIndex* The columnIndex index of the cell (from 0 to 3).

Example:

```
function testFunction() {
  // Example usages:
  var columnLockedIndex = mixColumnLockGet()
  print ("columnLockedIndex = " + columnLockedIndex)

  mixColumnLockSet(0)
```

```
  columnLockedIndex = mixColumnLockGet()

  print ("columnLockedIndex = " + columnLockedIndex)

  mixColumnLockSet(1)
  columnLockedIndex = mixColumnLockGet()
  print ("columnLockedIndex = " + columnLockedIndex)

  mixColumnLockSet(2)
  columnLockedIndex = mixColumnLockGet()
  print ("columnLockedIndex = " + columnLockedIndex)

  mixColumnLockSet(3)
  columnLockedIndex = mixColumnLockGet()
  print ("columnLockedIndex = " + columnLockedIndex)

  mixColumnLockSet(-1)
  columnLockedIndex = mixColumnLockGet()
  print ("columnLockedIndex = " + columnLockedIndex)
}
```

```
mixObjectCountGet (objectType)
```

Returns the number of objects of the specified type, in the current Wotja Mix.

The function has the following parameters:

- *objectType* the type of the object of interest, i.e. one of "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule".

Example:

```
function testFunction() {
  // Example usage:
  var scaleRuleCount = mixObjectCountGet("Scale Rule")
  print("Mix scale rule count = " + scaleRuleCount)
}
```

```
mixObjectIndexGet (objectType, objectIndex)
```

Returns the index of the named object, which will be greater than or equal to 0 if the named object is found; the function will return -1 if not found.

The function has the following parameters:

- *objectType* the type of the object of interest, i.e. one of "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule".
- *objectName* the name of the object of interest

Example:

```
function testFunction() {
  // Example usage:
  var scaleRuleName = mixObjectNameGet("Scale Rule", 0)
  print("Mix scale rule 0 has name = " + scaleRuleName)
  var indexOfScaleRuleName = mixObjectIndexGet("Scale Rule", scaleRuleName)
  print("Mix scale rule with name=" + scaleRuleName + " is at index=" + indexOfScaleRu
}
```

```
mixObjectNameGet (objectType, objectIndex)
```

Returns the name of the specified object.

The function has the following parameters:

- *objectType* the type of the object of interest, i.e. one of "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule".
- *objectIndex* the index of the object of interest, starting at 0 for the first object.

Example:

```
function testFunction() {
  // Example usage:
  var scaleRuleCount = mixObjectCountGet("Scale Rule")
  print("Mix scale rule count = " + scaleRuleCount)
  var scaleRuleName = mixObjectNameGet("Scale Rule", 0)
  print("Mix scale rule 0 has name = " + scaleRuleName)
  var indexOfScaleRuleName = mixObjectIndexGet("Scale Rule", scaleRuleName)
  print("Mix scale rule with name=" + scaleRuleName + " is at index=" + indexOfScaleRu
}
```

```
mixObjectParameterGet (objectType, objectIndex, parameterName)
```

Returns the value of the parameter, for the specified object index of the specified object type, in the playing mix.

The function has the following parameters:

- *objectType* the type of the object of interest, i.e. one of "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule".
- *objectIndex* the index of the object of interest, starting at 0 for the first object. If the objectType is "Cell", or "Mix", you do not provide this parameter.
- *parameterName* the parameter of interest.

Example:

```
function testFunction() {
  // Example usage:
  var scaleRuleValue = mixObjectParameterGet("Scale Rule", 0, "Value")
  print("Mix scale rule index 0, value was = " + scaleRuleValue)
  scaleRuleValue = mixObjectParameterSet("Scale Rule", 0, "Value", "1.0000 0.4961 0.00(
  scaleRuleValue = mixObjectParameterGet("Scale Rule", 0, "Value")
  print("Mix scale rule index 0, value now = " + scaleRuleValue)
}
```

```
mixObjectParameterSet (objectType, objectIndex, parameterName, newValue)
```

Sets the parameter to the the supplied value, for the specified object index of the specified object type, in the playing mix.

The function has the following parameters:

- *objectType* the type of the object of interest, i.e. one of "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule".
- *objectIndex* the index of the object of interest, starting at 0 for the first object. If the objectType is "Cell", or "Mix", you do not provide this parameter.
- *parameterName* the parameter of interest.
- *newValue* the new parameter value to use.

Special behaviour for Rules: if you set *newValue* to be the name a built-in rule (e.g. "Major") for the parameter value, the parameter value will be set magically to that of the corresponding rule type. Otherwise, you'd need to set the rule value in the normal way, with a string looking something like *"1 0 0.5 0 1 0.5 0 1 0 0.5 0 0.5"*

Example:

```
function testFunction() {
  // Example usage:
  var scaleRuleValue = mixObjectParameterGet("Scale Rule", 0, "Value")
  print("Mix scale rule index 0, value was = " + scaleRuleValue)
  scaleRuleValue = mixObjectParameterSet("Scale Rule", 0, "Value", "1.0000 0.4961 0.00(
  scaleRuleValue = mixObjectParameterGet("Scale Rule", 0, "Value")
  print("Mix scale rule index 0, value now = " + scaleRuleValue)
}
```

```
mixParameterSet (parameterName, newValue)
```

This function is used to set the mix's parameter to the supplied value, for the object within whose trigger script you place this call.

The function has two parameters:

- *parameterName* the name of the parameter for which you want to set the value.
- *newValue* the new parameter value to use.

  Special behaviour for Rules: if you set *newValue* to be the name a built-in rule for the parameter value (e.g. Scale Rule "All Scale Major"), the parameter value will be set magically to that of the corresponding rule type. Otherwise, you'd need to set the rule value in the normal way, with a string looking something like *"1 0 1 1 0 1 0 1 0 1"*

Example:

```
function testFunction() {
  // Example usage:
  var mixValue = mixParameterGet("Scale Rules")
  print("Mix: Scale Rules = " + mixValue)
  mixParameterSet("Scale Rules", "Default")
  mixValue = mixParameterGet("Scale Rules")
  print("Mix: Scale Rules = " + mixValue)
```

```
  mixValue = mixParameterGet("Mix Root")

  print("Mix: Mix Root = " + mixValue)
  mixParameterSet("Mix Root", "C")
  mixValue = mixParameterGet("Mix Root")
  print("Mix: Mix Root = " + mixValue)

  mixValue = mixParameterGet("Tempo")
  print("Mix: Tempo = " + mixValue)
  mixParameterSet("Tempo", "100")
  mixValue = mixParameterGet("Tempo")
  print("Mix: Tempo = " + mixValue)
}
```

```
mixParameterGet (parameterName)
```

This function returns the value of the named parameter, for the object within whose trigger script you place this call. Note that the value returned is always a *string*, which you can convert to a number using *tonumber()*.

The function has one parameter:

- *parameterName* the name of the parameter for which you want to determine the current value.

Example:

```
function testFunction() {
  // Example usage:
  var mixValue = mixParameterGet("Scale Rules")
  print("Mix: Scale Rules = " + mixValue)

  mixValue = mixParameterGet("Mix Root")
  print("Mix: Mix Root = " + mixValue)

  mixValue = mixParameterGet("Tempo")
  print("Mix: Tempo = " + mixValue)
}
```

```
mixTrackCellLoopGet (trackIndex)
```

Returns a value which is the column currently locked for that track; from 0 to 3; or -1 if no cell is looping

The function has the following parameters:

- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix

Example:

```
function testFunction() {
  var trackIndex = 0
  var loopingCellColumnIndex = mixTrackCellLoopGet(trackIndex)
  print("loopingCellColumnIndex = " + loopingCellColumnIndex)
}
```

```
mixTrackCellLoopSet (trackIndex, columnIndex)
```

sets cell to loop (or not); if columnIndex is 0 to 3, this starts the track looping at that column index; if columnIndex is -1, this stops the track looping; what happens next, depends on the track rule.

The function has the following parameters:

- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix
- *columnIndex* The columnIndex index of the cell (from 0 to 3).

Example:

```
function testFunction() {
  var trackIndex = 0
  var loopingCellColumnIndex = mixTrackCellLoopGet(trackIndex)
  print("loopingCellColumnIndex = " + loopingCellColumnIndex)

  mixTrackCellLoopSet(trackIndex, -1)
  loopingCellColumnIndex = mixTrackCellLoopGet(trackIndex)
  print("loopingCellColumnIndex = " + loopingCellColumnIndex)
}
```

```
mixTrackPanGet (trackIndex)
```

Returns pan value from 0 to 127; 0 is far left, 64 is middle, 127 is far right

The function has the following parameters:

- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix

Example:

```
function testFunction() {
  // Example usage:
  var trackIndex = 0
  var value = mixTrackPanGet(trackIndex) // Value from 0 to 127, 0 is far left, 64 is r
  print("Track pan = " + value)
}
```

```
mixTrackPanSet (trackIndex, panValue)
```

Sets the pan setting for the specified track.

The function has the following parameters:

- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix
- *panValue* The pan value to use, from 0 to 127; 0 is far left, 64 is middle, 127 is far right

Example:

```
function testFunction() {
  // Example usage:
  var trackIndex = 0
  var value = mixTrackPanGet(trackIndex) // Value from 0 to 127, 0 is far left, 64 is r
  print("Track pan = " + value)
  mixTrackPanSet(trackIndex, 64) // Set to middle!
  value = mixTrackPanGet(trackIndex)
  print("Track pan = " + value)
}
```

```
mixTrackRuleGet (trackIndex)
```

Returns a string value which is current track rule setting, one of: "Sequence", "Loop" or "One Shot"

The function has the following parameters:

- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix

Example:

```
function testFunction() {

  var trackIndex = 0

  var trackRule = mixTrackRuleGet(trackIndex)
  print("trackRule = " + trackRule)
}
```

```
mixTrackRuleSet (trackIndex, ruleValue)
```

Sets the pan setting for the specified track.

The function has the following parameters:

- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix
- *ruleValue* one of: "Sequence", "Loop" or "One Shot"

Example:

```
function testFunction() {

  var trackIndex = 0

  var trackRule = mixTrackRuleGet(trackIndex)
  print("trackRule = " + trackRule)

  mixTrackRuleSet(trackIndex, "One Shot")
  trackRule = mixTrackRuleGet(trackIndex)
  print("trackRule = " + trackRule)

  mixTrackRuleSet(trackIndex, "Loop")
```

```
  mixTrackRuleSet(trackIndex, "Loop")

  trackRule = mixTrackRuleGet(trackIndex)
  print("trackRule = " + trackRule)

  mixTrackRuleSet(trackIndex, "Sequence")
  trackRule = mixTrackRuleGet(trackIndex)
  print("trackRule = " + trackRule))
}
```

```
mixTrackVolumeGet (trackIndex)
```

Returns volume value from 0 to 127

The function has the following parameters:

- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix

Example:

```
function testFunction() {
  // Example usage:
  var trackIndex = 0
  var value = mixTrackVolumeGet(trackIndex)
  print("Track volume = " + value)
}
```

```
mixTrackVolumSet (trackIndex, volumeValue)
```

Sets the volume setting for the specified track.

The function has the following parameters:

- *trackIndex* the track index of interest; from 0 to 11; 0 is the first track at the top of the mix, 11 is the last track in the mix
- *volumeValue* The volume value to use, from 0 to 127

Example:

```
function testFunction() {
  // Example usage:
  var trackIndex = 0
  var value = mixTrackVolumeGet(trackIndex)
  print("Track volume = " + value)
  mixTrackVolumeSet(trackIndex, 127) // Set to max!
  value = mixTrackVolumeGet(trackIndex)
  print("Track volume = " + value)
}
```

```
mixVolumeGet (trackIndex)
```

Returns the mix master volume value from 0 to 127

The function has no parameters

Example:

```
function testFunction() {
  // Example usage:
  var trackIndex = 0
  var value = mixVolumeGet(trackIndex)
  print("mix volume = " + value)
}
```

```
mixVolumSet (trackIndex, volumeValue)
```

Sets the the mix master volume setting

The function has the following parameters:

- *volumeValue* The volume value to use, from 0 to 127

Example:

```
function testFunction() {
  // Example usage:
  var trackIndex = 0
```

```
  var trackIndex = 0

  var value = mixVolumeGet(trackIndex)
  print(" volume = " + value)
  mixVolumeSet(trackIndex, 127) // Set to max!
  value = mixVolumeGet(trackIndex)
  print("mix volume = " + value)
}
```

IME related functions

The IME contains many powerful scripting features. The IME related functions all start with the *ime* prefix.

```
imeParameterSet (parameterName, newValue)
```

This function is used to set the object's parameter to the supplied value, for the object within whose trigger script you place this call.

The function has two parameters:

- *parameterName* the name of the parameter for which you want to set the value.
- *newValue* the new parameter value to use.
  Special behaviour for Rules: if you set *newValue* to be the name a built-in rule (e.g. "Major") for the parameter value, the parameter value will be set magically to that of the corresponding rule type. Otherwise, you'd need to set the rule value in the normal way, with a string looking something like *"1 0 0.5 0 1 0.5 0 1 0 0.5 0 0.5"*

Example:

```
function onImeStart() {
  // Set the Chord Depth to 4 (assuming we make this call from within a
  // Generator's trigger script!)
  var chordDepth = 4
  imeParameterSet("Chord Depth", chordDepth)
}
```

```
imeParameterGet (parameterName)
```

This function returns the value of the named parameter, for the object within whose trigger script you place this call. Note that the value returned is always a *string*, which you can convert to a number using *tonumber()*.

The function has one parameter:

- *parameterName* the name of the parameter for which you want to determine the current value.

Example:

```
function onImeStart() {
  // Get the current Chord Depth (assuming we make this call from within a
  // Generator's trigger script!)
  var chordDepth = imeParameterGet("Chord Depth")
  wjsLog ("Chord Depth=" + chordDepth)
}
```

```
imeGeneratorMIDIChannelGet()
```

This function returns the MIDI channel (assuming the script is called from within a generator script!). The value returned is a value from 0 to 15 (MIDI channel is actually displayed in the IME Network screen as 1 to 16).

The function has no parameters.

Example:

```
function onImeStart() {
  var channel = imeGeneratorMIDIChannelGet ()
  wjsLog ("Channel=" + channel)
}
```

```
imeRuleElementGet (ruleObjectType, ruleObjectIndex, ruleElementIndex)
```

This function returns the rule element value of the specified index, for the indexed rule object. The value

returned is a value between 0 and 127.

The function has the following parameters:

- *ruleObjectType* the object type i.e. one of *"Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule"*.
- *ruleObjectIndex* the index of the rule object of interest, starting at 0 for the first object.
- *ruleElementIndex* the index of the element; starting at 0 for the initial element.

Example 1:

```
function onImeStart() {
  // Dump out the scale rule element values, for the first scale.
  var scaleRuleIndex = 0
  var itemIndex = 0
  while (itemIndex < 12)
  {
    var value = imeRuleElementGet ("Scale Rule", scaleRuleIndex, itemIndex)
    wjsLog ("value=" + value)
    index = itemIndex + 1
  }
}
```

Example 2:

```
function onImeStart() {
  var scaleRuleCount = imeObjectCountGet("Scale Rule")
  wjsLog ("Scale Rule objects=" + scaleRuleCount)

  for (var scaleRuleIndex = 0; scaleRuleIndex < scaleRuleCount; scaleRuleIndex++) {
    // Dump out the scale rule element values, for the scale!
    var ruleName = imeObjectNameGet("Scale Rule", scaleRuleIndex)
    wjsLog ("Scale=" + ruleName)

    var itemIndex = 0
    while (itemIndex < 12)
    {
      var value = imeRuleElementGet ("Scale Rule", scaleRuleIndex, itemIndex)
      wjsLog ("itemIndex" + ", " +  itemIndex + ", " +  "value" + ", " +  value)
      itemIndex = itemIndex + 1
    }
  }
}
```

```
imeMIDISendCC (channel, cc, value [, delay])
```

This function emits the specified MIDI CC event.

The function has the following parameters:

- *channel* the MIDI channel to use; from 1 to 16.
- *cc* the MIDI CC to use; from 0 to 127.
- *value* the controller value to use, from 0 to 127.
- *delay* an optional delay to use, from 0 up, defaulting to 0; in IME pattern time units (where 60 represents one crotchet or quarter note). The delay is relative to the current IME timebase relevant to the trigger script in question.

Example:

```
function onImeBar(bar) {
  // Generator trigger...
  // Apply a pan sweep through the bar from left to right,
  // to show-off the use of imeMIDISendCC.

  var duration = imeBarDurationGet()

  var midiChannel = imeGeneratorMIDIChannelGet()
  var delay = 0
  var ccController = 10 // Pan controller!
  var value = 0
  while delay <= duration {
    var value = (127 * delay) / duration
    // Note that the "delay" is optional; we use this in this specific
    // demo, to get a sweep effect from start to end of the bar.
    imeMIDISendCC(midiChannel, ccController, value, delay)
    delay = delay + 20
 }
}
```

```
imeBarDurationGet ()
```

This function returns the duration of the mix bar, in IME pattern time units; where 240 represents four crotchets or one whole note or a bar of 4:4 time.

The function has no parameters.

Example:

```
function onImeStart() {
  var duration = imeBarDurationGet()
  wjsLog ("duration=" + duration)
}
```

```
imeObjectCountGet (objectType)
```

Returns the number of objects of the specified type, in the current Wotja Mix.

The function has the following parameters:

- *objectType* the type of the object of interest, i.e. one of *"Generator", "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule"*

Example:

```
function onImeStart() {
  var generatorCount = imeObjectCountGet("Generator")
  wjsLog ("Generator objects=" + generatorCount)
}
```

```
imeObjectIndexGet (objectType, objectIndex)
```

Returns the index of the named object, which will be greater than or equal to 0 if the named object is found; the function will return -1 if not found.

The function has the following parameters:

- *objectType* the type of the object of interest, i.e. one of *"Generator", "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule"*.
- *objectName* the name of the object of interest

Example:

```
function onImeStart() {
  var majorScaleRuleIndex = imeObjectIndexGet("Scale Rule", "Major")
```

```
  wjsLog ("majorScaleRuleIndex=" + majorScaleRuleIndex)

}
```

```
imeObjectNameGet (objectType, objectIndex)
```

Returns the name of the specified object.

The function has the following parameters:

- *objectType* the type of the object of interest, i.e. one of *"Generator", "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule"*.
- *objectIndex* the index of the object of interest, starting at 0 for the first object.

Example:

```
function onImeStart() {
  var generatorCount = imeObjectCountGet("Generator")
  wjsLog ("Generator objects=" + generatorCount)

  for (var index = 0; index < generatorCount; index++) {
    wjsLog ("Generator name=" + imeObjectNameGet("Generator", index))
  }
}
```

```
imeObjectParameterGet (objectType, objectIndex, parameterName)
```

Returns the value of the parameter, for the specified object index of the specified object type, in the playing mix.

The function has the following parameters:

- *objectType* the type object of interest, i.e. one of *"Mix", "Cell", "Generator", "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule"*.
- *objectIndex* the index of the object of interest, starting at 0 for the first object. If the objectType is "Cell", or "Mix", you do not provide this parameter.
- *parameterName* the parameter of interest.

Example:

```
function onImeStart() {

  var valueWas =  imeObjectParameterGet("Generator", 0, "Chord Depth")

  imeObjectParameterSet("Generator", 0, "Chord Depth", 4)

  var valueNow =  imeObjectParameterGet("Generator", 0, "Chord Depth")

  wjsLog("Chord Depth valueWas=" + valueWas + ", valueNow = " + valueNow)
}
```

```
imeObjectParameterSet (objectType, objectIndex, parameterName, newValue)
```

Sets the parameter to the the supplied value, for the specified object index of the specified object type, in the playing mix.

The function has the following parameters:

- *objectType* the type object of interest, i.e. one of *"Mix", "Cell", "Generator", "Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule"*
- *objectIndex* the index of the object of interest, starting at 0 for the first object. If the objectType is "Cell", or "Mix", you do not provide this parameter.
- *parameterName* the parameter of interest.
- *newValue* the new parameter value to use.
  Special behaviour for Rules: if you set *newValue* to be the name a built-in rule (e.g. "Major") for the parameter value, the parameter value will be set magically to that of the corresponding rule type. Otherwise, you'd need to set the rule value in the normal way, with a string looking something like *"1 0 0.5 0 1 0.5 0 1 0 0.5 0 0.5"*

Example:

```
function onImeStart() {

  var valueWas =  imeObjectParameterGet("Generator", 0, "Chord Depth")

  imeObjectParameterSet("Generator", 0, "Chord Depth", 4)

  var valueNow =  imeObjectParameterGet("Generator", 0, "Chord Depth")

  wjsLog("Chord Depth valueWas=" + valueWas + ", valueNow = " + valueNow)
}
```

```
imeCellElapsedPercentGet ()
```

Returns the elapsed percentage of the playing mix, within the context of it's container cell. A cell with duration set to "Infinite" is actually around 8 hours before it restarts; so will tend to report 0 percent. A cell with duration of just a few bars, will progress quickly from 0 through to 100.

The function has no parameters.

Example:

```
function onImeBar() {
  var value = imeCellElapsedPercentGet()
  wjsLog ("imeCellElapsedPercentGet=" + value)
}
```

```
imeRuleElementSet (ruleObjectType, ruleObjectIndex, ruleElementIndex, newValue)
```

This function set the rule element value of the specified index, for the indexed rule object, to the specified value. The value must be between 0 and 127.

The function has the following parameters:

- *ruleObjectType* the object type i.e. one of *"Scale Rule", "Harmony Rule", "Rhythm Rule", "Next Note Rule"*.
- *ruleObjectIndex* the index of the rule object of interest, starting at 0 for the first object.
- *ruleElementIndex* the index of the element; starting at zero for the initial element.
- *newValue* the new element value to use, in a range from 0 to 127.

Example 1:

```
function onImeStart() {
  // Set the scale rule element values, to odd settings (!), for the first scale.
  var scaleRuleIndex = 0
  var itemIndex = 0
  while (itemIndex < 12) {
    imeRuleElementSet ("Scale Rule", scaleRuleIndex, itemIndex, (itemIndex * 127) / 12
    itemIndex = itemIndex + 1
  }
```

```
}
```

Example 2:

```
function onImeStart() {
  var ruleCount = imeObjectCountGet("Scale Rule")
  wjsLog ("Scale Rule objects=" + ruleCount)

  for (var scaleRuleIndex = 0; scaleRuleIndex < ruleCount; scaleRuleIndex++) {
    // Set the scale rule element values, for the scale, to stupid values!
    var ruleName = imeObjectNameGet("Scale Rule", scaleRuleIndex)
    wjsLog ("Scale=" + ruleName)

    var itemIndex = 0
    while (itemIndex < 12) {
      var value = (itemIndex * 127) / 12
      imeRuleElementSet ("Scale Rule", scaleRuleIndex, itemIndex, value)
      itemIndex = itemIndex + 1
    }
  }
}
```

```
imeGeneratorEnvelopePercentGet (generatorObjectIndex, parameterName, percent)
```

This function returns the percent value for the specified envelope at the given percent position. The value returned is a value between 0 and 127.

The function has the following parameters:

- *generatorObjectIndex* the index of the generator object of interest.
- *parameterName* the name of the envelope parameter (e.g. *Volume*).
- *percent* the percent value; from 0 to 100.

Example:

```
function onImeStart() {
  var generatorCount = imeObjectCountGet("Generator")
  wjsLog ("Generator objects=" + generatorCount)

  for (var generatorIndex = 0; generatorIndex < generatorCount; generatorIndex++) {
    // Ramp-up every generator volume envelope!
```

```
    var generatorName = imeObjectNameGet("Generator", generatorIndex)
    wjsLog ("Generator=" + generatorName)


    var percent = 0
    while (percent <= 100)
    {
      var valueWas = imeGeneratorEnvelopePercentGet (generatorIndex, "Volume", percent
      wjsLog ("Volume at " + percent + ", " +  "was=" + valueWas)

      var setToValue = percent + 1
      wjsLog ("set to new value =" + setToValue)
      imeGeneratorEnvelopePercentSet (generatorIndex, "Volume", percent, setToValue)

      var valueNow = imeGeneratorEnvelopePercentGet (generatorIndex, "Volume", percent
      wjsLog ("Volume at " + percent + ", " +  "now=" +  valueNow + "\n")

      percent = percent + 1
    }
  }
}
```

```
imeGeneratorEnvelopePercentSet (generatorObjectIndex, parameterName, percent, newValue
```

This function sets the envelope value at the specified percent, for the indexed generator, to the specified value. The value must be between 0 and 127.

The function has the following parameters:

- *generatorObjectIndex* the index of the generator object of interest.
- *parameterName* the name of the envelope parameter (e.g. *Volume*).
- *percent* the percent value; from 0 to 100.
- *newValue* the new element value to use, in a range from 0 to 127.

Example:

```
function onImeStart() {
  var generatorCount = imeObjectCountGet("Generator")
  wjsLog ("Generator objects=" + generatorCount)

  for (var generatorIndex = 0; generatorIndex < generatorCount; generatorIndex++) {
    // Ramp-up every generator volume envelope!
    var generatorName = imeObjectNameGet("Generator", generatorIndex)
    wjsLog ("Generator=" + generatorName)
```

```
    var percent = 0
    while (percent <= 100)
    {
      var valueWas = imeGeneratorEnvelopePercentGet (generatorIndex, "Volume", percent
      wjsLog ("Volume at " + percent + ", " +  "was=" + valueWas)

      var setToValue = percent + 1
      wjsLog ("set to new value =" + setToValue)
      imeGeneratorEnvelopePercentSet (generatorIndex, "Volume", percent, setToValue)

      var valueNow = imeGeneratorEnvelopePercentGet (generatorIndex, "Volume", percent
      wjsLog ("Volume at " + percent + ", " +  "now=" +  valueNow + "\n")

      percent = percent + 1
    }
  }
}
```

`imeCellTrackIndexGet()`

This function returns the track index of the Mix Cell containing the current generator/mix. This is used with various ISE related functions.

The function has no parameters

Example:

```
function onImeStart() {

  var cellTrackIndex = imeCellTrackIndexGet()

}
```

`imeCellColumnIndexGet()`

This function returns the column index of the Mix Cell containing the current generator/mix. This is used with various ISE related functions.

The function has no parameters

Example:

```
function onImeStart() {
  var cellColumnIndex = imeCellColumnIndexGet()
}
```

ISE functions

ISE related functions all start with the *ise* prefix

```
iseGetFxNetworkIdForMix()
```

This function returns the id that can be used for manipulating the effects chain for the Global Mix FX.

The function has no parameters

Example:

```
function onImeStart() {
  var fxNetworkIdForMix = iseGetFxNetworkIdForMix()
}
```

```
iseGetFxNetworkIdForCellGlobal(trackIndex, columnIndex)
```

This function returns the id that can be used for manipulating the effects chain for the Cell, identified by the supplied track and column indexes.

The function has the following parameters:

- *trackIndex* The track index of the cell (from 0 to 11).
- *columnIndex* The columnIndex index of the cell (from 0 to 3).

Example:

```
function onImeStart() {

  var fxNetworkIdForCellAtTrack0Column1 = iseGetFxNetworkIdForCellGlobal(0, 1)

}
```

```
iseGetFxNetworkIdForCellSynthTrackColumnChannel(trackIndex, columnIndex, channel)
```

This function returns the id that can be used for manipulating the effects chain for the Cell, identified by the supplied track and column indexes.

The function has the following parameters:

- *trackIndex* The track index of the cell (from 0 to 11).
- *columnIndex* The columnIndex index of the cell (from 0 to 3).
- *channel* the MIDI Channel, from 0 to 15 (MIDI channel is actually displayed in the IME Network screen as 1 to 16)

Example:

```
function onImeStart() {

  var synthFxNetworkIdForCellAtTrack0Column1MIDIChannel2 = iseGetFxNetworkIdForCellSyn

}
```

```
iseGetFxNetworkIdForCellEffectTrackColumnChannel()
```

This function returns the id that can be used for manipulating the effects chain for the Cell, identified by the supplied track and column indexes.

The function has the following parameters:

- *trackIndex* The track index of the cell (from 0 to 11).
- *columnIndex* The columnIndex index of the cell (from 0 to 3).
- *channel* the MIDI Channel, from 0 to 15 (MIDI channel is actually displayed in the IME Network screen as 1 to 16)

Example:

```
function onImeStart() {

  var trackFxNetworkIdForCellAtTrack0Column1MIDIChannel2 = iseGetFxNetworkIdForCellEff

}
```

```
iseFxNetworkAllPresetNamesGet(fxNetworkId)
```

This function returns an array of all FX presets that can be supplied to the FX network with the specified ID. The list returned using an FX Network ID generated by *iseGetFxNetworkIdForCellSynthTrackColumnChannel* is different to those using other FX network IDs (which all otherwise return the same array of values)

The function has the following parameters:

- *fxNetworkId* The id of the FX Network in question.

Example:

```
function onImeStart() {

  var fxNetworkIdForMix = iseGetFxNetworkIdForMix()
  var allEffectsPresets = iseFxNetworkAllPresetNamesGet(fxNetworkIdForMix)

  // Display the preset names on the console
  for (var effectPresetIndex = 0; effectPresetIndex < allEffectsPresets.length; effectI
    wjsLog(allEffectsPresets.length[effectPresetIndex])
  }

  var synthFxNetworkId = iseGetFxNetworkIdForCellSynthTrackColumnChannel(0, 0, 0)
  var allSynthPresets = iseFxNetworkAllPresetNamesGet(synthFxNetworkId)

  // Display the preset names on the console
  for (var synthFxIndex = 0; synthFxIndex < allSynthPresets.length; synthFxIndex++) {
    wjsLog(allSynthPresets.length[synthFxIndex])
  }
}
```

```
iseFxNetworkRandomPresetNamesGet(fxNetworkId)
```

This function returns an array of all FX presets that can be supplied to the FX network with the specified ID; filtered such that only the presets that you have enabled in your current randomization scheme are returned. The list returned using an FX Network ID generated by *iseGetFxNetworkIdForCellSynthTrackColumnChannel* is different to those using other FX network IDs (which all otherwise return the same array of values)

The function has the following parameters:

- *fxNetworkId* The id of the FX Network in question.

Example:

```
function onImeStart() {

  var fxNetworkIdForMix = iseGetFxNetworkIdForMix()
  var allEffectsPresets = iseFxNetworkRandomPresetNamesGet(fxNetworkIdForMix)

  // Display the preset names on the console
  for (var effectPresetIndex = 0; effectPresetIndex < allEffectsPresets.length; effectI
    wjsLog(allEffectsPresets.length[effectPresetIndex])
  }

  var synthFxNetworkId = iseGetFxNetworkIdForCellSynthTrackColumnChannel(0, 0, 0)
  var allSynthPresets = iseFxNetworkRandomPresetNamesGet(synthFxNetworkId)

  // Display the preset names on the console
  for (var synthFxIndex = 0; synthFxIndex < allSynthPresets.length; synthFxIndex++) {
    wjsLog(allSynthPresets.length[synthFxIndex])
  }
}
```

```
iseFxNetworkPresetSet(fxNetworkId, fxPreset)
```

This function applies the specified FX Preset to the specified FX Network

The function has the following parameters:

- *fxNetworkId* The id of the FX Network in question.
- *fxPreset* The preset to apply to the FX Network

Example:

```
function onImeStart() {

  var fxNetworkIdForMix = iseGetFxNetworkIdForMix()
```

```
  iseFxNetworkPresetSet(fxNetworkIdForMix, "fx/Filter/Filter 4Hz autowah.fxm")
}
```

```
imeCellTrackIndexGet()
```

This function returns the cell index for the current mix.

The function has the following parameters:

- *fxNetworkId* The id of the FX Network in question.
- *fxPreset* The preset to apply to the FX Network

Example:

```
function onImeStart() {

  var fxNetworkIdForMix = iseGetFxNetworkIdForMix()
  iseFxNetworkPresetSet(fxNetworkIdForMix, "fx/Filter/Filter 4Hz autowah.fxm")
}
```

ITE functions

ITE related functions all start with the *ite* prefix

```
iteRandomWordsGet (numberOfWords)
```

This function returns a string containing a number of random words.

The function has the following parameters:

- *numberOfWords* The number of random words to return.

Example:

```
function onImeStart() {
```

```
  // Make a string with 3 random words
  var ttmText = iteRandomWordsGet(3)

  // Set the TTM text using our random string
  imeParameterSet("Text Pattern Text", ttmText)
}
```

```
iteShuffleWords (textToShuffle)
```

This function returns a string containing a shuffled version of the suppled text. The words preserved from the supplied text, but the order of those words is changed.

The function has the following parameters:

- *textToShuffle* The text string to shuffle.

Example:

```
function onImeStart() {

  // Make a string that is a random shuffle of words
  var textToShuffle = "Shuffle these four words"
  var ttmText = iteShuffleWords(textToShuffle)

  // Set the TTM text using our shuffled string
  imeParameterSet("Text Pattern Text", ttmText)
}
```

```
iteShuffleCharacters (textToShuffle)
```

This function returns a string containing a shuffled version of the suppled text. All the characters are shuffled; words are broken-up.

The function has the following parameters:

- *textToShuffle* The text string to shuffle.

Example:

```
function onImeStart() {
```

```
  // Make a string that is a random shuffle of characters
  var textToShuffle = "Shuffle these characters"
  var ttmText = iteShuffleCharacters(textToShuffle)

  // Set the TTM text using our shuffled string
  imeParameterSet("Text Pattern Text", ttmText)
}
```

Trigger Script Cookbook

A great way to start thinking about Event Handler Script Functions, is to look at various real examples of how to do various interesting things with Event Handler Script Functions!

Example: How to choose a random Mix Root every time the mix starts

```
// Choose a random root

function chooseRandomRoot() {
  var roots = ["A", "A#", "B", "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#"]

  var rootIndex = wjsRandomGetFromTo(0, roots.length-1)
  var newRoot = roots[rootIndex]
  return newRoot
}

function onImeStart() {
  wjsLog ("onImeStart!")

  var newRoot = chooseRandomRoot()
  imeParameterSet("Mix Root", newRoot)
  wjsLog ("Mix Root now=" + newRoot)
}
```

Example: Change the scale to use depending on time of day

This is a script that you could use as a cell-level *Bar* trigger script.

```
function onImeStart() {
  var date = new Date()
```

```
  var hour = date.getHours()
  if ((hour > 20) || (hour <8)) {
    imeParameterSet("Scale Rules", "early morning scale")
  } else {
    imeParameterSet("Scale Rules", "middle of day scale")
  }
}
```

Example: randomly select a pattern to use for a generator

This is a generator-level *Bar* trigger script.

```
function onImeStart() {
  // Get a random integer between 1 and 3 inclusive.
  var value = wjsRandomGetFromTo (1, 3)

  wjsLog ("value=" + value)

  var patternString = ""
  if (value == 1) {
    patternString = "<100 B 240 1>"
  } else if (value == 2) {
    patternString = "<100 B 120 1 120 2>"
  } else {
    patternString = "<100 B 60 1 60 2 60 3 60 4>"
  }

  imeParameterSet("Patterns", patternString)
}
```

Example: script trigger that allows Event Handler Script Functions to emit MIDI CC events, with an optional delay

This is a script that you could use as a cell-level *Bar* trigger script.

```
function onImeBar(bar) {

  // Generator trigger...
  // Apply a pan sweep through the bar from left to right,
  // to show-off the use of imeMIDISendCC.

  wjsLog ("onImeBar(" + index + ") - Generator")

  var barDuration = imeBarDurationGet()

  wjsLog ("barDuration=" + barDuration)
```

```
  var midiChannel = imeGeneratorMIDIChannelGet()
  var delay = 0
  var ccController = 10 // Pan controller!
  var value = 0
  while (delay <= barDuration) {
    var value = (127 * delay) / barDuration
    // Note that the "delay" is optional; we're using the in this specific
    // demo, to get a sweep effect from start to end of the bar.
    imeMIDISendCC(midiChannel, ccController, value, delay)
    delay = delay + 20
  }
}
```

Example: cell bar event trigger, that automatically adjusts the mix root and tempo every 2 bars

```
function onImeBar(bar) {

  // Cell trigger...
  // Change bar / tempo every 2 bars...

  wjsLog ("onImeBar(" + bar + ") - Mix")

  if ( ((bar + 1) % 2) == 0) {
    wjsLog ("bar=" + bar)

    var rootWas = imeParameterGet("Mix Root")

    wjsLog ("Mix Root was=" + rootWas)

    // Every 2 bars, change the root at random,
    // from all those available!
    var roots = ["A", "A#", "B", "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#"]

    var rootIndex = wjsRandomGetFromTo(0, roots.length-1)
    var newRoot = roots[rootIndex]
    wjsLog ("New root=" + newRoot)

    imeParameterSet("Mix Root", newRoot)

    // Set tempo as well; to a value related to the selected root!
    var tempoWas = imeParameterGet("Tempo")
    wjsLog ("Mix tempo was=" + tempoWas)

    var newTempo = 100 + rootIndex * 5
    wjsLog ("New tempo=" + newTempo)
    imeParameterSet("Tempo", newTempo)
  }
```

```
}
```

Example: cell bar event trigger, that automatically adjusts the Mix Root and Scale Rule every 2 bars

```
// Choose a random root

function chooseRandomRoot() {
  var roots = ["A", "A#", "B", "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#"]

  var rootIndex = wjsRandomGetFromTo(0, roots.length-1)
  var newRoot = roots[rootIndex]
  return newRoot
}

// Check if the specified Scale Rule name exists in the mix

function getDoesScaleRuleExistWithThisName(lookForRuleName) {
  var scaleRuleCount = imeObjectCountGet("Scale Rule")
  //wjsLog ("Scale Rules objects=" + scaleRuleCount)

  for (var index = 0; index < scaleRuleCount; index++) {
    var scaleRuleName = imeObjectNameGet("Scale Rule", index)

    // wjsLog ("scaleRuleName=" + scaleRuleName)

    if (lookForRuleName == scaleRuleName) {
      // Found the rule, so return true.
      return true
    }
  }

  // To get here, the rule wasn't found, so return false.

  return false
}


// Event Triggers

function onImeBar(bar) {

  // Cell trigger...
  // Change bar / scale rule every 2 bars...
  // Note that this requires you to have at least two scale rules;
  // one called "Major" and one called "Minor"

  wjsLog ("onImeBar(" + bar + ") - Mix")
```

```
  // Every 2 bars, change the root at random, from all those available!
  if ( ((bar + 1) % 2) == 0) {
    wjsLog ("bar=" + bar)

    var rootWas = imeParameterGet("Mix Root")

    wjsLog ("Mix Root was=" + rootWas)
    var newRoot = chooseRandomRoot()
    imeParameterSet("Mix Root", newRoot)
    wjsLog ("Mix Root now=" + newRoot)

    // Set scale as well; to a value related to the selected root.

    // Choose one of two Scale Rule names at random.

    var useRuleName = ""
    if (wjsRandomGetFromTo(0, 100) <= 50) {
      useRuleName = "Major"
    } else {
      useRuleName = "Minor"
    }

    if (getDoesScaleRuleExistWithThisName(useRuleName)) {
      wjsLog ("Mix Scale Rule set to=" + useRuleName)

      // Apply rule to all generators!
      var generatorCount = imeObjectCountGet("Generator")
      // wjsLog ("Generator objects=" + generatorCount)

      for (var generatorIndex = 0; generatorIndex < generatorCount; generatorIndex++)
        // wjsLog ("Generator name=" + imeObjectNameGet("Generator", generatorIndex))
        imeObjectParameterSet("Generator", generatorIndex, "Scale Rules", useRuleName)
      }
    } else {
      wjsLog("Warning - rule name not found=" + useRuleName)
    }
  }
}
```

Example: cell start event trigger, that automatically adjusts the tempo to suit the time of day

```
function onImeStart() {
  wjsLog ("Mix!")

  // Wotja script to get hour of day as 24-hour clock value
  // ... and adjust tempo accordingly!
```

```
  var date = new Date()

  var hour = date.getHours()
  if ((hour > 20) || (hour <8)) {
    wjsLog ("late night / early morning!")
    var randomValue = wjsRandomGetFromTo(0,50)
    imeParameterSet("Tempo", 50 + randomValue)
  } else {
    wjsLog ("middle of day!")
    var randomValue = wjsRandomGetFromTo(0,50)
    imeParameterSet("Tempo", 100 + randomValue)
  }
}
```

Example: generator start event trigger, that automatically cycles the Chord Depth every time it starts playback

```
function onImeStart() {
  wjsLog ("Generator!")

  var chordDepth = imeParameterGet("Chord Depth")

  // Convert any string value to integer, then add one.
  chordDepth = parseInt(chordDepth) + 1

  // Loop the value around if it gets larger than we want.
  if (chordDepth > 8) {
    chordDepth = 1
  }
  imeParameterSet("Chord Depth", chordDepth)
}
```

Example: generator event trigger, that automatically changes Chord Depth every bar

```
function onImeBar(bar) {
  wjsLog ("onImeBar, bar=" + bar)
  var chordDepth = imeParameterGet("Chord Depth")

  // Convert any string value to integer, then add one.
  chordDepth = parseInt(chordDepth) + 1

  // Loop the value around if it gets larger than we want.
  if (chordDepth > 8) {
    chordDepth = 1
  }
```

```
  imeParameterSet("Chord Depth", chordDepth)
}
```

Example: bringing generators into the mix

In this example, imagine that you have a mutating drum Generator that you want to have muted at the start of the mix and come in with a bang at the start of bar 20. And you don't want the mutations to start until bar 21...

How would that work in the world of script?

Here is the Generator Start trigger:

```
function onImeStart() {
  // When we start, disable mutation for this Generator!
  // And set mute, too!
  imeParameterSet("Mute", "Yes")
  imeParameterSet("Mutation Factor", 0)
}
```

Here is the Generator Bar trigger:

```
function onImeBar(bar) {
  if (bar == 20) {
    // At bar 20, unmute the generator!
    imeParameterSet("Mute", "No")
  }

  if (bar == 21) {
    // At bar 21, adjust the mutation factor!
    imeParameterSet("Mutation Factor", 20)
  }
}
```

The power of using scripts like this, is that:

- The drums alway start on the bar specified.
- They don't morph until they are audible.
- The drum elements can now make staggered entry despite all being on the same MIDI channel.

Example: Emit MIDI CC in response to composed note events

```
var GLastNote = null
```

```
function onImeGeneratorComposed(generatorIndex, noteon, channel, pitch, velocity) {
  // Keep track of last composed note using the global (for this generator)
  // called GLastNote ...
  // Note that the "pitch" parameter should be ignored when noteon is false.

  midiChannel = imeGeneratorMIDIChannelGet()

  if (noteon) {
    if (GLastNote != null) {
      imeMIDISendCC(midiChannel, 11, GLastNote, 10)
    }

    GLastNote = pitch
    imeMIDISendCC(midiChannel, 11, pitch, 0)
  } else {
    if (GLastNote != null) {
      imeMIDISendCC(midiChannel, 11, GLastNote, 10)
      GLastNote = null
    }
  }
}
```

Example: Change TTM Text for Generator At Start of Mix

```
function makeRandomText() {
  // Make some random TTM text.
  // We show 3 different techniques!
  // Choose one of these to use at random.
  var technique = wjsRandomGetFromTo(0, 2)
  var ttmText = ""

  switch (technique) {
  case 0:
    // Make a string with 3 random words
    ttmText = iteRandomWordsGet(3)
    break
  case 1:
    // Make a string that is a random shuffle of words
    ttmText = "Shuffle these four words"
    ttmText = iteShuffleWords(ttmText)
    break
  default:
    // Make a string that is a random shuffle of characters
    ttmText = "Shuffle these characters"
    ttmText = iteShuffleCharacters(ttmText)
    break
  }
```

```
  wjsLog("random ttmText=", ttmText)

  return ttmText
}

function onImeStart() {

  // Generate some random TTM text!
  var ttmText = makeRandomText()

  // Now that we have our string, set it!
  imeParameterSet("Text Pattern Text", ttmText)
}
```

Example: Change TTM Repeats at Start

```
function onImeStart() {
  // Set the TTM property "Text Pattern Repeats" to 2

  imeParameterSet("Text Pattern Repeats", 2)
}
```

Example: Change TTM Text At Start Then Restore After 4 bars

```
// Complex scripting example:
// - At mix start, apply random TTM Text.
// - After 4 bars, or at mix stop - restore the original TTM Text.

// The original TTM text is stored in this "global" variable
var GOriginalTTMText = ""

// The value we last set in onImeStart
var GLastSetTTMText = ""

function setRandomTTMText() {
  // Take a copy of the original TTM text
  GOriginalTTMText = imeParameterGet("Text Pattern Text")
  wjsLog("GOriginalTTMText=" + GOriginalTTMText)

  // Make a string with 3 random words
  var ttmText = iteRandomWordsGet(3)
  wjsLog("ttmText=" + ttmText)
```

```
  GLastSetTTMText = ttmText

  // Now that we have our string, set it!
  imeParameterSet("Text Pattern Text", ttmText)
}

function restoreOriginalTTMText() {

  var ttmNow = imeParameterGet("Text Pattern Text")
  wjsLog("restoreOriginalTTMText, ttmNow=" + ttmNow)

  if (ttmNow != GLastSetTTMText) {
    wjsLog("The user value has changed since last random set - keep the current value"
  } else {
    wjsLog("Restoring TTM Text to=" + GOriginalTTMText)
    imeParameterSet("Text Pattern Text", GOriginalTTMText)
  }
}

//
// Trigger script functions
//

function onImeStart() {

  wjsLog("onImeStart")

  setRandomTTMText()
}

function onImeBar(bar) {
  // On the 4th bar, restore the TTM text... UNLESS the user
  // has already changed the value through the UI.

  wjsLog("onImeBar, bar=" + bar)

  if (bar == 4) {

    wjsLog("onImeBar, bar 4!")

    restoreOriginalTTMText()
  }
}

function onImeStop() {
  wjsLog("onImeStop")
  // At the end, restore the TTM text... UNLESS the user
  // has already changed the value through the UI.
  restoreOriginalTTMText()
}
```

Example: Cell: Query All Synth and Track Presets at Start

```
function onImeStart() {

  // Query what Synth and Effect presets are available, and list them in the Script Co

  var trackIndex = 0
  var columnIndex = 0
  var channel = 1

  var fxNetworkIdSynth = iseGetFxNetworkIdForCellSynthTrackColumnChannel(trackIndex, c
  var fxNetworkIdTrack = iseGetFxNetworkIdForCellEffectTrackColumnChannel(trackIndex,

  var synthFxPresets = iseFxNetworkAllPresetNamesGet(fxNetworkIdSynth)
  wjsLog("All synthFxPresets.length=" + synthFxPresets.length)

  for (var synthFxIndex = 0; synthFxIndex < synthFxPresets.length; synthFxIndex++) {
    wjsLog(synthFxPresets[synthFxIndex])
  }

  var trackFxPresets = iseFxNetworkAllPresetNamesGet(fxNetworkIdTrack)
  wjsLog("All trackFxPresets.length=" + trackFxPresets.length)

  for (var trackFxIndex = 0; trackFxIndex < trackFxPresets.length; trackFxIndex++) {
    wjsLog(trackFxPresets[trackFxIndex])
  }
}
```

Example: Cell: Query And Set Random Effects and Patches

```
function onImeStart() {

  // Generate a set of FX Network IDs, that are used for querying and setting differen
  var fxNetworkIdMix = iseGetFxNetworkIdForMix()
  var fxNetworkIdMixTrack0 = iseGetFxNetworkIdForMixTrack(0)
  var fxNetworkIdCellGlobalTrack0Column0 = iseGetFxNetworkIdForCellGlobal(0,0)
  var fxNetworkIdCellSynthTrack0Column0Channel1 = iseGetFxNetworkIdForCellSynthTrackCo
  var fxNetworkIdCellEffectTrack0Column0Channel1 = iseGetFxNetworkIdForCellEffectTrack

  // Query what Synth and Effect presets are available, and list them in the Script Co

  var trackFxPresets = iseFxNetworkRandomPresetNamesGet(fxNetworkIdMixTrack0)
  wjsLog("trackFxPresets.length=" + trackFxPresets.length)
```

```
  for (var trackFxIndex = 0; trackFxIndex < trackFxPresets.length; trackFxIndex++) {

    wjsLog("trackFxPreset=" + trackFxPresets[trackFxIndex])
  }

  var synthFxPresets = iseFxNetworkRandomPresetNamesGet(fxNetworkIdCellSynthTrack0Colu
  wjsLog("synthFxPresets.length=" + synthFxPresets.length)

  for (var synthFxIndex = 0; synthFxIndex < synthFxPresets.length; synthFxIndex++) {
    wjsLog("synthFxPreset=" + synthFxPresets[synthFxIndex])
  }

  if (synthFxPresets.length > 0) {
    // Set a random synth patch (from those available) for track 0, column 0, channel
    // Now... set patch random for cell 1!
    var synthPresetForTrack0Column0Channel1 = synthFxPresets[wjsRandomGetFromTo(0,synth
    wjsLog("Use synthPresetForTrack0Column0Channel1=" + synthPresetForTrack0Column0Cha

    iseFxNetworkPresetSet(fxNetworkIdCellSynthTrack0Column0Channel1, synthPresetForTra
  }

  if (trackFxPresets.length > 0) {
    // Set a random FX patch (from those available) for the global Mix FX
    var fxForMix = trackFxPresets[wjsRandomGetFromTo(0,trackFxPresets.length-1)]
    wjsLog("Use fxForMix=" + fxForMix)
    iseFxNetworkPresetSet(fxNetworkIdMix, fxForMix)
  }
}
```

Example: Cell: Query And Set Random Global Preset at Start

```
function onImeStart() {

  // Generate the FX Network ID ... for the Global Mix FX
  var fxNetworkIdMix = iseGetFxNetworkIdForMix()

  // Query what Effect presets are available... for the Global Mix FX
  var trackFxPresets = iseFxNetworkRandomPresetNamesGet(fxNetworkIdMix)

  // Set a random FX patch (from those available) for the global Mix FX
  // Check - has the user got any suitable FX enabled?
  if (trackFxPresets.length > 1) {
    // Yes, we have at least one suitable FX preset available

    // Choose one at random...
    var randomIndex = wjsRandomGetFromTo(0,trackFxPresets.length-1)
    var fxForMix = trackFxPresets[randomIndex]
```

```
    wjsLog("Use fxForMix=" + fxForMix)

    // Apply the randomly selected preset
    iseFxNetworkPresetSet(fxNetworkIdMix, fxForMix)
  }
}
```

Example: Generator Query And Set Random Synth and Track Preset at Start

```
function onImeStart() {

  // Generator trigger script - apply a random Synth patch at start

  // Generate the FX Network ID ... for the Generator's cell and Generator's MIDI chan

  var trackIndex = imeCellTrackIndexGet()
  var columnIndex = imeCellColumnIndexGet()
  var channel = imeGeneratorMIDIChannelGet()
  wjsLog("Generator trackIndex=" + trackIndex + ", columnIndex=" + columnIndex + ", ch

  var fxNetworkIdSynth = iseGetFxNetworkIdForCellSynthTrackColumnChannel(trackIndex, c
  var fxNetworkIdTrack = iseGetFxNetworkIdForCellEffectTrackColumnChannel(trackIndex,

  // Query what Synth and Effect presets are available, and list them in the Script Co

  var synthFxPresets = iseFxNetworkRandomPresetNamesGet(fxNetworkIdSynth)
  wjsLog("synthFxPresets.length=" + synthFxPresets.length)

  for (var synthFxIndex = 0; synthFxIndex < synthFxPresets.length; synthFxIndex++) {
    wjsLog("synthFxPreset=" + synthFxPresets[synthFxIndex])
  }

  var trackFxPresets = iseFxNetworkRandomPresetNamesGet(fxNetworkIdTrack)
  wjsLog("trackFxPresets.length=" + trackFxPresets.length)

  for (var trackFxIndex = 0; trackFxIndex < trackFxPresets.length; trackFxIndex++) {
    wjsLog("trackFxPreset=" + trackFxPresets[trackFxIndex])
  }

  if (synthFxPresets.length > 0) {
    // Set a random synth patch (from those available) for track 0, column 0, channel
    // Now... set patch random for cell 1!
    var synthPreset = synthFxPresets[wjsRandomGetFromTo(0,synthFxPresets.length-1)]
    wjsLog("Use synthPreset=" + synthPreset)

    iseFxNetworkPresetSet(fxNetworkIdSynth, synthPreset)
  }
```

```
  if (trackFxPresets.length > 0) {
    // Set a random FX patch (from those available) for the global Mix FX
    var trackPreset = trackFxPresets[wjsRandomGetFromTo(0,trackFxPresets.length-1)]
    wjsLog("Use fxForMix=" + trackPreset)
    iseFxNetworkPresetSet(fxNetworkIdTrack, trackPreset)
  }
}
```

Example: Mix Level script, showing various functions

```
// Scripting API methods related mix-level functions - various examples.

function mixObjectParameterGetAndSet() {
  // mixObjectParameterGet
  // mixObjectParameterSet

  // Example usage:
  var scaleRuleValue = mixObjectParameterGet("Scale Rule", 0, "Value")
  print("Mix scale rule index 0, value was = " + scaleRuleValue)
  scaleRuleValue = mixObjectParameterSet("Scale Rule", 0, "Value", "1.0000 0.4961 0.(
  scaleRuleValue = mixObjectParameterGet("Scale Rule", 0, "Value")
  print("Mix scale rule index 0, value now = " + scaleRuleValue)
}

function mixObjectQuerying() {
  // mixObjectNameGet
  // mixObjectIndexGet
  // mixObjectCountGet

  // Example usage:
  var scaleRuleCount = mixObjectCountGet("Scale Rule")
  print("Mix scale rule count = " + scaleRuleCount)
  var scaleRuleName = mixObjectNameGet("Scale Rule", 0)
  print("Mix scale rule 0 has name = " + scaleRuleName)
  var indexOfScaleRuleName = mixObjectIndexGet("Scale Rule", scaleRuleName)
  print("Mix scale rule with name=" + scaleRuleName + " is at index=" + indexOfScaleI
}

function mixCellObjectQuerying() {
  // mixCellObjectNameGet
  // mixCellObjectIndexGet
  // mixCellObjectCountGet

  // Example usage:
  var columnIndex = 0
  var trackIndex = 0
  var scaleRuleCount = mixCellObjectCountGet(columnIndex, trackIndex, "Scale Rule")
  print("Mix cell(" + columnIndex + "," + trackIndex + ") : scale rule count = " + s(
  var scaleRuleName = mixCellObjectNameGet(columnIndex, trackIndex, "Scale Rule", 0)
```

```
  print("Mix cell(" + columnIndex + "," + trackIndex + ") : scale rule 0 has name =
  var indexOfScaleRuleName = mixCellObjectIndexGet(columnIndex, trackIndex, "Scale R
  print("Mix cell(" + columnIndex + "," + trackIndex + ") : scale rule with name=" +
}

function mixCellParameterGetAndSet() {
  // mixCellObjectParameterSet
  // mixCellObjectParameterGet

  // Example usage:
  var columnIndex = 0
  var trackIndex = 0
  var generatorIndex = 0
  var generatorValue = mixCellObjectParameterGet(columnIndex, trackIndex, "Generator
  print("generatorValue = " + generatorValue)
  mixCellObjectParameterSet(columnIndex, trackIndex, "Generator", generatorIndex, "S
  generatorValue = mixCellObjectParameterGet(columnIndex, trackIndex, "Generator", g
  print("generatorValue = " + generatorValue)
  var cellValue = mixCellObjectParameterGet(columnIndex, trackIndex, "Cell", "Scale
  print("cellValue = " + cellValue)
  mixCellObjectParameterSet(columnIndex, trackIndex, "Cell", "Scale Rules", "Mix\nDe
  cellValue = mixCellObjectParameterGet(columnIndex, trackIndex, "Cell", "Scale Rule
  print("cellValue = " + cellValue)
}

function mixParameterGetAndSet() {
  // mixParameterSet
  // mixParameterGet

  // Example usage:
  var mixValue = mixParameterGet("Scale Rules")
  print("Mix: Scale Rules = " + mixValue)
  mixParameterSet("Scale Rules", "Default")
  mixValue = mixParameterGet("Scale Rules")
  print("Mix: Scale Rules = " + mixValue)

  mixValue = mixParameterGet("Mix Root")
  print("Mix: Mix Root = " + mixValue)
  mixParameterSet("Mix Root", "C")
  mixValue = mixParameterGet("Mix Root")
  print("Mix: Mix Root = " + mixValue)

  mixValue = mixParameterGet("Tempo")
  print("Mix: Tempo = " + mixValue)
  mixParameterSet("Tempo", "100")
  mixValue = mixParameterGet("Tempo")
  print("Mix: Tempo = " + mixValue)
}

function mixVolumeGetAndSet() {
  // mixVolumeSet
  // mixVolumeGet
```

```
  // Example usage:
  var value = mixVolumeGet() // Get current value, from 0 to 127 (127 is default)
  print(value)
  mixVolumeSet(0)   // Set to minimum volume
  value = mixVolumeGet()
  print(value)
  mixVolumeSet(127) // Set to maximum volume
  value = mixVolumeGet()
  print(value)
}

function mixTrackVolumeGetAndSet() {
  // mixTrackVolumeSet
  // mixTrackVolumeGet

  // Example usage:
  var trackIndex = 0
  var value = mixTrackVolumeGet(trackIndex) // Value from 0 to 127, 127 is max, 0 is
  print("Track volume = " + value)
  mixTrackVolumeSet(trackIndex, 100) // Value from 0 to 127, 127 is max, 0 is silent
  value = mixTrackVolumeGet(trackIndex) // Value from 0 to 127, 127 is max, 0 is sil
  print("Track volume = " + value)
}

function mixTrackPanGetAndSet() {
  // mixTrackPanSet
  // mixTrackPanGet

  // Example usage:
  var trackIndex = 0
  var value = mixTrackPanGet(trackIndex) // Value from 0 to 127, 127 is max, 0 is si
  print("Track pan = " + value)
  mixTrackPanSet(trackIndex, 64) // Value from 0 to 127, 127 is max, 0 is silent
  value = mixTrackPanGet(trackIndex) // Value from 0 to 127, 127 is max, 0 is silent
  print("Track pan = " + value)
}
```

Example: Mix Level script, showing various functions and triggers

```
//
// Example of script functions and triggers to cycle the Mix Default rule and root,
// through the progression used in Chopin's 24 preludes.
//

const cAllScaleRules = {

  "Major":
    "1 0 0.5 0 1 0.5 0 1 0 0.5 0 0.5",
  "Minor":
    "1 0 1 1 0 1 0 1 1 0 1 0",
```

```
  "All Scale Major":
    "1 0 1 0 1 1 0 1 0 1 0 1",
  "All Scale Minor Natural":
    "1 0 1 1 0 1 0 1 1 0 1 0",
  "All Scale Minor Harmonic":
    "1 0 1 1 0 1 0 1 0 0 0 1",
  "Chord Major triad":
    "1 0 0 0 1 0 0 1 0 0 0 0",
  "Chord Major sixth":
    "1 0 0 0 1 0 0 1 0 1 0 0",
  "Chord Dominant seventh":
    "1 0 0 0 1 0 0 1 0 0 1 0",
  "Chord Major seventh":
    "1 0 0 0 1 0 0 1 0 0 0 1",
  "Chord Augmented triad":
    "1 0 0 0 1 0 0 0 1 0 0 0",
  "Chord Augmented seventh":
    "1 0 0 0 1 0 0 0 1 0 1 0",
  "Chord Minor triad":
    "1 0 0 1 0 0 0 1 0 0 0 0",
  "Chord Minor sixth":
    "1 0 0 1 0 0 0 1 0 1 0 0",
  "Chord Minor seventh":
    "1 0 0 1 0 0 0 1 0 0 1 0",
  "Chord Minor-major seventh":
    "1 0 0 1 0 0 0 1 0 0 0 1",
  "Chord Diminished triad":
    "1 0 0 1 0 0 1 0 0 0 0 0",
  "Chord Diminished seventh":
    "1 0 0 1 0 0 1 0 0 1 0 0",
  "Chord Half-diminished seventh":
    "1 0 0 1 0 0 1 0 0 0 1 0",
  "Chord Augmented major seventh":
    "1 0 0 0 1 0 0 0 1 0 0 1",
  "Chord Seven-six":
    "1 0 0 0 1 0 0 1 0 0 1 0",
  "Chord Mixed-third":
    "1 0 0 1 1 0 0 1 0 0 0 0",
  "Chord Suspended fourth":
    "1 0 0 0 0 1 0 1 0 0 0 0",
  "Chord Dominant ninth":
    "1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0",
  "Chord Dominant eleventh":
    "1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0",
  "Chord Dominant thirteenth":
    "1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0",
  "Chord Seventh minor ninth":
    "1 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0",
  "Chord Seventh sharp ninth":
    "1 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0",
  "Chord Seventh augmented eleventh":
    "1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0",
  "Chord Seventh diminished thirteenth":
```

```
    "Chord Seventh diminished thirteenth":
      "1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 0",
    "Chord Add nine":
      "1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0",
    "Chord Add fourth":
      "1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0",
    "Chord Add sixth":
      "1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0",
    "Chord Six-nine":
      "1 0 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0",
    "Chord Suspended second":
      "1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0",
    "Chord Jazz sus":
      "1 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0",
    "Mode Ionian (I)":
      "1 0 1 0 1 1 0 1 0 1 0 1",
    "Mode Dorian (II)":
      "1 0 1 1 0 1 0 1 0 1 1 0",
    "Mode Phrygian (III)":
      "1 1 0 1 0 1 0 1 1 0 1 0",
    "Mode Lydian (IV)":
      "1 0 1 0 1 0 1 1 0 1 0 1",
    "Mode Mixolydian (V)":
      "1 0 1 0 1 1 0 1 0 1 1 0",
    "Mode Aeolian (VI)":
      "1 0 1 1 0 1 0 1 1 0 1 0",
    "Mode Locrian (VII)":
      "1 1 0 1 0 1 1 0 1 0 1 0"
}

const cProgression = [
    "C Major",
    "A Minor",
    "G Major",
    "E Minor",
    "D Major",
    "B Minor",
    "A Major",
    "F# Minor",
    "E Major",
    "C# Minor",
    "B Major",
    "G# Minor",
    "F# Major",
    "Eb Minor",
    "Db Major",
    "Bb Minor",
    "Ab Major",
    "F Minor",
    "Eb Major",
    "C Minor",
    "Bb Major",
    "G Minor",
    "F Major"
```

```
    "F Major",
    "D Minor"]

  const cChangeEveryBars = 4

  var cycleIndex = 0; // We cycle round from 0 through to cProgression.length-1, and ba

  function applyScaleName(scaleName) {
    var newRuleValue = cAllScaleRules[scaleName]

    mixObjectParameterSet("Scale Rule", 0, "Value", newRuleValue)

    var newRuleValue = mixObjectParameterGet("Scale Rule", 0, "Value")
    wjsLog ("Scale Rule Value (" + scaleName + "(, now=" + newRuleValue)
  }

  function applyRandomTempo() {
    var newTempo = 60 + wjsRandomGetFromTo(0, 60)

    mixParameterSet("Tempo", newTempo)

    var testTempo = mixParameterGet("Tempo")
    wjsLog ("Mix tempo=" + testTempo)
  }

  function applyProgression() {

    var item = cProgression[cycleIndex]

    // Split into Root, and scale
    var pairOfItems = item.split(" ");

    var newRoot = pairOfItems[0]

    mixParameterSet("Mix Root", newRoot)
    var testRoot = mixParameterGet("Mix Root")
    wjsLog ("Mix Root now=" + testRoot)

    // Set scale as well; to a value related to the selected root.

    var scaleName = pairOfItems[1]
    applyScaleName(scaleName)

    applyRandomTempo()
  }

  function startProgression() {
    cycleIndex = 0

    applyProgression()
  }

  function nextProgression() {
    cycleIndex = cycleIndex + 1
```

```
    cycleIndex = cycleIndex + 1
    if (cycleIndex >= cProgression.length) {
      // We've reached the end - go back to the beginning...
      cycleIndex = 0
    }

    applyProgression()
  }

  // Check if the specified Scale Rule name exists in the mix
  function checkForChangeInProgression(bar) {

    // Every 4 bars (or whatever...), change the root and scale, following the progres
    // seen in Chopin's preludes...

    if ( ((bar + 1) % cChangeEveryBars) == 0) {

      nextProgression()
    }
  }

  // Event Triggers

  function onImeStart() {
    startProgression()
  }

  function onImeBar(bar) {

    wjsLog ("Mix: onImeBar(" + bar + ") - Cell")
    checkForChangeInProgression(bar)
  }
```

Hyperinstrument

What is a hyperinstrument?

A hyperinstrument can be defined as being an instrument that can have extraordinary, semi-automated response to relatively simple real-time inputs.

Wotja is a hyperinstrument in many ways. For example from simple interaction in real-time through mouse and keyboard, you can have it generate extraordinary, ever-changing music that responds to your inputs. Wotja also directly responds to any real-time MIDI note data fed into it, for example from a MIDI keyboard or other MIDI controller; where you feed MIDI data into Wotja, you will find that Wotja can automatically harmonise its generators with all incoming MIDI note data. You can also configure Wotja to respond in many advanced ways to both MIDI note events and MIDI control events (otherwise known as "*MIDI CCs*").

Wotja can function as a hyperinstrument for 2 main reasons: The IME can real-time harmonise with incoming

MIDI note events and control events (otherwise known as "*MIDI CCs*"), and Wotja's generative engine is an instrument that creates more than the sum of your direct inputs with keyboard, mouse, and MIDI. In other words, Wotja's generative engine, MIDI control, scripting and direct MIDI input harmonisation all work together to make itself greater than the sum of its parts; a **customisable generative hyperinstrument**!

See the Hyperinstrument - Quick Start if you want to get started quickly.

How does Wotja work as a hyperinstrument?

You can respond to any incoming MIDI CC however you want; for example, to change the current scale or adjust a harmony rule in real-time in response to incoming MIDI CC events.

You can also respond to any incoming MIDI Note on/off event however you want; for example, you could use some notes (such as accidentals in a scale) to change parameter values. In other words, you can use certain specific notes to trigger various complex responses by Wotja, for example; or to enable/disable the chording parameters only for certain notes (immediately that a note is triggered!); or to change scale; or pretty much whatever you can imagine.

Here is an example of a Generator MIDI In Note trigger script that you could use in your own mix.

```
function onMIDIInNote(channel, noteon, pitch, velocity) {
  if (channel == imeGeneratorMIDIChannelGet()) {
    wjsLog("onMIDIInNote for our channel!")
  }
}
```

Quick Start - Listening generators / Hyperinstrument

To have a Generator "listen" to incoming MIDI data, you need to follow the instructions below.

1. Set "MIDI Input" to be your MIDI input device.
2. Set the generator you want to listen to MIDI input, such that:
   - the Generator Type is Listen
   - the MIDI In Script is something like this:

   ```
   function onMIDIInNote(channel, noteon, pitch, velocity) {
     if (channel == imeGeneratorMIDIChannelGet()) {
       wjsLog ("onMIDIInNote for our channel!")
     }
   }
   ```
3. You might want to set your Listening generator to auto-chord with depth of 2 or greater.
4. If you want an different Generator to also make sounds in response to your incoming MIDI note event, set it to FOLLOW the Listening Generator.