

INScore Web

Version 1.27

D. Fober



GRAME

Centre national de création musicale

Contents

1	Introduction	1
2	Behavioural differences	1
2.1	Optional components	1
2.2	Using files in a script	1
3	Unsupported	2
3.1	Unsupported objects	2
3.2	Unsupported messages	2
3.2.1	Common messages	2
3.2.2	Application messages	2
3.2.3	Application log window	2
3.2.4	Scene messages	3
3.2.5	Type specific messages	3
3.2.6	Synchronization	3
3.2.7	Events	3
4	Specific new messages	3
4.1	Leveraging CSS	3
4.2	MIDI support	4
5	Faust objects	5
5.1	Creating a Faust object	5
5.2	Faust messages	5
5.3	Faust objects parameters	6
5.4	Polyphonic objects	6
6	Communication scheme	6

1 Introduction

Since version 1.27, the INScore engine is available as Javascript libraries:

- a WebAssembly [WASM] library providing all the services of the abstract INScore model,
- a Javascript library providing an HTML view of the INScore model.

The web environment provides a very different runtime context than a native application: it is much more modular; due to the absence of a 'concrete machine' a number of INScore primitives do not make sense in a web environment; finally it provides new rendering capabilities with CSS.

This document is intended to present the differences between the native and web versions of INScore. A special section is also devoted to the implementation of INScore Web in standalone HTML pages.

2 Behavioural differences

The OSC protocol is not supported in the Web version. As a result, the mode of communication with the INScore engine is different (see section 6) and may also depend on the application that uses this engine.

By default:

- the OSC output and error ports are redirected to the Javascript console.
- drag & drop works like in the native version: you can drop files or text to an INScore scene (an HTML div in the Web version).

The `log` window (address `/ITL/log`) is dependent on the host application. By default, input messages addressed to the `log` node are directed to the Javascript console.

2.1 Optional components

As the architecture of the web version is completely modular, the available objects depend on the host application: e.g. a page which wants to use objects in symbolic notation (type `gmn`) will have to include the Guido library. This architecture allows applications to be optimized to fit their needs. It also facilitates the extension of the INScore engine. The table 1 presents the current supported components.

Component	Name	Dependent types
Guido Engine	<code>libGUIDOEngine.js</code>	<code>gmn</code> , <code>gmnf</code> , <code>gmnstream</code> , <code>pianoroll</code> , <code>pianorollf</code> , <code>pianorollstream</code>
MusicXML library	<code>libmusicxml.js</code>	<code>musicxml</code> , <code>musicxmlf</code> use of MusicXML implies to have also the Guido Engine
Faust compiler	<code>libfaust-wasm.js</code> <code>FaustLibrary.js</code>	<code>faust</code> , <code>faustf</code>

Table 1: Components required by specific objects

2.2 Using files in a script

You can use file based objects in an INScore script but the file path is interpreted differently:

- when using an absolute path, it refers to the document root of the HTTP server,
- when using a relative path, it refers to the location of the HTML page

NOTE

Browsers infer a MIME type from the file extension and generally, download any file which extension is not recognized (this behavior depends on the browser you are using). It is therefore recommended to use a `.txt` extension for any textual resource with non-standard extension. For example, a `score.gmn` file could be renamed and used as `score.gmn.txt`.

3 Unsupported

3.1 Unsupported objects

The table 2 presents the objects that are not supported:

Type	Comment
fileWatcher	doesn't make sense in a web environment
httpd	unsupported in a web environment
websocket	unsupported in a web environment
Faust plugins	deprecated and redesigned (see section 5)
Gesture follower	unsupported

Table 2: Unsupported objects

The following components are not yet implemented:

- graphic signals objects: graph, fastgraph, radialgraph
- Pianoroll stream: pianorollstream
- Misc.: grid
- Memory image: memimg
- Sensors: acceleromter, gyroscope, compass, etc.

3.2 Unsupported messages

A number of messages are not supported in the web version, either because they do not make sense in the runtime context or because they cannot be implemented.

3.2.1 Common messages

- export, exportAll, save: are not supported due to the lack of file system
- shear, dshear: not implemented
- effect: colorize: not implemented
- edit: not yet implemented

3.2.2 Application messages

- quit: do not make sense in the web environment
- rootPath: do not make sense in the web environment
- mouse: not yet implemented
- read: not implemented
- load: not implemented
- port, outport, errport: do not make sense without OSC
- browse: not implemented
- get: guido-version musicxml-version: not implemented.
Note that the corresponding components are loaded, they print their version number to the Javascript console.

3.2.3 Application log window

As mentioned above, the log window (address /ITL/log) is dependent on the host application. By default, input messages addressed to the log node are directed to the Javascript console.

- clear: dependent on the host application (do nothing by default)

- `save`: not supported due to the lack of file system
- `foreground`: dependent on the host application (do nothing by default)
- `wrap`: dependent on the host application (do nothing by default)
- `scale`: dependent on the host application (do nothing by default)
- `write`: dependent on the host application (write to the Javascript console by default)

3.2.4 Scene messages

- `save`: not supported due to the lack of file system
- `foreground`: dependent on the host application (do nothing by default)
- `rootPath`: not yet implemented
- `load`: not yet supported
- `frameless windowOpacity`: not supported

3.2.5 Type specific messages

- `brushstyle`: not yet implemented
- Symbolic score:
 - `columns rows`: not supported
 - `pageFormat`: not yet implemented
 - `getPageCount systemCount`: not yet implemented
- Piano roll messages: not yet implemented
- Video get messages: not yet implemented
- SVG animate: not yet implemented
- Arc close: not yet implemented
- Line arrows: not yet implemented
- debug node: not yet implemented

3.2.6 Synchronization

The following synchronisation modes are not yet supported:

- Stretch modes: `h v hv`

3.2.7 Events

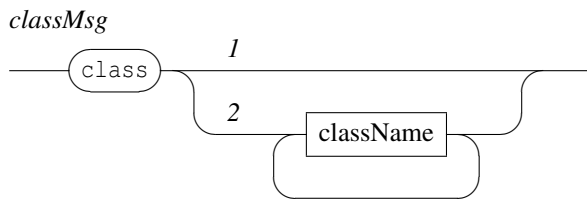
The following events are not yet supported:

- Touch events: `touchBegin touchEnd touchUpdate`
- Url events: `success error cancel`
- Video events: `ready end`
- Score event: `pageCount`
- `export`: not supported since the `export` message is not supported
- `endPaint`: not supported

4 Specific new messages

4.1 Leveraging CSS

Cascading Style Sheets [CSS] are a powerful way to control the appearance of elements on a web page. The Web version of INScore provides a specific `class` message to use CSS in parallel to the standard mechanisms. This message is supported by all the score components.

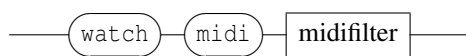


- 1: without argument, remove all class settings
- 2: set the CSS classes of an object

4.2 MIDI support

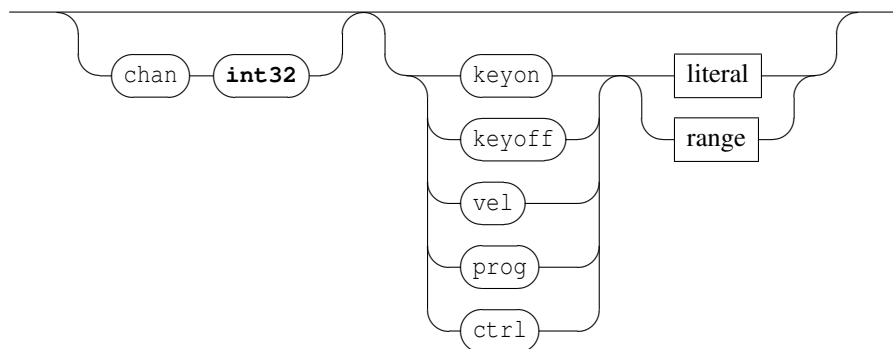
MIDI is supported using a specific event that you can configure using the `watch` message.

watchMIDI



A filter is used to select the MIDI messages that will trigger the event.

midifilter



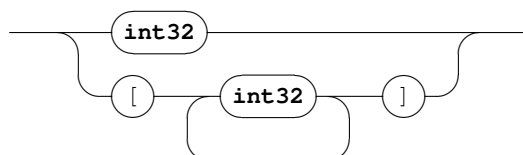
- with an empty filter, any MIDI message will trigger the event
- `chan`: an optional channel number may be used to select only the MIDI messages on a specific MIDI channel
- `keyon`, `keyoff`, `vel`, `prog`, `ctrl`: are used to select MIDI messages according to their status.

EXAMPLE

Filtering MIDI messages that are not on channel 0

```
/ITL/scene/obj watch midi chan 0 (inscore messages list);
```

literal

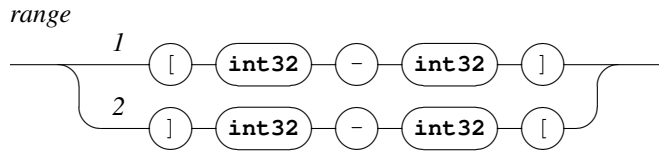


Literal values are either a specific value or a list of values enclosed in brackets.

EXAMPLE

Accepting MIDI key on messages for 3 specific pitches.

```
/ITL/scene/obj watch midi keyon [60 62 64] (inscore messages list);
```



Range may be used when a value enter or leave the specified range.

- 1: trigger the event when the value enters the range
- 2: trigger the event when the value leaves the range

5 Faust objects

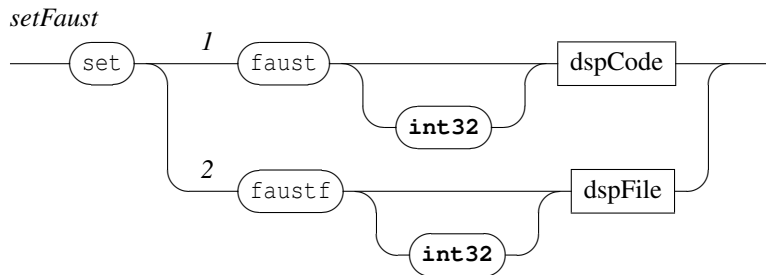
Faust is a functional programming language for sound synthesis and audio processing. Faust objects are available, providing that the Faust library has been loaded.

5.1 Creating a Faust object

The `faust` or `faustf` types must be used to create a Faust object.

NOTE

The `faust` type exists with the native version but to load a pre-compiled DSP. `faustdsp` and `faustdspf` types are not supported.



The expected arguments of the `set` message are:

- an optional integer that indicates a number of voices used to create a polyphonic DSP (see section 5.4). Note that when present, a polyphonic DSP is created even if equal to 1.
- 1: Faust DSP code (see the **Faust language** for more information).
- 2: a Faust DSP file. (not yet implemented)

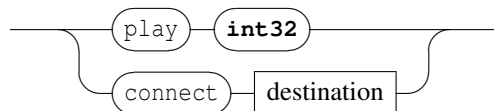
By default, a Faust DSP appears as a browsable block diagram.

NOTE

The Faust language uses characters that have a special meaning in HTML: for example, with the split operator `<:`, the `<` character will be interpreted as an opening HTML tag and you should use HTML escapes (e.g. `<`; instead of `<`). This is necessary for inline DSP code only, DSP files are not concerned.

5.2 Faust messages

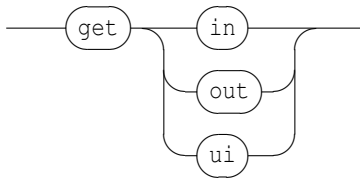
faustMsgs



- play: expected parameter is '1' to connect the Faust processor to the audio output, '0' to disconnect from the audio output. (maybe connect is enough)

- connect: gives the number of outputs of the Faust processor (not yet implemented)

faustgetMsgs



- in: gives the number of inputs of the Faust processor
- out: gives the number of outputs of the Faust processor
- ui: gives the Faust processor parameters (see section 5.3) (not yet implemented)

5.3 Faust objects parameters

A Faust DSP code can declare UI elements that are used by *architecture files* to build controllers providing users with dynamic control of the DSP parameters. In INScore, DSP UI elements are used to extend the Faust object address space. For example, when a DSP code declares a UI element named 'Volume', a Faust object which address is /ITL/scene/dsp is extended as /ITL/scene/dsp/Volume.

Faust parameters support two types of messages:

faustParamMsgs

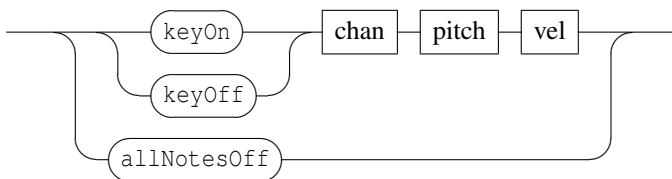


- 1: set the parameter value
- 2: gives the parameter value

5.4 Polyphonic objects

Polyphonic objects (i.e. Faust objects created using the optional voice number) support additional messages:

faustPolyMsgs



- keyOn and keyOff messages take 3 integer parameters: a MIDI channel, the note pitch and the velocity.
- allNotesOff: similar to MIDI all notes off message

6 Communication scheme