INScore
OSC Messages Reference
v.0.63

Grame, Centre national de création musicale
<research@grame.fr>
ANR-08-CORD-010

January 14, 2011

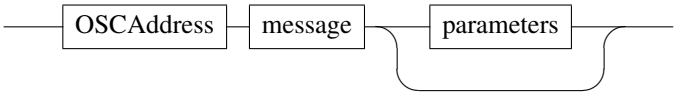
Contents

Chapter 1

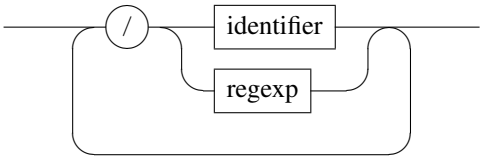
General format

An OSC message is made of an OSC address, followed by a message string, followed by zero to n parameters. The message string could be viewed as the method name of the object identified by the OSC address. The OSC address could be string or a regular expression matching several objects.

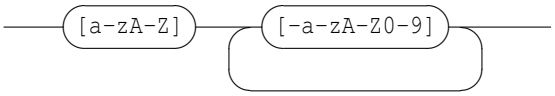
OSCMessage



OSCAddress



identifier

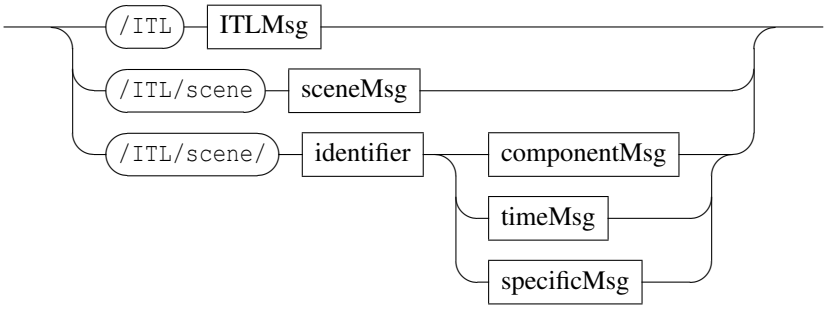


The OSC types *int32*, *float32* and *OSC-string* are used as terminal symbols and are denoted by **`int32`**, **`float32`** and **`string`**.

1.1 Address space

Here is a description of the OSC address space, including static and dynamic nodes and the corresponding messages.

addressSpace

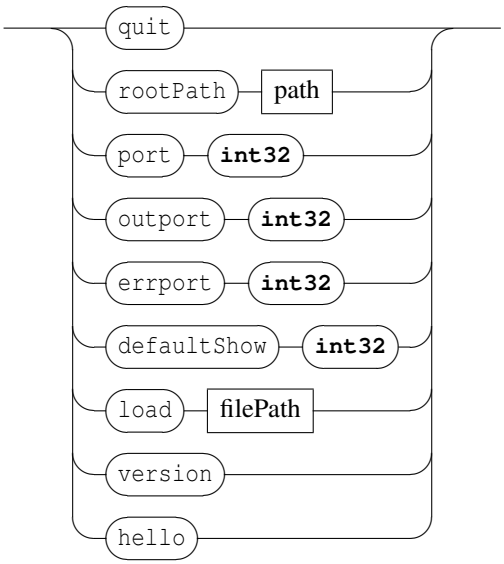


Chapter 2

ITL messages

ITL messages are messages global to the client application, which address is `/ITL`. It handles the following messages:

ITLMsg



- `quit`: requests the client application to quit.
- `rootPath`: *rootPath* of an Interlude application is the default path where the application reads or writes a file when a relative path is used for this file. The default value is the user home directory. Sending the `rootPath` message without parameter resets the application path to its default value.
- `port`, `outport`, `errport`: changes the UDP port numbers. `port` defines the listening port number, `outport` and `errport` define the ports used to send messages and error messages. The `int32` parameter should be a positive value in the range `[1024-49150]`.
  The default `port`, `outport` and `errport` values are 7000, 7001 and 7002.
- `defaultShow`: changes the default `show` status for new objects.
  The default `defaultShow` value is 1.
- `load`: loads a file previously saved using the `save` message (see section 4 p.6). Note that the load operation appends the new objects to the existing scene. Whe necessary, it's the sender responsability to clear the scene before loading a file.
- `hello`: query the host IP number. The message is intended for ITL applications discovery. Answer

to the query has the following format:

`IP inPort outPort errPort` where `IP` is sent as a string and port numbers as integer values.

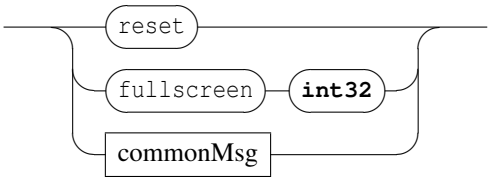
- `version`: version number request.

Chapter 3

Scene messages

A scene may be viewed as a window on the score elements. Its address is `/ITL/scene`. It handles the following messages:

sceneMsg



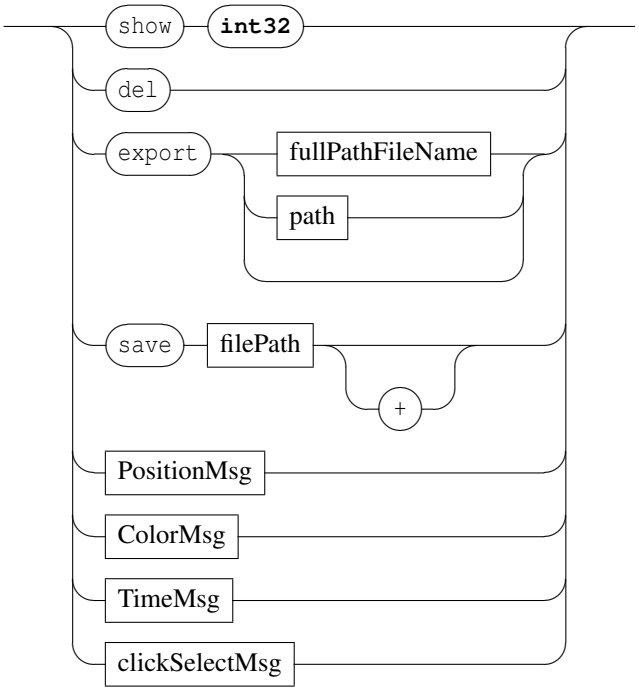
- `reset`: clears the scene (i.e. delete all components) and resets the scene to its default state (position, size and color).
- `fullscreen`: requests the scene to switch to full screen or normal screen. The parameter is interpreted as a boolean value. Default value is `0`.
- `commonMsg`: see section 4 p.6.

Chapter 4

Common components messages

Common messages are mainly intended to control the graphic appearance of the components and of the scene. They could be sent to any address with the form `/ITL/scene` or `/ITL/scene/`*`identifier`* where *`identifier`* is the unique identifier string of a scene component.

commonMsg

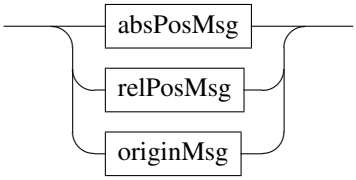


- `show`: shows or hides the destination object. The parameter is interpreted as a boolean value. Default value is `1`.
- `del`: deletes the destination object.
- `export`: exports an object to an image file.
  The first form exports to the parameter file, which specifies a full path name. The file extension is used to infer the export format. Supported extensions and formats are: *pdf, bmp, gif, jpeg, png, pgm, ppm, tiff, xbm, xpm*.
  The second form exports to `path/`*`identifier`*`.pdf`. When path is a relative path, it exports to `rootPath/path/`*`identifier`*`.pdf`. A scene *identifier* is '`scene`'.
  The third form exports to `rootPath/`*`identifier`*`.pdf`.

- `save`: recursively saves objects states to a file. The `filePath` can be relative or absolute. When relative, an absolute path is build using the current `rootPath`. The optional `+` parameter indicates an append mode for the write operation. The message must be sent to the `/ITL` address to save the whole application state.
- 'PositionMsg' are absolute and relative position messages.
- 'ColorMsg' are absolute and relative color control messages.
- 'TimeMsg' are time management messages. They are described in chaper 5 p.14.
- 'clickSelectMsg' are provided to query objects relative positions.

4.1 Positioning

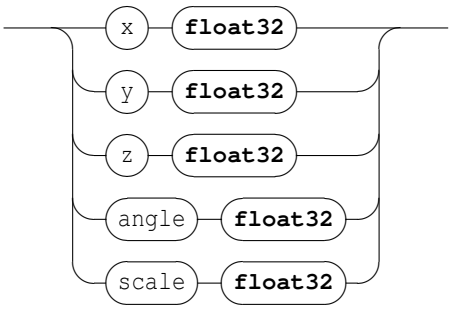
PositionMsg



Graphic position messages are absolute position messages or relative position messages.

4.1.1 Absolute positioning

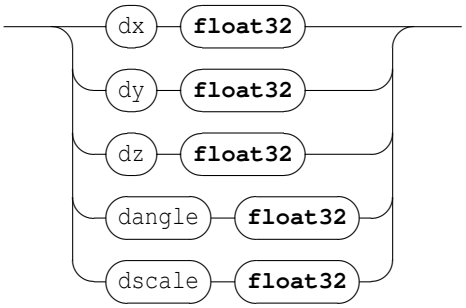
absPosMsg



- `x y`: moves the `x` or `y` coordinate of a component. By default, components are centered on their `x`, `y` coordinates. The coordinates space range is `[-1,1]`.
  For a `scene` component, -1 is the leftmost or topmost position, 1 is the rightmost or bottommost position. `[0,0]` represents the center of the `scene`.
  For the `scene` itself, it moves the window in the screen space and the coordinate space is orthonormal, based on the screen lowest dimension (*i.e.* with a 4:3 screen, `y=-1` and `y=1` are respectively the exact top and bottom of the screen, but neither `x=-1` nor `x=1`are the exact left and right of the screen).
  Default coordinates are `[0,0]`.
- `z`: sets the `z` order of a component. The range is $[0,\infty[$. `z` order is actually relative to the `scene` components: objects of high `z` order will be drawn on top of components with a lower `z` order. Components sharing the same `z` order will be drawn in an undefined order, although the order will stay the same for as long as they live.
  Default `z` order is 0.
- `angle`: sets the `angle` value of a component, which is used to rotate it around its center. The angle is measured in clockwise degrees from the `x` axis.
  Default angle value is 0.
- `scale`: reduce/enlarge a component. The range is $[0,\infty[$. Default scale is `1`.

4.1.2 Relative positioning

relPosMsg

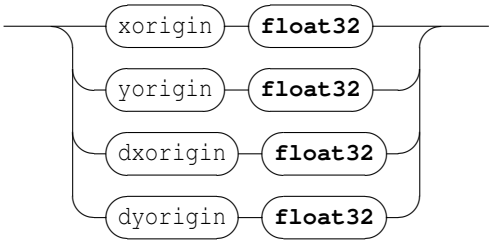


- `dx`,`dy`, etc. messages are similar to `x`,`y`, etc. messages but the parameters represents values represent a displacement of the current target value.
- `dscale` is similar to `scale` but the parameters represents a scale multiplying factor.

4.1.3 Components origin

The origin of a component is the point (x_o, y_o) such that the (x, y) coordinates and the (x_o, y_o) point coincide graphically. For example, when the origin is the top left corner, the component top left corner is drawn at the (x, y) coordinates.

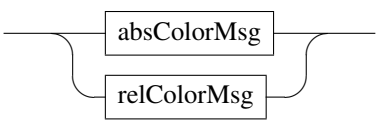
originMsg



- `xorigin`,`yorigin` are relative to the component coordinates space i.e. $[-1, 1]$, where -1 is the top or left border and 1 is the bottom or right border. The default origin is $[0, 0]$ i.e. the component is centered on its (x, y) coordinates.
- `dxorigin`,`dyorigin` represents displacement of the current `xorigin` or `yorigin`.

4.2 Color messages

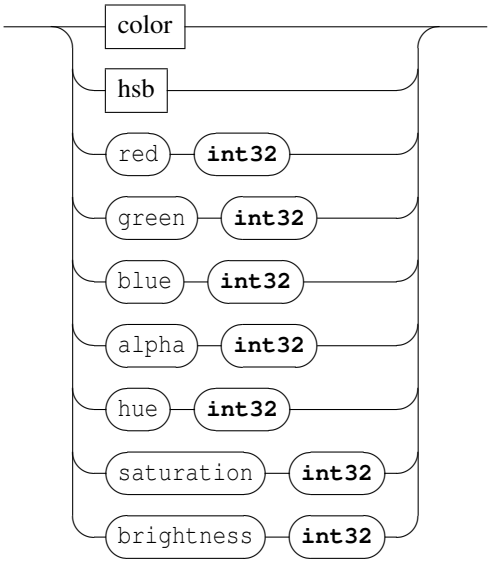
ColorMsg



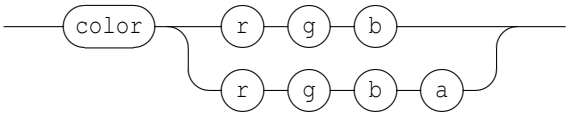
Color messages are absolute or relative color control messages.

4.2.1 Absolute color messages

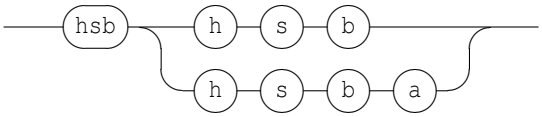
absColorMsg



color



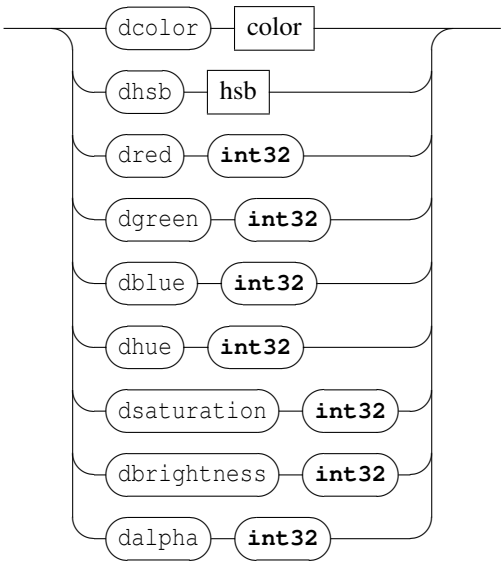
hsb



- `color`: sets an object color. The color scheme is RGBA. When A is not specified, the color is assumed to be opaque. The data range for each color component is `[0,255]`. Default color value is `[0,0,0,255]`.
- `hsb`: similar to `color` but using an HSBA scheme. When A is not specified, the color is assumed to be opaque. The data range for S, B is `[0,100]`, for H `[0,360]`, and `[0,255]` for A.
- `red`,`green`,`blue`, `hue`,`saturation`,`brightness`,`alpha` messages address only the specified part of the color.

4.2.2 Relative color messages

relColorMsg

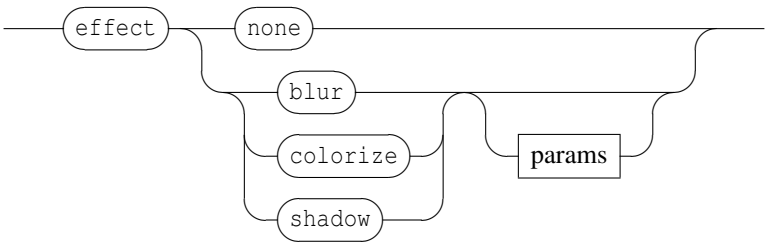


- `dred`,`dgreen`, etc. messages are similar to `red`,`green`, etc. messages but the parameters values represent a displacement of the current target value.
- `dcolor` and `dhsb` are similar and each color parameter represents a displacement of the corresponding target value.

4.3 The 'effect' messages

The `effect` message sets a graphic effect on the target object. These messages are experimental and could be changed or removed in a future version.

effectMsg

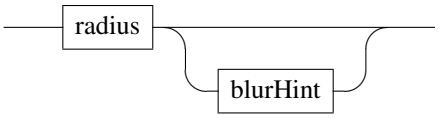


- `none`: removes any effect set on the target object.
- `blur, colorize, shadow`: sets the corresponding effect. An effect always replaces any previous effect. The effect name is followed by optional specific effects parameters.

Note that an effect affects the target object but also all the target slaves.

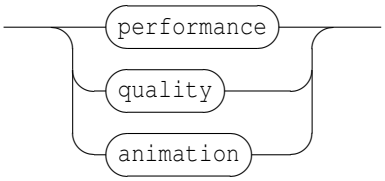
4.3.1 The blur effect

blurParams



Blur parameters are the blur radius and a rendering hint. The radius is an int32 value. By default, it is 5 pixels. The radius is given in device coordinates, meaning it is unaffected by scale.

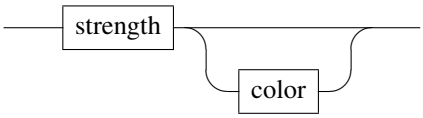
blurHint



Use the `performance` hint to say that you want a faster blur, the `quality` hint to say that you prefer a higher quality blur, or the `animation` when you want to animate the blur radius. The default hint value is `performance`.

4.3.2 The colorize effect

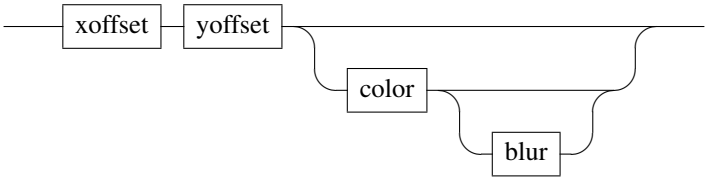
colorizeParams



Colorize parameters are a strength and a tint color. The strength is a float value. By default, it is 1.0. A strength 0.0 equals to no effect, while 1.0 means full colorization.
The color is given as a RGB triplet (see 4.2 p.8) by default, the color value is light blue (0, 0, 192).

4.3.3 The shadow effect

shadowParams



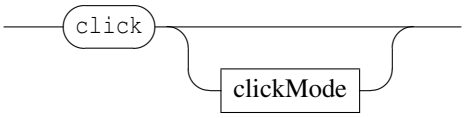
`xoffset` and `yoffset` are the shadow offset and should be given as int32 values. The default value is 8 pixels. The offset is given in device coordinates, which means it is unaffected by scale.
The color is given as a RGBA color (see 4.2 p.8) by default, the color value is a semi-transparent dark gray (63, 63, 63, 180)
The blur radius should be given as an int32 value. By default, the blur radius is 1 pixel.

4.4 The 'click' and 'select' messages

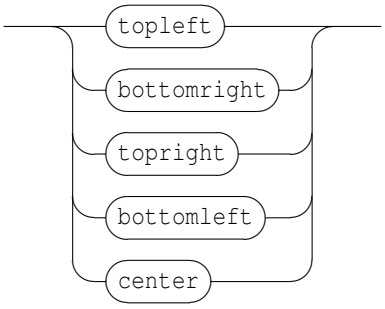
The `click` and `select` messages can be sent to any address with the form `/ITL/scene/`*`identifier`* where *`identifier`* is the unique identifier string of a scene component. They are intended to provide

information about components relative positions: it returns a list of components identifiers that are 'under' the *clicked* component: z order of the components is used for the 'under' relationship.

click



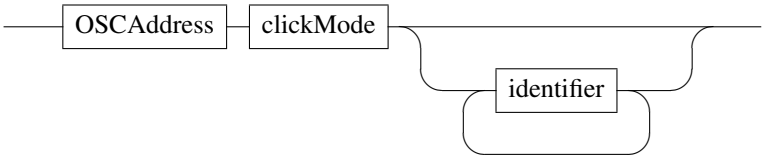
clickMode



- `topleft`, `bottomright`, `topright`, `bottomleft`, `center`: collects the components that are respectively under the top-left, bottom-right, top-right, bottom-left or center point of the target component. The clicked point is computed from the component bounding box.
- Sent without parameter, the `click` message is equivalent to `click topleft`.

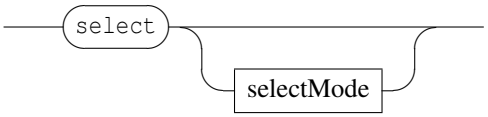
Reply of the `click` message has the following form:

clickReply

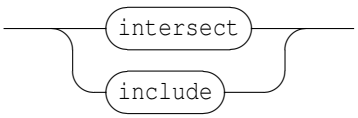


where 'identifier' is a component name. Note that 'identifiers' are sorted by decreasing z order.

select



selectMode

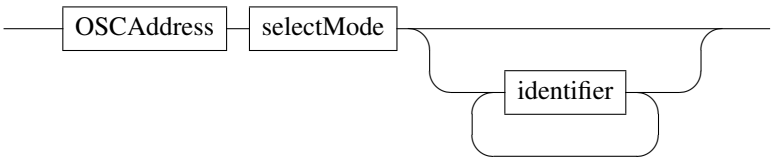


- `intersect`: looks for the components that are under the *clicked* component and that intersect with its bounding box.

- `include`: looks for the components that are under the *clicked* component and that are included in its bounding box.
- Sent without parameter, the `select` message is equivalent to `select intersect`.

Reply of the `select` message is similar to `click` reply:

selectReply

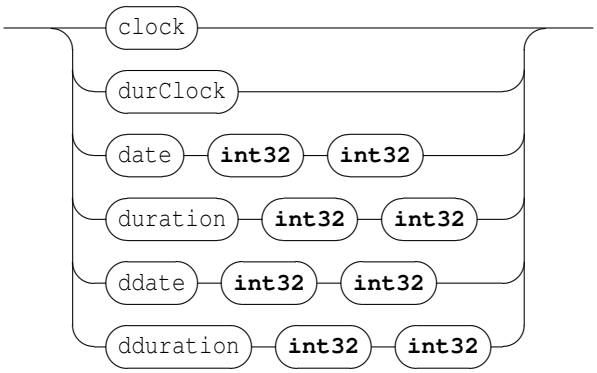


Chapter 5

Time management messages

Time messages control the time dimension if the score components. They could be sent to any address with the form `/ITL/scene/`*`identifier`* where *`identifier`* is the unique identifier string of a scene component.

timeMsg



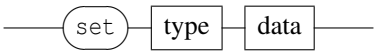
- `clock`: a clock message similar to MIDI clock message: advances the object date by 1/24 of quarter note.
- `durClock`: a clock message applied to duration: increases the object duration by 1/24 of quarter note.
- `date`: sets the time position of an object. Time is expressed as a rational value `d/n` where `n` represents the whole note division and `d` the divisions count. The parameters order is `numerator` followed by `denominator`.
  Default value is `0/1`.
- `duration`: changes the object duration. Duration is expressed as a rational value.
  Default value is `0/1`.
- `ddate`: relative time positioning message: adds the specified value to the object date.
- `ddate`: relative duration message: adds the specified value to the object duration.

Chapter 6

The ’set’ message

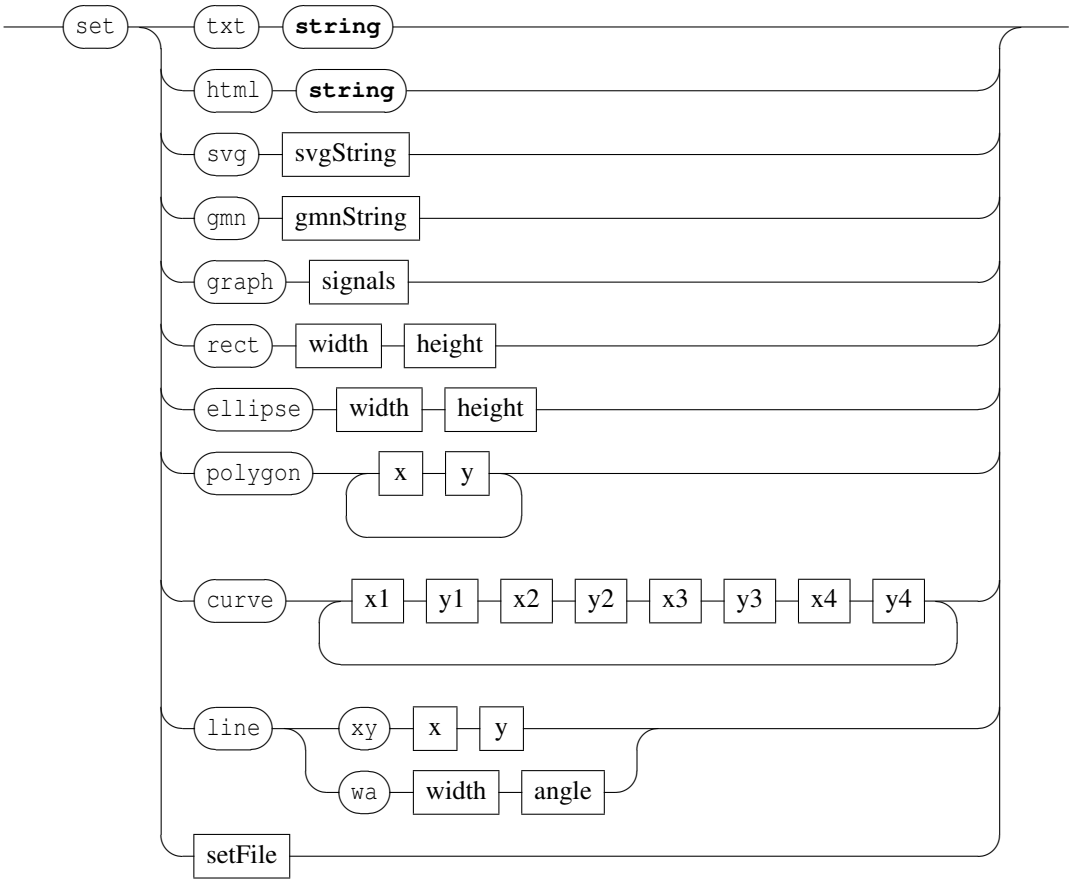
The `set` messages can be sent to any address with the form `/ITL/scene/`*`identifier`*. The global form of the message is:

setMsg



It sets a `scene` component data. When there is no destination for the OSC address, the component is first created before being given the message. When the target destination type doesn’t correspond to the message `type`, the object is replaced by an adequate object.
Format of the `set` message is:

setMsg



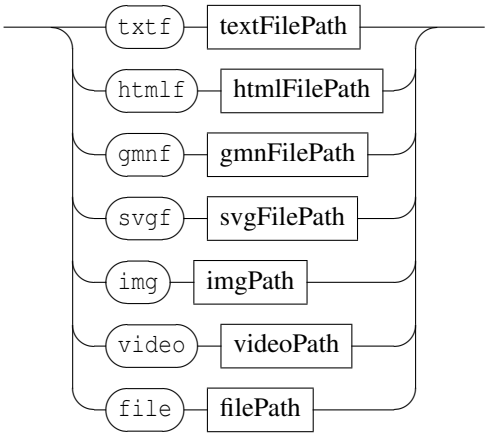
Supported types are the following:

- `txt`: a textual component.
- `html`: an html component defined by an HTML string.
- `gmn`: a Guido score defined by a GMN string.
- `svg`: SVG graphics defined by a SVG string.
- `graph`: graphic of a signal. See section 13 p.28 for details about the `graph` objects data.
- `rect`: a rectangle specified by a width and height. Width and height are expressed in `scene` coordinates space, thus a width or a height of 2 corresponds to the width or a height of the `scene`.
- `ellipse`: an ellipse specified by a width and height.
- `polygon`: a polygon specified by a sequence of points, each point being defined by its (x,y) coordinates. Those coordinates are expressed in the `scene` coordinate space, but only taking in acount the relative position of the points (*i.e* a polygon A = { (0,0) ; (1,1) ; (0,1) } is equivalent to a polygon B = { (1,1) ; (2,2) ; (1,2) }).
- `curve`: a sequence of 4-points bezier cubic curve. If the end-point of a curve doesn't match the start-point of the following one, the curves are linked by a straight line. The first curve follows the last curve. The inner space defined by the sequence of curves is filled, using the object color. The points coordinates are handled as in a `polygon`.
- `line`: a simple line specified by a point (x,y) expressed in `scene` coordinate space or by a width and angle. The point form is used to compute a line from (0,0) to (x,y), which is next drawn centered on the scene.
- `txtf`: a textual component defined by a file.
- `htmlf`: an html component defined by an HTML file.

- `gmnf`: a Guido score defined by a GMN file.
- `svgf`: vectorial graphics defined by a SVG file.
- `img`: an image file based component. The image format is infered from the file extension.
- `video`: a video file based component. The video format is infered from the file extension. Note that navigation through the video is made using its `date`.
- `file`: a generic type to handle file based objects. Actually, the `file` type is translated into a one of the `txtf`, `gmnf`, `img` or `video` types, according to the file extension (see table 6.1).

Note that the default position of any component is `[0,0]`. Objects are drawn centered on their position.

setFile



See also: the application `rootPath` message (section 2) for file based objects.

Table 6.1: File extensions supported by the `file` translation scheme.

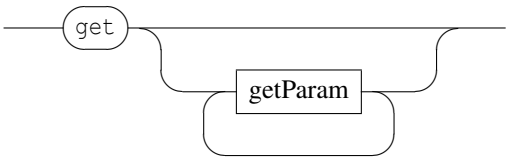
file extension	translated type
.txt .text	txtf
.htm .html	htmlf
.gmn .xml	gmnf
.svg	svgf
.jpg .jpeg .png .gif .bmp .tiff	img
.avi .wmv .mpg .mpeg .mp4	video

Chapter 7

The 'get' messages

The `get` messages can be sent to any valid OSC address. It is intended to query the system state. It is the counterpart of all the messages modifying this state. The result of the query is sent to the OSC output port with the exact syntax of the counterpart message. The global form of the message is:

getMsg



The `get` message without parameter is the counterpart of the `set` message.

7.1 Special 'get' forms

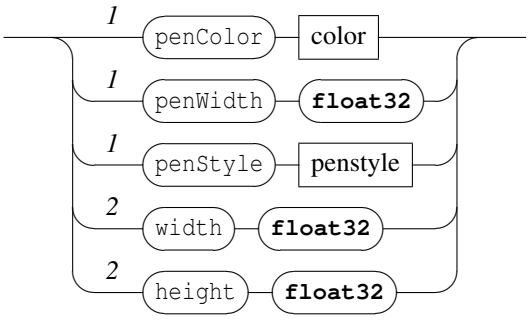
The `get` message without parameter addressed to a container (the application `/ITL`, a scene `/ITL/scene`, the signal node `/ITL/scene/signal`) is actually relayed to each of the container component.
The `get width` and `get height` messages adressed to components that have no explicit width and height (text, images, etc.) return 0 as long as the target component has not been drew.

Chapter 8

Component specific messages

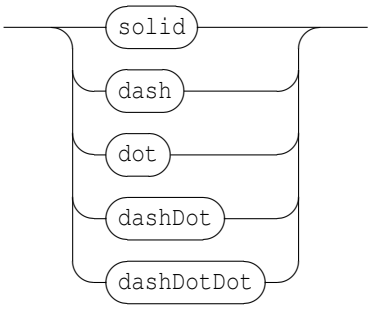
Some of the messages are specific to the component `type`. These messages can be sent to addresses of the form `/ITL/scene/`*`identifier`*. They are ignored by objects that don't support the message.

specificMsg



- **[1]** messages supported by the components types `rect | ellipse | polygon | curve | line | graph`.
- **[2]** messages supported by the components types `rect | ellipse | graph`.

penstyle



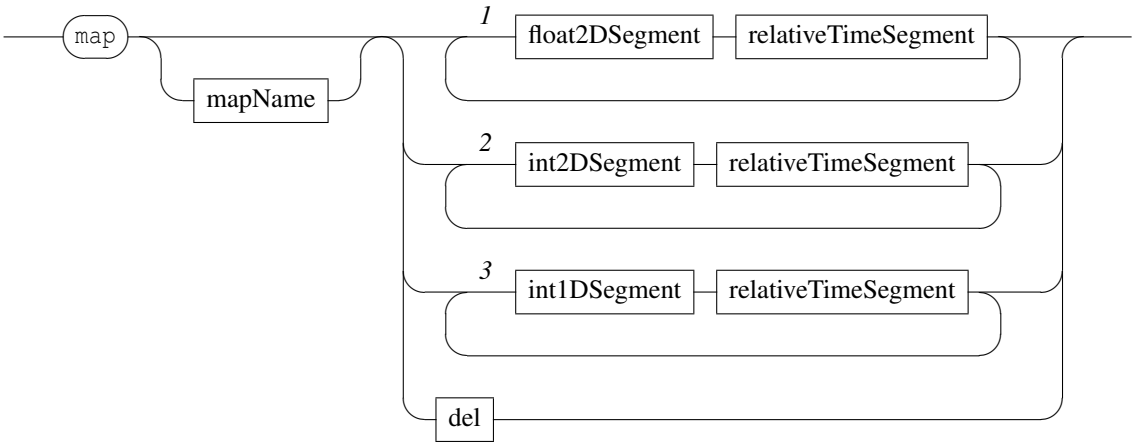
The color scheme is described in section 4.
The pen width default value is 0 (excepted for `line` objects, where 1.0 is the default value). It is expressed in arbitrary units (1 is a reasonable value).
The pen style default value is `solid`.

Chapter 9

The 'map' message

The `map` messages can be sent to any address with the form `/ITL/scene/`*`identifier`*. It is intended to describe the target object relation to time and sets a relation between an object segmentation and a time segmentation. The global form of the message is:

mapMsg



Segments are expressed as a list of intervals. For a 1 dimension resource, a segment is a made of a single interval. For a 2 dimensions resource, a segment is a made of 2 intervals: an interval on the *x*-axis and one on the *y*-axis for graphic based resource, or an interval on columns and one on lines for text based resources. Intervals are right-opened.
The different kind of relations corresponds to:

- **[1]** a relation between a 2 dimensions segmentation expressed in float values and a relative time segmentation. These segmentations are used by `rect, ellipse, polygon, curve, line` components.
- **[2]** a relation between a 2 dimensions segmentation expressed in integer values and a relative time segmentation. These segmentations are used by `txt, txtf, img` components.
- **[3]** a relation between a 1 dimension segmentation expressed in integer values and a relative time segmentation. These segmentations are used by the `graph` component and express a relation between a signal space and time.

Table 9.1 summarizes the specific segmentations used by each component type.
The specified `map` can be named with an optional `mapName` string; this name can be further reused, during object synchronization, to specify the mapping to use. When `mapName` is not specified, the mapping has a default *empty name*.

The `del` command deletes the mapping specified with `mapName`, or the *'empty name'* mapping if no map name is specified.

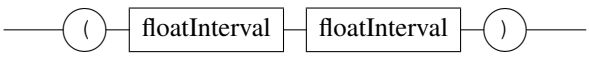
Table 9.1: Mapping resources for each component

component type	segmentation type
`txt, txtf`	int2DSegments
`img`	int2DSegments
`rect, ellipse, polygon, curve`	float2DSegments
`graph`	int1DSegments

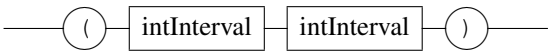
Note for `html, htmlf` + guido score mappings

relativeTimeSegment

float2DSegment

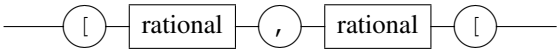


int2DSegment

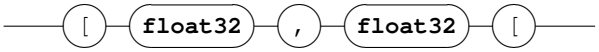


int1DSegment

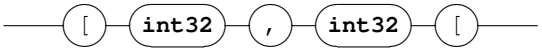
relativeTimeInterval



floatInterval

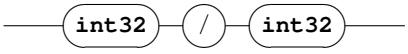


intInterval



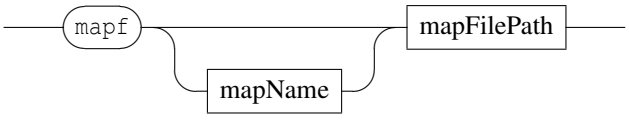
Relative time is expressed as `rational` values where 1 represents a whole note.

rational



The `mapf` messages is similar to the `map` message but gives the path name of a file containing the mapping data, along with the optional map name.

mapfMsg

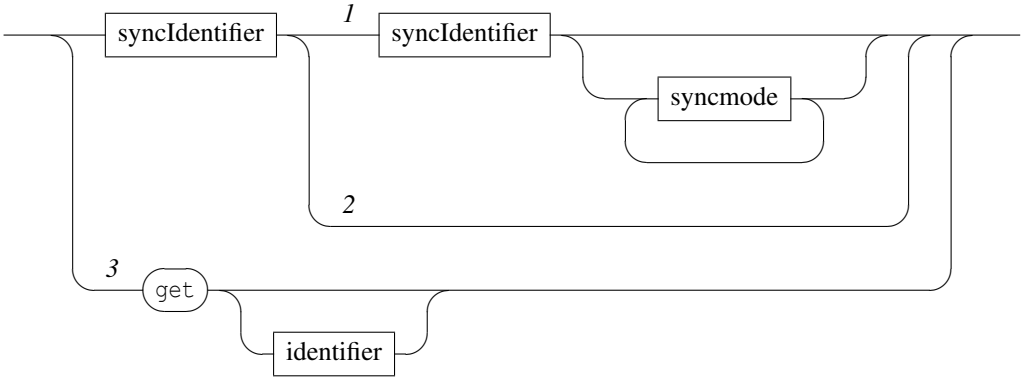


Chapter 10

Synchronization

Synchronization between components is achieved at `scene` level: a `sync` node is automatically embedded in the scene, its address is `/ITL/scene/sync` and it supports messages to add or remove a master / slave relation between components or to query the synchronizations state.

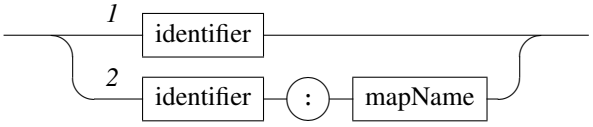
sync



- **[1]** this is the `slave master` form followed by an optional synchronization mode. It adds a slave / master relation between components identified by identifier1 and identifier2.
- **[2]** this is the `slave` form that removes the slave synchronization for the component identified by identifier1.
- **[3]** the `get` message is intended to query the synchronization state. The optional parameter is the identifier of a component,

Synchronization between components has no effect if any of the required mapping is missing (see table 9.1).

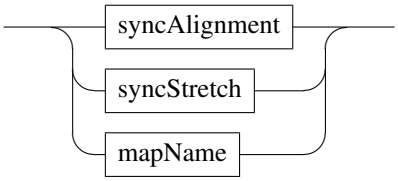
syncIdentifier



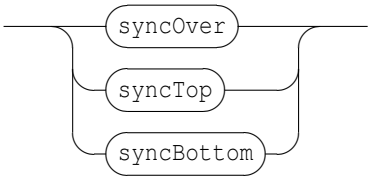
Synchronization identifiers indicates 1) the name of a `scene` component or 2) the name of a `scene` component associated to a mapping name.
Note that the `sync` node doesn't responds to common component messages, but accept the `get` message. The `get` message without parameter is equivalent to a `get` message addressed to each object declared in the `sync` node.

10.1 Synchronization modes

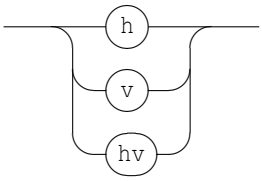
syncmode



syncAlignment



syncStretch



Synchronizing a component `A` to a component `B` has the following effect:

- `A` position (x,y) is modified to match the `B` time position corresponding to `A` date.
- depending on the optional `syncStretch`, `A` width and/or height are modified to match the corresponding `B` dimensions (see below).
- if `A` date has no graphic correspondance in `B` mapping (the date is not mapped, or out of `B` mapping bounds), `A` won't be visible.

The synchronization alignment modes have the following effects on the slave `y` position:

- `syncOver`: the center of the slave is aligned to the master center.
- `syncTop`: the bottom of the slave is aligned to the top of the master.
- `syncBottom`: the top of the slave is aligned to the bottom of the master.

The default synchronization mode is `syncOver`.
The synchronization stretch modes have the following effects on the slave dimensions:

- `h`: the slave is horizontally stretched to align its begin and end dates to the corresponding master locations.
- `v`: the slave is vertically stretched to the master vertical dimension.
- `hv`: combines the above parameters.

By default, no stretching is applied.
The optional 'mapName' string argument specifies which mapping of the master object should be used (see `map` message). If the master doesn't have a mapping with the specified 'mapName', the slave object won't be visible. Not specifying the map name is equivalent to an *empty name* mapping.

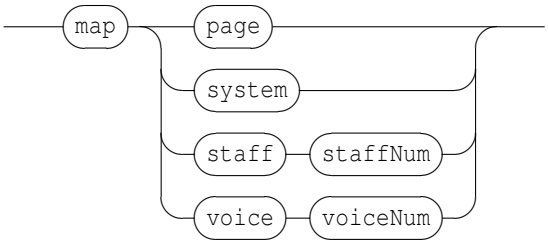
Chapter 11

Score specific messages

11.1 Specific score mappings

Music score mappings are automatically computed by the system. However, since a score may support a large number of mappings (staff based, system based, voice based, etc.), music scores supports a specific `map` message to request the computation of a specific mapping. This message has the form:

scoreMapMsg



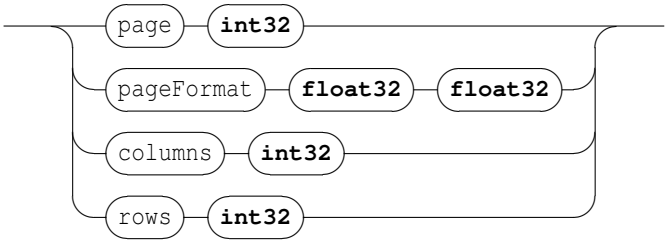
The Guido map name must be concatenated as a string (as a usual map name).

- `page`: a page level mapping ;
- `system`: a system level mapping ;
- `staff`: a staff level mapping ; the staff indicated by the next parameter (between 1 and the score staves count).
- `voice`: a voice level mapping ; the voice indicated by the next parameter (between 1 and the score voices count).

The default synchronization mode for a Guido score is `staff1`. Note that a voice may be distributed on several staves and thus a staff may contain serveral voices.

11.2 Score browsing and layout

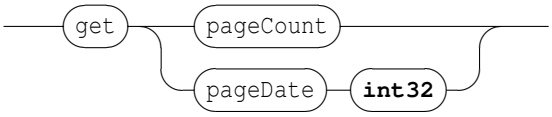
scoreMsg



- `page`: sets the first displayed page. Index starts at `1`. Default: `1`.
- `pageFormat`: sets the score's page format, in cm (Warning: the `pageFormat` is not used when another page format is 'hard-coded' in the score's Guido code). Default: `(21.0 , 29.7)`.
- `columns`: as multiples pages can be displayed and organized in a *grid of pages*, `columns` sets the number of columns in the grid. Default: `2`.
- `rows`: as multiples pages can be displayed and organized in a *grid of pages*, `rows` sets the number of rows in the grid. Default: `1`.

11.3 Score queries

scoreQuery



- `pageCount`: gives the score page count.
- `pageDate`: gives the time location of a page, which number is given as argument. The system reply has the form `pageDate pageNum date` where `pageNum` is the request page number given as argument and `date` is the page time location expressed as a ra tional number.

Note that a score object duration is automatically set to the score duration. It can be retrieved using a `get duration` message.

Chapter 12

Virtual address space

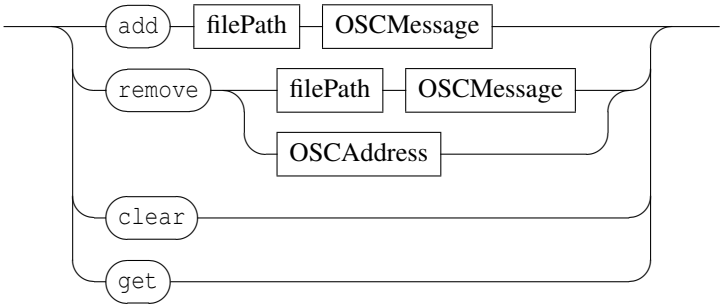
Virtual address space extends the OSC address scheme to objects automatically embedded into other objects.

12.1 File watcher

The `fileWatcher` is a component automatically embedded at scene level.
It receives messages at the address `/ITL/scene/fileWatcher`.
The `fileWatcher` service monitors files on demand; it is aware of file modifications and file removals (or renamings, which is equivalent). The `fileWatcher` works with associations between a file and an OSC message : when a file change occurs, the file watcher posts the associated OSC message.
The `fileWatcher` is controlled with 4 messages:

- the `add` message is used to add a new 'file' - 'OSC message' association ;
- the `remove` message is used to remove a 'file' - 'OSC message' association.
- the `clear` message is used to clear all existing associations.
- the `get` message is used to retrieve the current associations.

fileWatcher



where `OSCMessage` is any message complying to the format desscribed in section 1 p.1.
Example:

```
/ITL/scene/fileWatcher add fuga.gmn /ITL/scene/myScore set gmnf fuga.gmn
```

Note that the `get` query returns a set of messages complying to the `add` message format.

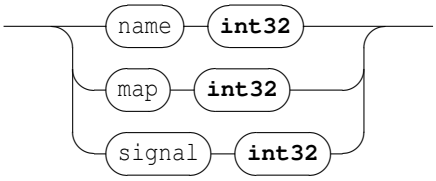
The `remove` message may take only 1 parameter, in which case the only parameter is an OSC address : it stops monitoring any file associated to an OSC message with this OSC address.

Note that an OSC message can be associated to several files, and that a file can be associated to several OSC messages. When a file has been removed (or renamed), the `fileWatcher` stops monitoring it.

12.2 Debug

Each object has a `debug` sub-node for debugging purposes. This `debug` virtual node has 3 flags, that can be actived or desactived with `0` or `1`:

debug



- When the `name` flag is on, each scene component displays its bounding rectangle and name.
- When the `map` flag is on, each scene component displays its mappings.
- When the `signal` flag is on, each object (even the scene or the application) will, according to its type, emit 'performance related' signals (or no signal). This 'performance signal' is specific to the type of object. The name of 'performance signal' is of the form `debug-`*objectName-SomeName*. Currently, only the `application`, the `scene` and `graph` objects emit 'performance signals'.

Chapter 13

Signals and graphic signals

The graphic representation of a signal is approached with *graphic signals*. As illustrated in figure 13.1, a graphic representation of a signal could be viewed as a stream of a limited set of parameters : the y coordinate, the graph thickness h and the graph color c. A *graphic signal* is a composite signal including a set of 3 parallel signals that control these parameters. Thus the INScore library provides messages to create signals and to combine them into *graphic signals*.

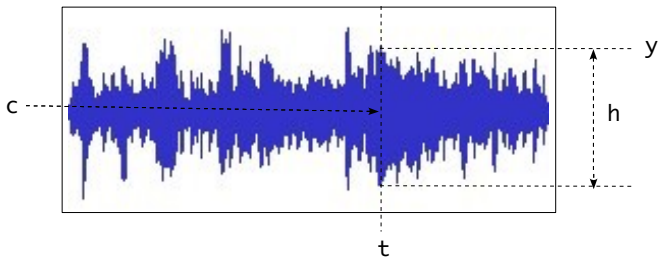


Figure 13.1: A simple graphic signal, defined at time t by a coordinate y, a thickness h and a color c

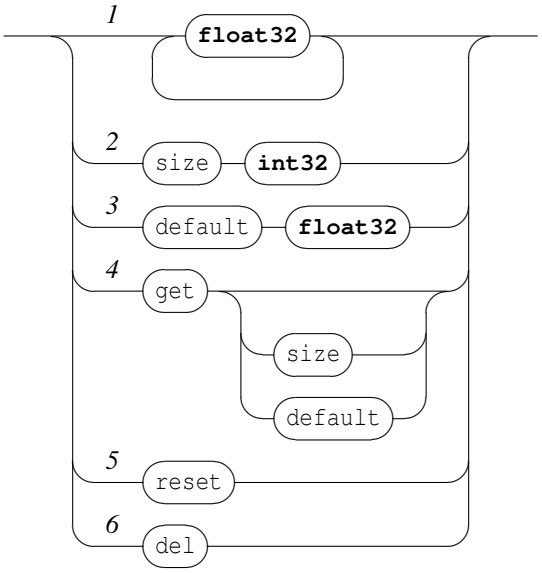
13.1 The *signal* virtual address space.

A `scene` includes a virtual address space, which OSC address is `/ITL/scene/signal`. This address space is reserved for *simple signals* as well as for composing signals in parallel.
The `signal` virtual address space supports only the `get` messages that gives the list of the defined signals with a set of messages complying to the syntax defined in section 13.1.1.

13.1.1 Simple signals.

The signal messages can be sent to any address with the form `/ITL/scene/signal/`*`identifier`*, where *`identifier`* is a unique signal identifier. The set of messages supported by a signal is the following:

simpleSignal



- **[1]** push an arbitrary data count into the signal buffer. The data range should be `[-1,1]`. Note that the signal buffer behaves like a ring buffer, thus data are wrapped when the data count if greater than the signal size.
- **[2]** the `size` message sets the signal buffer size. When not specified, the buffer size value is the size of the first data message.
- **[3]** the `default` message sets the *default signal value*. A signal *default value* is the value returned when a query asks for data past the available values.
- **[4]** the `get` gives the signal values. The `size` and `default` parameters are used to query the signal size and default values.
- **[5]** the `reset` message clears the signal data.
- **[6]** the `del` message deletes the signal from the `signal` space. Note that it is safe to delete a signal even when used by a graphic signal.

Example:

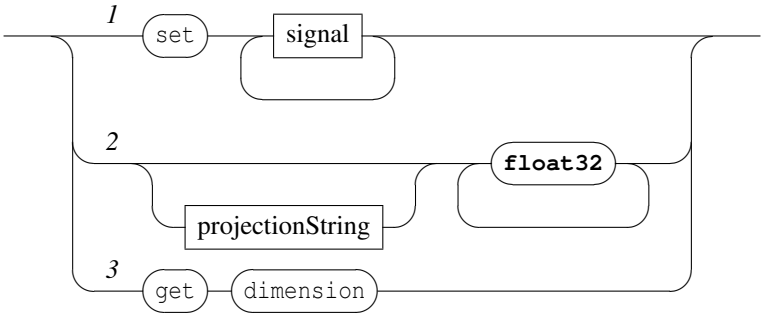
```
/ITL/scene/signal/null 0.0
```

creates a constant signal named `null`, which value is `0`.

13.1.2 Composing signals in parallel.

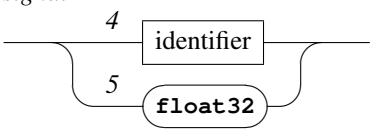
Composing signals in parallel produces a signal which value at a time `t` is a vector of the composed signals values. Thus an additional property is defined on *parallel signals* : the signal *dimension* which is size of the signals values vector. Note that the dimension property holds also for simple signals. The format of the messages for parallel signals is the following:

parallelSignal



where

signal



- **[1]** defines a new signal composed of the signals list and which address is the message OSC address. A signal list element is defined as:
  - **[4]** an `identifier` i.e. a signal name refering to an existing signal in the `signal` virtual address space.
  - **[5]** or as a float value. This form is equivalent to an anonymous constant signal holding the given value.
- **[2]** sets the values of the signals addressed by a projection string. See section 13.1.3 p.30.
- **[3]** in addition to the`get` format above, a parallel signal supports the `get dimension` message, which gives the number of simple signals in parallel. It is actually the sum of each signal dimension. The dimension of a simple signal is 1.

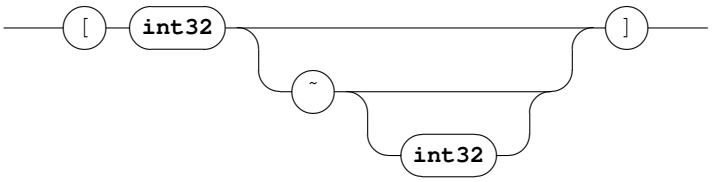
Note that for a parallel signal:

- the `get` message gives the list of the signals in parallel with the exact form of the composing message in [1].
- the `get size` message gives the maximum of the components size.
- the `get default` message gives the default value of the first signal.

13.1.3 Signals projection.

Individual components of a parallel signal may be addressed using a *projection string* which is defined as:

projectionString



The projection string is made of a *index value*, followed by an optional *parallel marker* (`~`), followed by an optional *step value*, all enclosed in brackets. The *index value* `n` is the index of a target signal. When the *parallel marker* option is not present, the values are directed to the target signal and the message is

equivalent to: `/ITL/scene/signal/target values...`
where *target* is the name of the target signal addressed by `n`.
Note that:

- the message is ignored when `n` is greater than the number of signals in parallel. Default `n` value is `0`.
- setting directly the values of a simple signal or as the projection of a parallel signal are equivalent.

The *parallel marker* (`~`) and the *step value* `w` options affect the target signals. Let's consider `s[n]` as the signal at index `n`. The values are distributed in sequence and in loop to the signals `s[n], s[n+w]...s[m]` where `m` is the greatest value of the index `n+(w.i)` that is less than the signal dimension. The default *step value* is `1`.
Example:

the message:

```
/ITL/scene/signal/target [2~] 0.1 0.2
```

is equivalent to the following messages:

```
/ITL/scene/signal/target [2] 0.1
/ITL/scene/signal/target [3] 0.2
```

13.2 Graphic signals.

A graphic signal is created in the standard scene address space. A simple graphic signal is defined by a parallel signal controling the `y` value, the graphic thickness and the color at each time position. The color is encoded as HSBA colors (Hue, Saturation, Brightness, Transparency). The mapping of a signal value (`[-1,1]`) to the HSBA color space is given by the table 13.1.

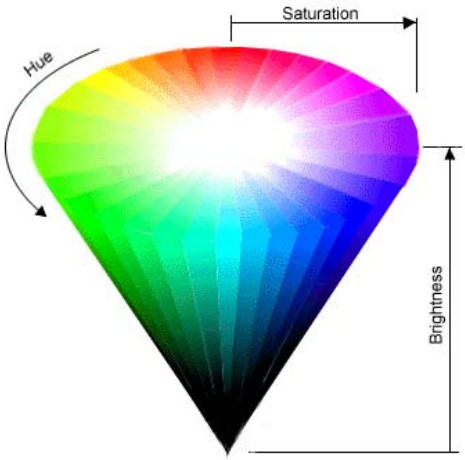


Figure 13.2: The HSB color space

A graphic signal responds to common component messages (section 4 p.6). In addition, it supports the following messages:

graphicSignal

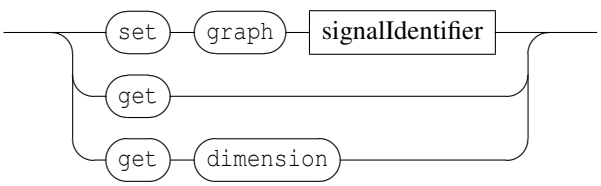


Table 13.1: HSBA color values.

parameter	mapping	
`hue`	`[-1,1]`	corresponds to `[-180,180]` angular degree where `0` is red.
`saturation`	`[-1,1]`	corresponds `0%` to `100%` saturation.
`brigthness`	`[-1,1]`	corresponds `0%` (black) to `100%` (white) brithgness.
`transparency`	`[-1,1]`	corresponds `0%` to `100%` tranparency.

- the `set` message is followed by the `graph` type and a *`signalIdentifier`*, where `signalIdentifier` must correspond to an existing signal from the `signal` address space. In case `signalIdentifier` doesn't exist, then a new signal is created at the `signalIdentifier` address with default values.
- the `get` message is the counterpart of the `set` message (see section 7 p.18).
- the `get dimension` message gives the number of graphic signals in parallel (see section 13.2.2 p.32).

13.2.1 Graphic signal default values.

As mentionned above, a graphic signal expects to be connected to parallel signals having at least an `y` component, a graphic thickness component and HSBA components. Thus, from graphic signal viewpoint, the expected dimension of a signal should be equal or greater than 6. In case the `signalIdentifier` dimension is less than 6, the graphic signal will use the default values defined in table 13.2.

Table 13.2: Graphic signal default values.

parameter	default value	
`y`	`0`	the center line of the graphic
`thickness`	`0`	
`hue`	`0`	meaningless due to brigthness value
`saturation`	`0`	meaningless due to brigthness value
`brigthness`	`-1`	black
`transparency`	`1`	opaque

13.2.2 Parallel graphic signals.

When the dimension d of a signal connected to a graphic signal is greater than 6, then the input signal is interpreted like parallel graphic signals. More generally, the dimension n of a graphic signal is:

$$n \mid n \in \mathbb{N} \wedge 6.(n-1) < d \leqslant 6.n$$

where d is the dimension of the input signal.
Note than when d is not a mutiple of 6, then the last graphic signal makes use of the default values mentionned above.

Chapter 14

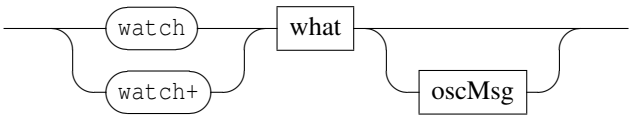
Events and Interaction

Interaction messages are user defined messages associated to events and triggered when these events occur. These messages accept variables as message arguments.

14.1 Interaction messages

The general form of the message is:

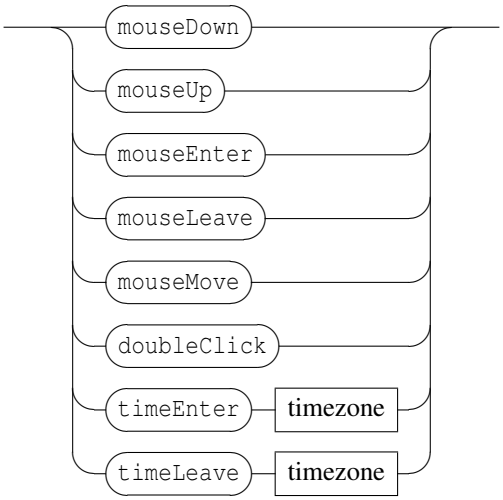
interactMsg



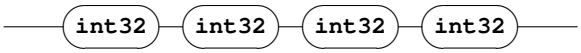
- `watch`: clears the messages associated to the event and when specified, associates the new message to the watched event.
- `watch+`: adds a message to the watched event message list.

Events currently *watchable* are:

what

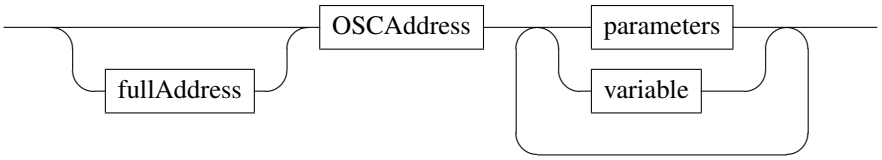


timezone

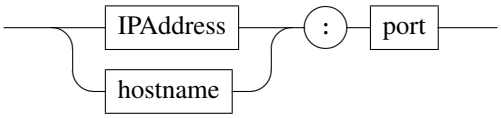


A time zone is defined by 2 dates expressed as rational values (i.e. with 4 integers).
The associated OSC message is any valid OSC message (not restricted to the Interlude message set), with an extended address scheme, supporting IP addresses or host names and udp port number to be specified as OSC addresses prefix. The message parameters are any valid OSC type or variables (see section 14.2).

oscMsg



fullAddress



Example:

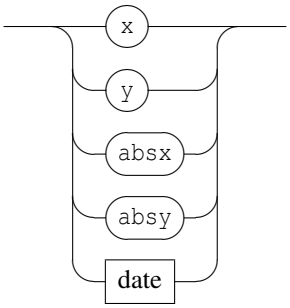
```
1) /ITL/scene/myObject watch mouseDown "/ITL/scene/myObject" "show" 0
2) /ITL/scene/myObject watch mouseDown "host.domain.org:12100/an/address" "start"
```

1) Request the object `myObject` to watch `mouseDown` events and send a message to itself.
2) Request the object `myObject` to watch `mouseDown` events and send a "start" message to host.domain.org on udp port 12100 to OSC address /an/address

14.2 Message variables

Variables denote values computed when an event is triggered. These values are send in place of the variable. A variable name starts with a '$' sign. Currently, the following variables are supported by mouse events:

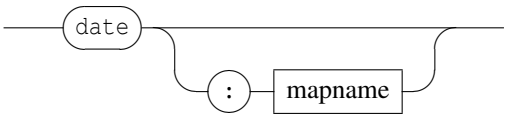
variable



- `$x` `$y`: denotes the mouse pointer position at the time of the event. The values are in the range $[0,1[$ where 1 is the object size in the x or y dimension. The value is computed according to the object origin: it represents the mouse pointer distance from the object x or y origin (see 4.1.3 p.8).

- `$absx` `$absx`: denotes the mouse pointer absolute position at the time of the event. The values represent a pixel position relative to the top-left point of the target object. Note that this position is unaffected by scale. Note also that the values are not clipped to the object dimensions and could exceed its width or height or become negative in case of mouse move events.
- `$date`: denotes the object date corresponding to the mouse pointer position at the time of the event. The detailed date variable is given below: it is optionnaly followed by a colon and the name of the mapping to be used to compute the date . The `$date` variable is replaced by its rational value (i.e. two integers values).

date



Note that a variable can be used several times in a message, but several `$date` variables must always refer to the same mapping.

`Warning:` the interaction message set is provided for experiment. It is likely to change in a future version. Ideas, comments, suggestions are welcome for the design of a stable API.

Chapter 15

Changes list

15.1 Differences to version 0.60

- new 'mousemove' event (see section 14.1 p.33).
- interaction messages accept variables ($x, $y, $date...) 14.2 p.34).
- SVG code and files support (see section 6 p.17).
- set line message change: the `x y` form is deprecated, it is replaced by the following forms: `'xy' x y` (equivalent to the former form) and `'wa' width angle` (see section 6 p.15).
- new 'effect' message (section 4.3 p.10).
- utf8 support on windows corrected
- transparency support for stretched synchronized objects corrected
- multiple application instances supported with dynamic udp port number allocation.
- command line option with –port portnumber option to set the receive udp port number at startup.

15.2 Differences to version 0.55

- new 'xorigin' and 'yorigin' messages (section 4.1.3 p.8).
- new interaction messages set (section 14 p.33).
- alpha channel handled by images and video
- bug correction in line creation corrected (false incorrect parameter returned)
- bug correction in line 'get' message handling
- memory leak correction (messages not deleted)

Known issues:

- incorrect graphic rendering when 'sync a b' is changed to 'sync b a' in the same update loop
- incorrect nested synchronization when master is horizontaly stretched,

15.3 Differences to version 0.53

- ITL parser corrected to support regexp in message string (used by messages addressed to sync node)
- format of mapping files and strings changed (section 9 p.20).
- format of sync messages extended to include map name (section 10 p.22).
- signal node: 'garbage' message removed
- new 'reset' message for the scene (/ITL/scene) (section 3 p.5).
- new 'version' message for the application (/ITL) (section 2 p.3).
- new 'reset' message for signals (section 13.1.1 p.28).

- bug parsing messages without params corrected
- slave segmentation used for synchronization
- new H synchronization mode (preserves slave segmentation)
- crash bug corrected for load message and missing ITL files

15.4 Differences to version 0.50

- Graphic signal thickness is now symmetrically drawn around y position.
- ITL file format supports regular expressions in OSC addresses.
- IP of a message sender is now used for the reply or for error reporting.
- new `line` object (section 6 p.15).
- new `penStyle` message for vectorial graphics (section 8 p.19).
- new color messages `red, green, blue, alpha, dcolor, dred, dgreen, dblue` (section 4 p.6 and 4.1.2 p.8).
- color values for objects are bounded to [0,255]
- `get map` message behaves according to new map message (section 7 p.18).
- `get width` and `get height` is now supported by all objects (section 7 p.18).
- bug in signal projection corrected (index 0 rejected)
- bug in signals default value delivery corrected
- new `pageCount` message for guido scores
- debug nodes modified state propagated to parent node (corrects the debug informations graphic update issue)
- rational values catch null denominator (to prevents divide by zero exceptions).

15.5 Differences to version 0.42

- `identifier` specification change (section 1 p.1).
- new application `hello` and `defaultShow` messages (section 2 p.3).
- new `load` and `save` messages (sections 2 p.3 and 4 p.6).
- `click` and `select` messages (section 4.4 p.11):
  - `rightbottom` and `leftbottom` modes renamed to `bottomright` and `bottomleft`
  - new `center` mode for the `click` message
  - query mode sent back with the reply both for `click` and `select` messages
- new `file`, `html` and `htmlf` types for the `set` message (section 6 p.15).
- `get` syntax change for the `scene` (section **??** p.**??**).
- `fileWatcher` messages completely redesigned (section 12.1 p.26).
- mappings can be identified by names (section 9 p.20).
- rect, ellipse, curve, line and polygon object support graphic to relative-time mapping
- new synchronization modes for Guido scores: voice1, voice2, ... , staff1, staff2, ... , system, page (section 11.1 p.24).
- Guido mapping manages repeat bars.
- Graphic signals messages design (section 13.2 p.31).

Index