## Contents

1 General format ................................. 1  
   1.1 Parameters ........................................ 1  
   1.2 Address space .................................... 2  

2 Application messages .......................... 3  

3 Scene messages ................................... 5  

4 Common components messages ................. 6  
   4.1 Positioning ........................................ 7  
      4.1.1 Absolute positioning ............................ 7  
      4.1.2 Relative positioning ............................ 8  
      4.1.3 Components origin .............................. 8  
      4.1.4 Components transformations .................... 9  
   4.2 Color messages ................................. 9  
      4.2.1 Absolute color messages ....................... 9  
      4.2.2 Relative color messages ....................... 10  
   4.3 The 'effect' messages ........................... 11  
      4.3.1 The blur effect ................................. 11  
      4.3.2 The colorize effect ............................. 12  
      4.3.3 The shadow effect ............................... 12  
   4.4 The 'click' and 'select' messages .......... 11  

5 Time management messages ..................... 14  

6 The 'set' message ................................ 15  

7 The 'get' messages .............................. 18  
   7.1 Special 'get' forms .............................. 18  

8 Component specific messages ................... 19  
   8.1 Pen control ....................................... 19  
   8.2 Width and height control ....................... 19  
   8.3 Score pages management ....................... 20  

9 Time to space mapping .......................... 21
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.

\[\text{OSCMMessage}\]

\[\text{OSCAddress } \Rightarrow \text{ OSCAddress } \Rightarrow \text{ message } \Rightarrow \text{ parameters}\]

\[\text{OSCAddress} \Rightarrow / \Rightarrow \text{ identifier } \Rightarrow \text{ regexp}\]

\[\text{identifier} \Rightarrow [-_a-zA-Z] \Rightarrow [-_a-zA-Z0-9]]\]

Some specific nodes accept (like signals - see section 13.1.1) OSC messages without message string:

\[\text{OSCMMessage}\]

\[\text{OSCAddress } \Rightarrow \text{ parameters}\]

1.1 Parameters

Message parameters types are between the OSC types \textit{int32}, \textit{float32} and \textit{OSC-string}. In the remainder of this document, they are used as terminal symbols, denoted by \textit{int32}, \textit{float32} and \textit{string}.

When used in a script file (see section 16), \textit{string} should be single or double quoted. If an ambiguous double or single quote is part of the string, it must be escaped using a `\'\'.
Parameters types policy is relaxed: the system makes its best to convert a parameter to the expected type, which depend on the message string. When an incorrect type and when no conversion is applied, an incorrect parameter message is issued.

1.2 Address space

The OSC address space is made of static and dynamic nodes, hierarchically organized as follows (figure 1.1):

![Diagram of the OSC address space](image)

Figure 1.1: The OSC address space. Nodes in italic/blue are dynamic nodes.

OSC messages are accepted at any level of the hierarchy:

- the application level responds to messages for application management (udp ports management, loading files, query messages)
- at scene level, each node is associated to a window and respond to specific scene/window management messages.
- the component level contains the scene objects and two static nodes:
  - a signal node that may be viewed as a folder containing signals
  - a sync node, in charge of the synchronization messages
- the signals level contains signals i.e. objects that accept data streams and that may be graphically rendered as a scene component.
Chapter 2

Application messages

Application messages are accepted by the static OSC address /ITL. It covers the following messages:

- **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.
- **mouse**: hide or show the mouse pointer.
- **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. Where necessary, it’s the sender responsibility 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.

- **require**: check for a version number equal or greater to the number given as argument. The version number should be encoded as an integer value (for example: 76 for version 0.76, 112 for version 1.12). An OSC message is associated to the require message with the same syntax and semantic described in section 15 p.38. This message is triggered when the check fails.

The application supports the `get` messages for its parameters (see section 7 p.18). In addition, it provides the following messages to query version numbers.

**ITLRequest**

```
get
   version
      guido-version
      musicxml-version
```

- **version**: version number request.
- **guido-version**: Guido engine version number request.
- **musicxml-version**: MusicXML and Guido converter version numbers request. Returns "not available" when the library is not found.
Chapter 3

Scene messages

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

sceneMsg

- **new**: creates a new scene and opens it in a new window.
- **del**: deletes a scene and closes the corresponding window.
- **rootPath**: rootPath of a scene is the default path where the scene reads or writes a file when a relative path is used for this file. When no value has been specified, the application rootPath is used.
- **load**: loads an INScore file to the scene. Note that the OSC addresses are translated to the scene OSC address.
- **reset**: clears the scene (i.e. delete all components) and resets the scene to its default state (position, size and color).
- **foreground**: put the scene to foreground.
- **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. When the destination file is not completely specified (third form or missing extension), there is an automatic numbering of output names when the destination file already exists.

- **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.
- **rename**: rename the destination object. Changes its OSC address. **Warning**: OSC pattern matching allows to give the same name to a whole collection of objects; in this case, there is no way to individually address an object when its name is shared with other objects.
  - 'PositionMsg' are absolute and relative position messages.
  - 'ColorMsg' are absolute and relative color control messages.
  - 'TimeMsg' are time management messages. They are described in chapter 5 p.14.
  - 'clickSelectMsg' are provided to query objects relative positions.

### 4.1 Positioning

#### PositionMsg

- **absPosMsg**
- **relPosMsg**
- **originMsg**

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

#### 4.1.1 Absolute positioning

**absPosMsg**

- **x** float32
- **y** float32
- **z** float32
- **angle** float32
- **scale** float32

- 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.
• \( \text{angle} \): sets the \( \text{angle} \) 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.
• \( \text{scale} \): reduce/enlarge a component. The range is \([0, \infty)\).
  Default scale is 1.

### 4.1.2 Relative positioning

**relPosMsg**

- \( \text{dx} \):
  - \( \text{dy} \):
  - \( \text{dz} \):
  - \( \text{dangle} \):
  - \( \text{d scale} \):

\( \text{dx}, \text{dy}, \text{etc. messages are similar to } x, y, \text{etc. messages but the parameters represents values represent a displacement of the current target value.} \)

\( \text{dscale is similar to scale but the parameters represents a scale multiplying factor.} \)

### 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**

- \( \text{xorigin} \):
  - \( \text{yorigin} \):
  - \( \text{dxorigin} \):
  - \( \text{dyorigin} \):

\( \text{xorigin, yorigin are relative to the component coordinates space i.e. } [-1, 1], \text{ where -1 is the top or left border and 1 is the bottom or right border. The default origin is } [0, 0] \text{ i.e. the component is centered on its } (x, y) \text{ coordinates.} \)

\( \text{dxorigin, dyorigin represents displacement of the current xorigin or yorigin.} \)
4.1.4 Components transformations

A component transformation specifies 2D transformations of its coordinate system. It includes shear and object rotation.

\[
\text{transformMsg} \\
\begin{array}{c}
\text{rotatex} \\
\text{rotatey} \\
\text{rotatez} \\
\text{shear} \\
x \\
y
\end{array}
\]

- \text{rotatex} \text{ rotatey} \text{ rotatez}: rotates the component around the corresponding axis. Parameter value expresses the rotation in degrees.
- \text{shear} transforms the component in x and y dimensions. x and y are float values expressing the transformation value in the corresponding dimension.

4.2 Color messages

\[
\text{ColorMsg} \\
\begin{array}{c}
\text{absColorMsg} \\
\text{relColorMsg}
\end{array}
\]

Color messages are absolute or relative color control messages.

4.2.1 Absolute color messages

\[
\text{absColorMsg} \\
\begin{array}{c}
\text{color} \\
\text{hsb} \\
\text{red} \\
\text{green} \\
\text{blue} \\
\text{alpha} \\
\text{hue} \\
\text{saturation} \\
\text{brightness}
\end{array}
\]

\[
\text{colorvalue} \\
\text{colorvalue} \\
\text{colorvalue} \\
\text{colorvalue} \\
\text{colorvalue} \\
\text{colorvalue} \\
\text{colorvalue} \\
\text{colorvalue}
\]
The value of a color value may be specified as integer or float values. When expressed as float, the value range is $[-1,1]$. The correspondence between the float and integer ranges is expressed below.

- **color**: sets an object color. The color scheme is RGBA. When A is not specified, the color is assumed to be opaque. The integer data range for each color component is $[0,255]$. The float data range $[-1,1]$ is linearly mapped to $[0,255]$. 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 $[0,100]$, $[0,360]$ for H, and $[0,255]$ for A. The float data range $[-1,1]$ is mapped to $[0,100]$ for S and B. For H, 0 refers always to the red color and thus $[0,1]$ is mapped to $[0,180]$ and $[-1,0]$ is mapped to $[360,180]$.

4.2.2 Relative color messages

- **relColorMsg**
- `dred` and `dgreen` messages are similar to `red` and `green` 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

```
<table>
<thead>
<tr>
<th>colorizeParams</th>
</tr>
</thead>
<tbody>
<tr>
<td>strength</td>
</tr>
<tr>
<td>color</td>
</tr>
</tbody>
</table>
```

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.9) by default, the color value is light blue (0, 0, 192).

4.3.3 The shadow effect

```
<table>
<thead>
<tr>
<th>shadowParams</th>
</tr>
</thead>
<tbody>
<tr>
<td>xoffset</td>
</tr>
<tr>
<td>yoffset</td>
</tr>
<tr>
<td>color</td>
</tr>
<tr>
<td>blur</td>
</tr>
</tbody>
</table>
```

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.9) 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

**Warning**: the click and select messages are deprecated and will be removed in a future version. 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.

```
<table>
<thead>
<tr>
<th>click</th>
</tr>
</thead>
<tbody>
<tr>
<td>clickMode</td>
</tr>
</tbody>
</table>
```

clickMode

```
<table>
<thead>
<tr>
<th>topleft</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottomright</td>
</tr>
<tr>
<td>topright</td>
</tr>
<tr>
<td>bottomleft</td>
</tr>
<tr>
<td>center</td>
</tr>
</tbody>
</table>
```
• **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:

\[
\text{clickReply} \quad \text{OSCAddress} \quad \text{clickMode} \quad \text{identifier}
\]

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

`select`

\[
\text{select} \quad \text{selectMode}
\]

`selectMode`

\[
\text{intersect} \quad \text{include}
\]

- **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`:

\[
\text{selectReply} \quad \text{OSCAddress} \quad \text{selectMode} \quad \text{identifier}
\]
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**: 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. Default value is 0/1.
- **duration**: changes the object duration. Default value is 1/1.
- **ddate**: relative time positioning message: adds the specified value to the object date.
- **dduration**: relative duration message: adds the specified value to the object duration.

- Time is specified as a rational value \( \frac{d}{n} \) where \( n \) represents the whole note division and \( d \) the divisions count. The parameters order is numerator followed by denominator.
- Time may also be specified with a single integer, then 1 is used as implicit denominator value
- or using a single float value converted to int32 using the following approximation: when \( f \) is the floating point value, the corresponding rational value is \( f \times \frac{10000}{10000} \).

- int32
- float32

- int32
- int32

- int32
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.

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.31 for details about the graph objects data.
- **rect**: a rectangle specified by a width and height. Width and height are expressed in scene coordinate 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 account 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 one of the txtf, gmnf, img or video types, according to the file extension (see table 6.1).

Format of the set message is:

```
setMsg

set
  txt txtStream
  html string
  svg svgString
  gmn gmnString
  graph signals
  rect width height
  ellipse width height
  polygon x y
  curve x1 y1 x2 y2 x3 y3 x4 y4
  line xy x y
  wa width angle

setFile
```

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

```
txtStream

string
  int32
  float32
```
**setFile**

<table>
<thead>
<tr>
<th>file extension</th>
<th>translated type</th>
</tr>
</thead>
<tbody>
<tr>
<td>.txt</td>
<td>txtf</td>
</tr>
<tr>
<td>.htm</td>
<td>htmlf</td>
</tr>
<tr>
<td>.gmn</td>
<td>gmnf</td>
</tr>
<tr>
<td>.svg</td>
<td>svgf</td>
</tr>
<tr>
<td>.jpg .jpeg .png .gif .bmp .tiff</td>
<td>img</td>
</tr>
<tr>
<td>.avi .wmv .mpg .mpeg .mp4</td>
<td>video</td>
</tr>
</tbody>
</table>

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

\[
\text{getMsg}
\]

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 addressed 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.

8.1 Pen control

Messages accepted by the components types rect | ellipse | polygon | curve | line | graph.

\[\text{penMsg} \]
\[
\begin{array}{c}
\text{penColor} \\
\text{penWidth} \\
\text{penStyle}
\end{array}
\]

\[\text{penstyle} \]
\[
\begin{array}{c}
\text{solid} \\
\text{dash} \\
\text{dot} \\
\text{dashDot} \\
\text{dashDotDot}
\end{array}
\]

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. The color scheme is described in section 4.2.

8.2 Width and height control

Messages accepted by the components types rect | ellipse | graph.
8.3 Score pages management

Messages accepted by the components types `gmn` | `gmnf`.

- **page**: set the score current page
- **dpage**: set the score current page
- **pageFormat**: set the page format. The parameters are the page width and height. Note that the message has no effect when the score already includes a `\pageformat` tag.
- **columns**: for multi pages display: set the number of columns.
- **rows**: for multi pages display: set the number of rows.
- **pageCount**: a read only attribute, gives the score pages count.
Chapter 9

Time to space mapping

Time to space mapping refers to the description of relationship between an object local graphic space and its time space.

9.1 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

<table>
<thead>
<tr>
<th>component type</th>
<th>segmentation type</th>
</tr>
</thead>
<tbody>
<tr>
<td>txt, txtf</td>
<td>int2DSegments</td>
</tr>
<tr>
<td>img</td>
<td>int2DSegments</td>
</tr>
<tr>
<td>rect, ellipse, polygon, curve</td>
<td>float2DSegments</td>
</tr>
<tr>
<td>graph</td>
<td>int1DSegments</td>
</tr>
</tbody>
</table>

Note for html, htmlf + guido score mappings

relativeTimeInterval

float2DSegment

int2DSegment

int1DSegment

relativeTimeInterval

floatInterval

intInterval

rational

float32

int32

22
Relative time is expressed as rational values where 1 represents a whole note.

$$\text{rational}$$

\[ \frac{\text{int32}}{-\text{int32}} \]

### 9.2 The ’map+’ message

The map+ messages is similar to the map message but doesn’t replace the existing mapping data: the specified relations are added to the existing one.

$$\text{mapAddMsg}$$

$$\text{map}^+ \quad \text{relation} \quad \text{mapName}$$

### 9.3 Mapping files

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.

$$\text{mapfMsg}$$

$$\text{mapf} \quad \text{mapFilePath} \quad \text{mapName}$$
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.

- [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).

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 respond 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`
- `mapName`

### `syncAlignment`

- `syncOver`
- `syncTop`
- `syncBottom`

### `syncStretch`

- `h`
- `v`
- `hv`

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 correspondence 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
- **systemflat**: a system level mapping without system subdivision (one graphic segment per system)
- **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 rational 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 described in section 1 p.1.

Example:

```
/ITL/scene/fileWatcher add 'fuga.gmn' '/ITL/scene/score' '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 activated or deactivated with 0 or 1:

```plaintext
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](image)

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. **float32**

2. **size** **int32**

3. **default** **float32**

4. **get**

   - **size**
   - **default**

5. **reset**

6. **del**

- [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:

```plaintext
/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

1

set

2

signal

float32

projectionString

3

get

dimension

where

signal

4

identifier

float32

5

- [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 referring 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.33.
- [3] in addition to the get format defined for signals, 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

1

int32

The projection string is made of an index value, followed by an optional parallel marker (\(\sim\)), followed by an optional step value, all enclosed in brackets. The index value \(n\) is the index of a target signal. When
the \textit{parallel marker} option is not present, the values are directed to the target signal and the message is equivalent to: 

\texttt{/ITL/scene/signal/target values...}

where \texttt{target} is the name of the target signal addressed by \texttt{n}.

Note that:

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

The \textit{parallel marker} ($\sim$) and the \textit{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 $s[n], s[n+w], \ldots s[m]$ where $m$ is the greatest value of the index $n+(w,i)$ that is less than the signal dimension. The default \textit{step value} is 1.

Example:

the message:

\begin{verbatim}
/ITL/scene/signal/target [2˜] 0.1 0.2
\end{verbatim}

is equivalent to the following messages:

\begin{verbatim}
/ITL/scene/signal/target [2] 0.1
/ITL/scene/signal/target [3] 0.2
\end{verbatim}

### 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 controlling 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](image)

A graphic signal responds to common component messages (section 4 p.6). In addition, it supports the following messages:
Table 13.1: HSBA color values.

<table>
<thead>
<tr>
<th>parameter</th>
<th>mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>hue</td>
<td>([-1,1]) corresponds to ([-180,180]) angular degree where 0 is red.</td>
</tr>
<tr>
<td>saturation</td>
<td>([-1,1]) corresponds 0% to 100% saturation.</td>
</tr>
<tr>
<td>brightness</td>
<td>([-1,1]) corresponds 0% (black) to 100% (white) brightness.</td>
</tr>
<tr>
<td>transparency</td>
<td>([-1,1]) corresponds 0% to 100% transparency.</td>
</tr>
</tbody>
</table>

13.2.1 Graphic signal default values.

As mentioned 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.

<table>
<thead>
<tr>
<th>parameter</th>
<th>default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y)</td>
<td>0</td>
</tr>
<tr>
<td>thickness</td>
<td>0</td>
</tr>
<tr>
<td>hue</td>
<td>0</td>
</tr>
<tr>
<td>saturation</td>
<td>0</td>
</tr>
<tr>
<td>brightness</td>
<td>-1</td>
</tr>
<tr>
<td>transparency</td>
<td>1</td>
</tr>
</tbody>
</table>

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:

\[ n \mid n \in \mathbb{N} \land 6 \cdot (n - 1) < d \leq 6 \cdot n \]

where \(d\) is the dimension of the input signal.

Note than when \(d\) is not a multiple of 6, then the last graphic signal makes use of the default values mentioned above.
Chapter 14

FAUST plugins

FAUST [Functional Audio Stream] is a functional programming language specifically designed for real-time signal processing and synthesis. A FAUST/INScore architecture allows to embed FAUST processors in INScore, for the purpose of signals computation. A FAUST/INScore plugin is viewed as a parallel signal and thus it is created in the signal address space. Thus and similarly to signals, a FAUST plugin is associated to OSC addresses in the form /ITL/scene/signal/name where name is a user defined name.

faustprocessor

\[
\begin{array}{c}
\text{set} \quad \text{faust} \quad \text{path} \\
\end{array}
\]

A FAUST processor is created with a set message followed by the faust type, followed by the plugin path name. Thus the faust name is a reserved signal name.

14.1 Specific messages

A FAUST processor is characterized by the numbers of input and output channels and by a set of parameters. Each parameter carries a name defined by the FAUST program. The set of messages supported by a FAUST processor is the set of signals messages extended with the parameters names and with specific query messages.

faustmessage

\[
\begin{array}{c}
\text{signalMsgs} \\
1 \quad \text{msg} \quad \text{float32} \\
2 \quad \text{get} \quad \text{in} \quad \text{out} \\
\end{array}
\]

1 \( \text{msg} \) is any of the FAUST processor parameters, as defined by the FAUST program.
2 the \text{get} message is extended to query the FAUST processor: \text{in} and \text{out} give the number of input and output channels.

\(^1\)http://faust.grame.fr
14.2 Feeding and composing FAUST processors

A FAUST processor accepts float values as input, which are taken as interleaved data. From composition viewpoint, a FAUST processor is a parallel signal which dimension is the number of output channels. Thus, a FAUST processor can be used like any parallel signal. However, the signal identifier defined in 13.1.2 is extended to support addressing single components of parallel signal as follows:

\[
\text{signal identifier} / n
\]

where \( n \) selects the signal \#n of a parallel signal. Note that indexes start at 0.
Chapter 15

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.

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` without arguments ([1]) clears all the messages associated to all events.
- `watch+`: adds a message to the watched event message list.

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 15.3).

```
oscMsg
```

```
fullAddress
```

```
IPAddress
```

```
hostname
```

```
port
```
15.1 Interaction messages

Events currently watchable are:

- `mouseDown`
- `mouseUp`
- `mouseEnter`
- `mouseLeave`
- `mouseMove`
- `doubleClick`

Example:

```
/ITL/scene/myObject watch mouseDown "/ITL/scene/myObject" "show" 0;
```

Request the object `myObject` to watch `mouseDown` events and send a message to itself.

```
/ITL/scene/myObject watch mouseDown "host.domain.org:12100/an/address" "start";
```

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`

15.2 Notification messages

Events currently watchable are:

- `timeEnter`
- `timeLeave`
- `newElement`

- `timezone`

```
timezone
```

A time zone is defined by 2 dates expressed as rational values (i.e. with 4 integers).

The `newElement` event is supported at scene level only and triggered when a new element is added to the scene.

15.3 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:
$xy$: 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). $sx$ and $sy$ variables support an optional range in the form $[low, high]$ that transforms the $[0, 1]$ values range into the $[low, high]$ range.

$absx$ $absy$: 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.

$sx$ $sy$: denotes the mouse pointer position in the scene coordinates space. $sx$ and $sy$ are computed like

$date$: 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 optionally 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). The optional rational enclosed in brackets may be used to indicate a quantification: the date value is rounded to an integer count of the specified rational value.

$name$, $address$: replaced by the target object name or OSC address. Note that in case of newElement event, the target object is the new element.

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.

### 15.4 Message based variables

A message based variable is a variable containing an OSC message which will be evaluated at the time of the event. They are supported by all kind of events. Like the variables above, a message based variable starts with a `$` sign followed by a valid `get` message enclosed in parenthesis:

```
msgVar
```

The evaluation of a `get` message produces a message or a list of messages. The message based variable will be replaced by the parameters of the messages resulting from the evaluation of the `get` message. Note that all the `get` messages attached to an event are evaluated at the same time.

**Example:**

```
/ITL/s/o1 watch 'mouseDown' '/ITL/s/o1' 'show' $(/ITL/s/o2 get 'show');
/ITL/s/o1 watch+ 'mouseDown' '/ITL/s/o2' 'show' $(/ITL/s/o1 get 'show');
/ITL/s/o2 watch 'mouseDown' '/ITL/s/o1' 'show' $(/ITL/s/o2 get 'show');
/ITL/s/o2 watch+ 'mouseDown' '/ITL/s/o2' 'show' $(/ITL/s/o1 get 'show')
```

When clicked, the objects o1 and o2 swap their visibility state.

### 15.5 OSC address variables

The OSC address parameter of a watch message supports the following variables:

- `$self`: replaced by the object name at the time of the message setting.
- `$scene`: replaced by the scene name at the time of the message setting.

Note: the system doesn’t check for the consistency of the variables placement.
Chapter 16

Scripting

Since INScore saves its states as text files containing lists of OSC messages, it becomes natural to design
INSCore scenes using these messages as a textual scripting language. For a better support of scripting,
_variables have been introduced to provide a better parameters control and programming languages have
been embedded to generate messages algorithmically.

16.1 Variables

A variable may hold any value supported by the parameter types. _string_ must be enclosed in simple or
double quotes. It is declared as follows:

```
variabledecl
  ident = int32 ;
  float32
  string
```

A variable may be used in place of any message parameter. A reference to a variable must have the form
$ident where ident is a previously declared variable.

Note that interaction messages may also use variables with similar reference forms ($ident). There is
however no ambiguity concerning these references: interaction variables are always passed as strings and
should be quoted, scripts variables as used without quotes.

Example:

```
x=0.5;
/ITL/scene/a x $x;
/ITL/scene/b x $x;
```

16.2 Languages

INScore supports Javascript and Lua as scripting languages. Javascript is embedded by default (using the v8
engine - see http://code.google.com/p/v8/). You need to recompile INScore to embed the Lua engine
(http://www.lua.org/).

The principle of scripting using a programming language is the following: you can embed _javascript of
lua_ code in an INScore file in a way similar to scripts embedded into html files. This code is expected to
produce INScore messages on output.

`lang`

The output of the script section is parsed like INScore messages. Note that INScore variables are exported to the current language environment.

Example:

```javascript
<?javascript
    "/*ITL/scene/version set 'txt' 'Javascript v." + version() + ");";
?>
```
Chapter 17

Changes list

17.1 Differences to version 0.95
- switch to v8 javascript engine
- lua not embedded by default

17.2 Differences to version 0.92
- new 'mouse' 'show/hide' message supported at application level (see section 2 p.3)
- graphic signal supports alpha messages at object level
- javascript and lua embedded and supported in inscore scripts (see section 16.2 p.42).
- bug correction in sync delete (introduced with version 0.90)

17.3 Differences to version 0.91
- bug corrected: crash with messages addressed to a signal without argument
- date and duration messages support one arg form using 1 as implicit denominator value the one arg form accepts float values (see section 5 p.14).

17.4 Differences to version 0.90
- bug in sync management corrected (introduced with the new sync parsing scheme)

17.5 Differences to version 0.82
- at application level: osc debug is now 'on' by default
- new scripting features (variables) (see section 16.1 p.42).
- ITL file format change:
  - semicolon added at the end of each message
  - '//’ comment not supported any more
  - ‘%’ comment char replaced by ‘!’
  - new variables scripting features
  - single quote support for strings
  - messages addressed to sync node must use the string format
• new ’grid’ object for automatic segmentation and mapping

17.6 Differences to version 0.81

• new Faust plugins for signals processing
• colors management change: all the color models (RGBA and HSBA) accept now float values that are interpreted in the common [-1,1] range. For the hue value, 0 always corresponds to ’red’ whatever the scale used.
• stretch adjustment for video objects (corrects gaps in sync h mode)
• support for opening inscore files on the command line
• system mapping correction
• splash screen and about menu implemented by the viewer

17.7 Differences to version 0.80

• behavior change with synchronization without stretch: now the system looks also in the slave map for a segment corresponding to the master date.
• \$date variable change: the value is now (0,0) when no date is available and \$date is time shifted according to the object date.
• date message change: the date 0 0 is ignored

17.8 Differences to version 0.79

• corrects the map not saved by the save message issue
• corrects get map output: 2D segments were not correctly converted to string

17.9 Differences to version 0.78

• crash bug corrected for the ’save’ message addressed to ’/ITL’
• message policy change: relaxed numeric parameters policy (float are accepted for int and int for float)
• bug in get watch for time events corrected (incorrect reply)

Known issues:

• map not saved by the save message

17.10 Differences to version 0.77

• guido system map extended: supports flat map or subdivided map (see section 11.1 p.27).
• new shear and rotate transformations messages (see section 4.1.4 p.9).
• new rename message to change an object name (and thus its OSC address) (see section 4 p.6).
• relaxed bool parameter policy: objects accept float values for bool parameters
• automatic numbering of exports when destination file is not completely specified i.e. no name, no extension. (see section 4 p.6).
• quantification introduced to \$date variable (see section 15.3 p.39).
• reset message addressed to a scene clears the scene rootPath
17.11 Differences to version 0.76

- `get guidoversion` and `musicxml-version` messages supported by the application (see section 2 p.3).
- `save` message bug correction - introduced with version 0.70: only partial state of objects was saved
- `rootPath` message introduced at scene level (see section 3 p.5).
- Scene name translation strategy change: only the explicit ‘scene’ name is translated by the scene load message handler into the current scene name, other names are left unchanged.
- Bitmap copy adjustment in sync stretched mode is now only made for images

17.12 Differences to version 0.75

- New `require` message supported by the /ITL node (see section 2 p.3).
- New event named `newElement` supported at scene level (see section 15.2 p.39).
- New `name` and `address` variables (see section 15.3 p.39).
- New system map computation making use of the new slices map provided by the guidolib version 1.42
- INScore API: the `newMessage` method sets now the message src IP to localhost With the previous version and the lack of src IP, replies to queries or error messages could be sent to undefined addresses (and mostly lost).
- Bug corrected with ellipse and rect: integer graphic size computation changed to float (prevents objects disappearance with small width or height)
- Bug in scene export: left and right borders could be cut, depending on the scene size corrected by rendering the QGraphicsView container instead the QGraphicsScene
- Crash bug with $date:name corrected: crashed when there is no mapping named `name`.

17.13 Differences to version 0.74

- New `map+` message (see section 9.2 p.23).
- The `click` and `select` messages are deprecated (but still supported). They will be removed in a future version (see section 4.4 p.12).

17.14 Differences to version 0.63

- New `dpage` message accepted by `gmn` objects (see section 8.3 p.20).
- X and y variables: automatic range type detection (int | float)
- `set txt` message: accepts polymorphic stream like parameters (see section 6 p.15).
- Drag and drop files support in INScore viewer
- Interaction variables extension: `$sx`, `$sy` variables added to support scene coordinate space (see section 15.3 p.39).
- Automatic range mapping for `$x`, `$y` variables.
- New `$self` and `$scene` variables in the address field (see section 15.5 p.41).
- OSC identifiers characters set extended with ‘_’ and ‘-‘ (see section 1 p.1).
- Support for multiple scenes: `new`, `del` and `foreground` messages (see section 3 p.5).
- `load` message supported at scene level (see section 3 p.5).
- `get watch` implemented.
- `watch` message without argument to clear all the watched events (see section 15.3 p.39).
- Order of rendering and width, height update corrected (may lead to incorrect rendering)
- Bug with gmn score corrected: missing update for page, columns and rows changes.
• package delivered with the Guido Engine version 1.41 that corrects minimum staves distance and incorrect mapping when optimum page fill is off.

17.15 Differences to version 0.60

• new ‘mousemove’ event (see section 15.1 p.39).
• interaction messages accept variables ($x$, $y$, $date$...) (see section 15.3 p.39).
• SVG code and files support (see section 6 p.16).
• 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.11).
• 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.

17.16 Differences to version 0.55

• new ‘xorigin’ and ‘yorigin’ messages (section 4.1.3 p.8).
• new interaction messages set (section 15 p.38).
• 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 horizontally stretched,

17.17 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 9.1 p.21).
• format of sync messages extended to include map name (section 10 p.24).
• 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.31).
• 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

17.18 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).

17.19 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.12):
  – 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.29).
• mappings can be identified by names (section 9.1 p.21).
• 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.27).
• Guido mapping manages repeat bars.
• Graphic signals messages design (section 13.2 p.34).
Index

ColorMsg, 9
ITLMsg, 3
ITLRequest, 4
OSCAдрес, 1
OSCMesage, 1
PositionMsg, 7
absColorMsg, 9
absPosMsg, 7
blurHint, 11
blurParams, 11
click, 12
clickMode, 12
clickReply, 13
color, 10
colorizeParams, 12
colorvalue, 10
commonMsg, 6
date, 40
debug, 30
effectMsg, 11
faustmessage, 36
faustprocessor, 36
fileWatcher, 29
float2DSegment, 22
floatInterval, 22
fullAddress, 38
getMsg, 18
graphicSignal, 34
hsb, 10
identifier, 1
int1Dsegment, 22
int2Dsegment, 22
intInterval, 22
interactMsg, 38
lang, 43
mapAddMsg, 23
mapMsg, 23
msgVar, 41
originMsg, 8
oscMsg, 38
parallelSignal, 32
penMsg, 19
penstyle, 19
projectionString, 33
rational, 23
relColorMsg, 10
relPosMsg, 8
relation, 21
relativeTimeInterval, 22
relativeTimeSegment, 22
sceneMsg, 5
scoreMapMsg, 27
scoreMsg, 20, 28
scoreQuery, 28
select, 13
selectMode, 13
selectReply, 13
setFile, 16
setMsg, 15, 16
shadowParams, 12
signal, 33, 37
simpleSignal, 31
sync, 24
syncAlignment, 25
syncIdentifier, 24
syncStretch, 25
syncmode, 25
time, 14
timeMsg, 14
timezone, 39
transformMsg, 9
txtStream, 16
variable, 39
variabledecl, 42
whMsg, 19
what, 39
xy, 40

Common messages
angle, 7
click, 12
bottomleft, 13
bottomright, 13
center, 13
topleft, 13

49
topright, 13
color, 10
alpha, 10
blue, 10
brightness, 10
green, 10
hue, 10
red, 10
saturation, 10
det, 6
dx, 8
dy, 8
dz, 8
export, 6
get, 18
hsb, 10
map, 21
map+, 23
mapf, 23
save, 6
scale, 7
select, 13
include, 13
intersect, 13
set, 16
show, 6
x, 7
y, 7
z, 7

Effect messages
effect
blur, 11
colorize, 11
shadow, 11

faustprocessor
in, 36
max, 36
min, 36
out, 36
set, 36

fileWatcher, 29
add, 29
clear, 29
get, 29
remove, 29

Graphic signal
dimension, 35
get, 35
set, 35

Interaction
timezone, 39
doubleClick, 39
mouseDown, 39
mouseEnter, 39
mouseLeave, 39
mouseMove, 39
mouseUp, 39
newElement, 39
timeEnter, 39
timeLeave, 39
variable
abss, 40
abys, 40
address, 40
date, 40
name, 40
sx, 40
sy, 40
x, 40
y, 40
watch, 38
watch+, 38

ITL messages
defaultShow, 3
errport, 3
guido-version, 4
hello, 3
load, 3
mouse, 3
musicxml-version, 4
outport, 3
port, 3
require, 3
rootPath, 3
version, 4

Position messages
absolute
angle, 7
scale, 7
x, 7
y, 7
z, 7
color
dalpha, 11
dblue, 11
dbrightness, 11
dcolor, 11
dgreen, 11
dhsb, 11
dhue, 11
dred, 11
dsaturation, 11
relative
dangle, 8
dscale, 8
dx, 8
dxorigin, 8
dy, 8
dyorigin, 8
dz, 8
xorigin, 8
yorigin, 8
Scene messages, 5
del, 5
foreground, 5
fullscreen, 5
load, 5
new, 5
reset, 5
rootPath, 5
Scripting
language, 43
variable, 42
Set type
curve, 16
ellipse, 16
file, 17
gmn, 16
gmnf, 17
graph, 16
html, 16
htmlf, 17
img, 17
line, 16
polygon, 16
rect, 16
svg, 16, 17
txt, 16
txf, 17
video, 17
signal, 32
parallel signal, 33
get, 33
list, 33
projection string, 33
simple signal
default, 32
del, 32
get, 32
reset, 32
size, 32
Specific messages
height, 20
penColor, 19
penStyle, 19
dash, 19
dashDot, 19
dashDotDot, 19
dot, 19
solid, 19
penWidth, 19
score
columns, 20
dpage, 20
page, 20
dCount, 20
pageFormat, 20
rows, 20
width, 20
Synchronization, 24
syncIdentifier, 24
get, 24
Guido map, 27
page, 27
staff, 27
system, 27
systemflat, 27
voice, 27
syncAlignment, 25
cBottom, 25
cOver, 25
cTop, 25
cmode, 25
cStretch, 25
h, 25
hv, 25
v, 25
Time messages
absolute
date, 14
duration, 14
relative
clock, 14
ddate, 14
dduration, 14
durClock, 14
Transform messages
rotate, 9
shear, 9