Interactive Sonification Markup Language (Ishmael) Manual

James Walker

This user guide was written for version 0.5.2 of the software. If you are using a different version of the software, there may be discrepancies with this user guide.

Table of Contents

1. Introduction

Interactive Sonification Markup Language (ISML, pronounced “Ishmael”) is a scripting language for writing custom sonification configurations for the IVS interactive sonification system. The interactive sonification system plays music and sound that change dynamically in response to the movements of the user. With ISML, it is possible to precisely control the process by which the system changes audio output in response to user movements. The ISML Generator is a graphical user interface (GUI) designed to allow easy production of ISML scripts. This guide explains how to use the ISML Generator and includes an explanation of the ISML specification.

2. Launching the ISML Generator

The ISML Generator is a completely browser-based application that can be launched from any web browser. The application can be accessed online at the following URL:
http://cs.mtu.edu/~jwwalker/isml-creator/ISML-Creator.html

Alternatively, if you have access to the source files, you can launch the application from your own computer by launching the file “ISML-Creator.html” in your web browser. Note that the application requires all of the following files to be in the same folder in order to work correctly:

ISML-Creator.html
generator.js
style.css

3. How ISML Works

The interactive sonification system dynamically changes sound output via the following procedure. First, the system checks to see whether a set of *conditions* has been met; for example, “The user is currently moving at a speed equal to or greater than 1 meter per second.” If these conditions are met, the system executes one or more *actions;* for example, “Change the key signature to C-major and the time signature to 4/4.” Conditions are optional: It is allowed to specify a set of actions that is executed all the time, without any conditions being met.

Each set of conditions and actions is organized into an *activity.* Activities are simply a mechanism for grouping conditions and actions together. By having multiple activities, it is possible to have different sets of conditions and actions which can be checked and executed. Within an activity, the conditions must be satisfied in order for the actions to be executed; outside of that activity, its conditions do not matter.

Lastly, activities are organized into *items.* Items provide a scoping mechanism for *variables.* A variable is simply a placeholder for a value; or a name attached to a value. For example, if I specify “x = 5”, I have assigned the value 5 to a variable called x. If I later specify “x = x + 4”, then I take the value inside x (5), add 4 to it for a total of 9, and then assign the 9 back into x. Thus x now has a value of 9. You have 26 variables available to you (a-z) for use in ISML scripting. However, each variable exists independently within the item in which it is used. For example, if I have 2 items, Item A and Item B, and I have a variable x in both items, I can assign x = 5 in Item A, and x's value in Item B is unchanged. Therefore, you are limited to 26 variables per item, but as you can have a theoretically unlimited number of items, you also have an unlimited number of variables for use. This is what is meant by the term “scoping mechanism.”

Note that, within an item, all of that item's activities are executed in *sequential order* from top to bottom. This is important since the execution order must be clearly defined in order for the values of variables and the results of executing the script to be deterministic (predictable).

Each item, and all of its activities, is executed (all conditions are checked and the corresponding actions are executed) every time the interactive sonification system cycles. By default, the system cycles at a rate of (xx-not yet defined as of this writing) times per second.

Concept Review: ISML scripts have *conditions* which must be true and a set of *actions* which are executed if the relevant conditions are met. Conditions are optional; you may specify actions which are always executed unconditionally. Each set of conditions and actions is organized into an *activity.* Activities are likewise organized into *items.* You have 26 *variables* available for use, a-z, for each item. Variables associate a name with a value. The values of the variables in one item do not affect the values of the variables in other items. Within each item, its activities are executed sequentially from top to bottom.

4. Using the ISML Generator

4.1. Editing the Script Structure

When you first start the ISML Generator, you will see a screen similar to the following:

Figure 1. The ISML Generator startup screen.

You can click on the “Help Guide” link to view the application's documentation (which you are currently reading). The “Load script” button is used to load preexisting ISML scripts for editing. This functionality is explained in full in section 4.5.

Clicking the “Add Item” button inserts a new item into your script for editing, as shown in figure 2. You may add as many items as desired to your script. You can press the “Move Up” and “Move Down” links to move each item up and down in the list. This functionality is provided only as a convenience, as the order of items does not matter in an ISML script.

Figure 2. Adding a new item to the script.

Once you have added at least one item to your script, the option to “Download ISML File” appears at the bottom of the script. This functionality is explained in full in section 4.4.

You can delete an item by clicking the “Delete Item” button. In order to prevent you from accidentally deleting items, you must first check the box next to the button before clicking it, or the delete operation will not be executed.

Within each item, you can add any number of activities by clicking the “Add Activity” button, as shown in figure 3.

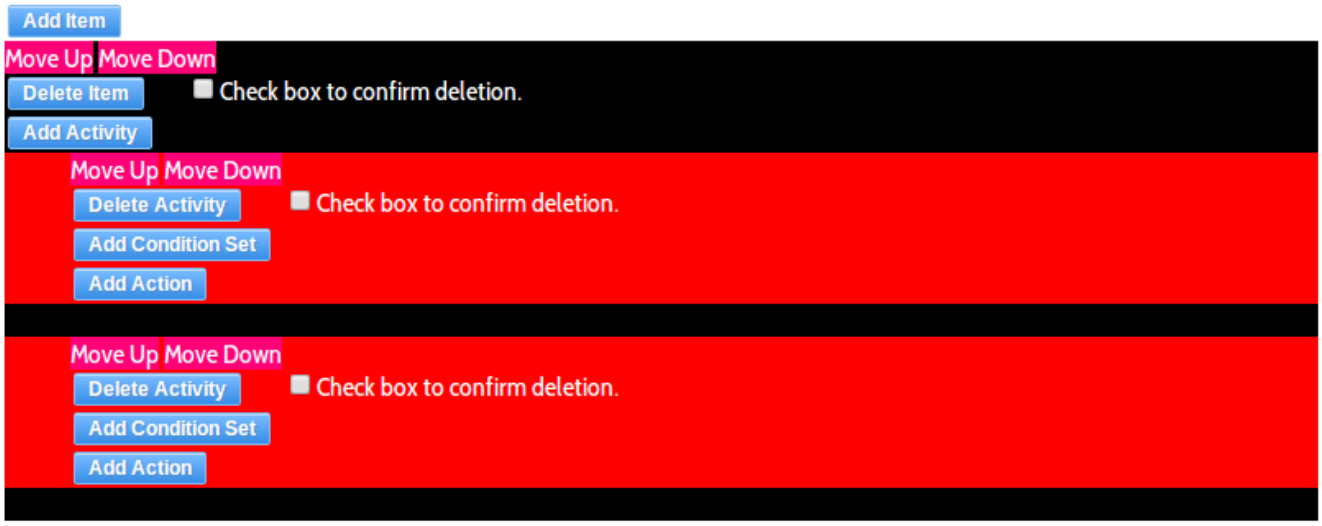

Figure 3. The result of adding 2 activities to an item.

As with items, you can delete activities (after checking the confirmation box), and move them up and down. Unlike items, moving activities up and down is not just a matter of user convenience, because activities are executed in top-down order, so their order matters.

Note that activities have a different background color, and their contents are offset from the left side of the screen. This indentation and color scheming is intended to help you keep track of the structure of your script, so that you can easily tell which conditions belong to which condition sets; which condition sets and actions belong to which activity; and which activities belong to which items.

Within each activity, you can add condition sets and actions, as shown in figure 4.

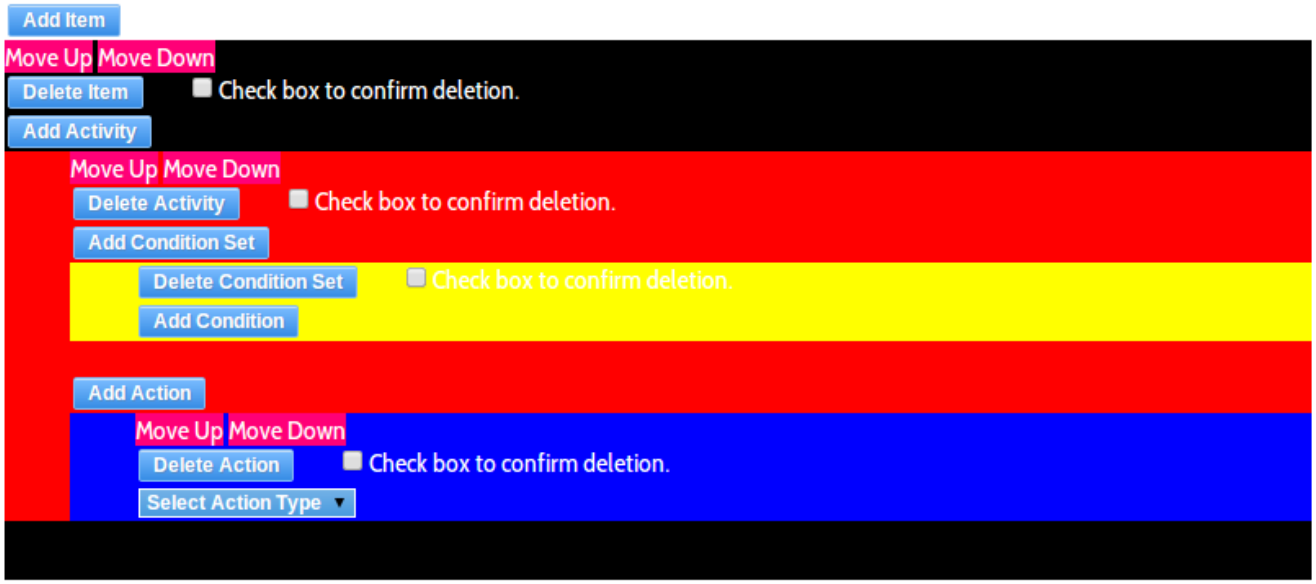

Figure 4. The result of adding one condition set and one action to an activity.

Both condition sets and actions can be deleted as normal. Actions can be moved up and down. As with activities, the order of conditions matter, as they are executed in top-down order. Editing actions is discussed in full in section 4.3.

Each condition set can have any number of conditions added to it, as shown in figure 5.

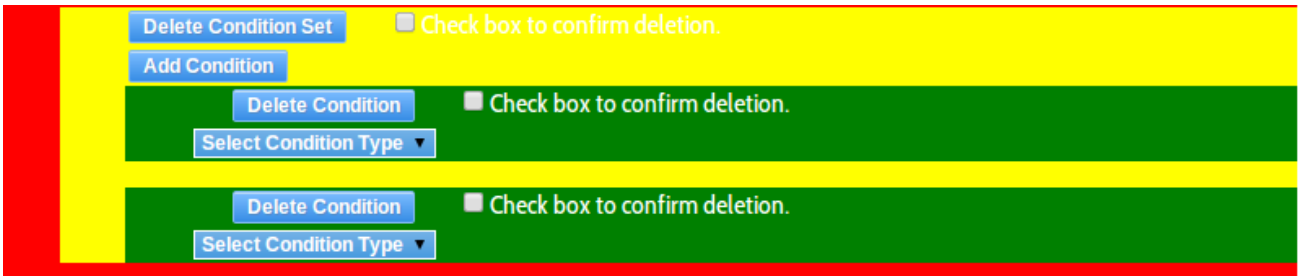

Figure 5. A condition set with 2 conditions added to it.

A condition set is a container for any number of conditions. The rules for conditions and condition sets are as follows:

- *All* conditions within a condition set must be true for that condition set to evaluate as true.
- If *any* condition set evaluates as true, the actions within that activity are executed.
- In other words, if you want the actions to execute only if *both* condition A *and* condition B are true, then put conditions A and B in the *same* condition set. If you want the actions to execute if *either* conditions (A, B, etc.) are true *or* conditions (X, Y, etc.) are true, then put the corresponding conditions in *different* condition sets.
- For those users with a background in formal logic, each condition set is a clause of literals ANDed together, and all of the condition sets are ORed together; i.e.,

  $(x_1 \text{ AND } x_2 \text{ AND } \ldots x_n) \text{ OR } (y_1 \text{ AND } y_2 \text{ AND } \ldots y_n) \text{ OR } \ldots$

With that understanding of how condition sets and conditions work, you should be able to structure the conditions of your scripts properly.

Editing conditions is explained in full in the next section.

4.2. Editing Conditions

When you first add a condition, you have the option to delete the condition as normal, and there is also a “Select Condition” dropdown box. Click this box to bring up a list of condition types. As of this writing, there are only 2 types of conditions: Object and Comparison.

Object

When you select “Object,” The dropdown box changes to “Select Parameter Value.” Currently there are 5 objects that can be tracked: the user's right hand, left hand, right foot, left foot, or head. You may also specify that the condition is valid for either foot, either hand, both feet, or both hands.

This condition only makes sense when used in conjunction with other conditions. The meaning of this

condition is that it only checks the other conditions for the specified object. If no object is specified, the conditions are checked based (as applicable) on the average values for all tracked objects. For example, if you check to see whether x_velocity is over a certain value, then specifying “Object is of type right_foot” only checks the right foot's x_velocity. If you specify no object, the average x_velocity of all tracked objects will be checked.

Note that if you specify two contradictory objects in the same condition set (for example, left hand and right foot), the condition can obviously never evaluate as true (an object cannot simultaneously be a left hand and a right foot), so the corresponding actions will never be executed.

Comparison

Comparisons are of the form `value_1 comparator value_2`, where value_1 and value_2 can be many types of values, and comparator can be >, <, =, or != (greater than, less than, equal to, or not equal to).

Therefore, when you select “Comparison,” three dropdown boxes are created, 2 for selecting values to check and 1 for selecting the comparator. There are many kinds of possible values that you can check, which are explained below:

- Number: This is used when you want to compare a raw numeric value. When you select this option, a box appears that allows you to enter a number. You may enter any valid whole or decimal value in this box, then click “Set value” to enter that as the number to be compared.
- Variable: This is used when you want to compare the value being stored in a variable. By default, all variables begin with a value of 0 until that value is changed through assignment (which occurs in actions). When you select this option, the dropdown box is replaced with “Select Variable” which presents you with 26 variables (a-z) to choose from.
- bpm: Beats Per Minute the music is currently playing at.
- cur_velocity_x: The current velocity in the X dimension, in meters per second.
- cur_velocity_y: The current velocity in the Y dimension.
- cur_velocity_z: The current velocity in the Z dimension.
- cur_velocity_composite: The current velocity averaged from all 3 dimensions.
- avg_velocity_x: The average velocity over the lifetime of the tracked object(s) in the X dimension. By default, the system keeps 5 seconds worth of data, so this will return the object(s) average velocity from the last 5 seconds.
- avg_velocity_y: The average velocity over the lifetime of the tracked object(s) in the Y dimension.
- avg_velocity_z: The average velocity over the lifetime of the tracked object(s) in the Z dimension.
- avg_velocity_composite: The average velocity over the lifetime of the tracked object(s) averaged from all 3 dimensions.
- acceleration_x: The current acceleration in the X dimension, in meters per second per second.
- acceleration_y: The current acceleration in the Y dimension.
- acceleration_z: The current acceleration in the Z dimension.
- acceleration_composite: The current acceleration averaged from all 3 dimensions.

- avg_proximity: The average distance, in meters, between all of the tracked objects.
- x_position: The current X position, in meters.
- y_position: The current Y position.
- z_position: The current Z position.
- elapsed_time_once: The amount of time that has passed since the program started, in milliseconds (there are 1000 milliseconds in 1 second). This condition will only evaluate to true the first time it is met, and after that it will always evaluate to false.
- elapsed_time_repeatable: The amount of time that has passed since the program started, in milliseconds. This condition can be used to set up a condition that occurs repeatedly at regular time intervals by using the remainder operator. For example, the condition `elapsed_time_repeatable % 5000` will evaluate to true once every 5 seconds.

"Select Comparison" presents you with 4 options: equal to, not equal to, greater than, and less than.

4.3. Editing Actions

When you first add an action, you have the option to delete the action as normal, and there is also a "Select Action" dropdown box. Click this box to bring up a list of action types. As of this writing, there are 5 types of conditions: Assignment, Set Key Signature, Set Time Signature, Set Instruments, and Play Notes.

Assignment

Assignments are of the form `value_1 = value_2 operator value_3` or simply `value_1 = value_2`. When executing this kind of action, the system first evaluates the result of applying the operator to values 2 and 3. It then takes this result and assigns it to value_1. For example, `x = y + 5` takes the value in y, adds 5 to it, then assigns the result to x (so if y contained the value 12, x would contain the value 17 after this action was executed). If the operator and value_3 are not used, then the system simply assigns the value in value_2 to value_1.

Therefore, when you select "Assignment," three dropdown boxes are created, 3 for selecting values and 1 for selecting the operator. All of the same values that were described for conditions are also available for actions (numbers, variables, and many others).

The ISML Generator currently supports 7 types of operators, described below:

+ Add value_2 and value_3.
- Subtract value_3 from value_2.
* Multiply value_2 and value_3.
/ Divide value_2 by value_3.
% The remainder operator. Divides value_2 by value_3 and returns the *remainder* of the division.
^ Raise value_2 to the power of value_3.
abs Get the absolute value of value_3.

It is possible to construct partial assignments by not selecting options for the unneeded components. For example, to create the action `bpm = 120`, select "bpm" for the first value, select "Number" and

enter “120” for the second value, and do not assign an operator or a third value. To create the action `x = abs y` (assign the absolute value of y to x), select “Variable” then “x” for the first value, “abs” for the operator, and “Variable” then “y” for the third value (leave the second value unselected).

Set Key Signature
When you select this option, the dropdown box changes to “Select Key Signature,” and you may then choose from the list of provided key signatures.

Set Time Signature
When you select this option, the dropdown box changes to “Select Time Signature,” and you may then choose from the list of provided time signatures.

Set Instruments
Select this option to change the instruments used by the system. Instruments are categorized by musical genre. At present, 3 genres are supported: electronica, rock, and orchestral.

Play Notes
When you select this option, a text box appears, allowing you to enter a series of notes to be played. These notes must conform to the following format: note duration (1 = whole, 2 = half, 4 = quarter, 8 = eight, etc.), position on the staff (A#, B, C, C#, D, D#, E, F, F#, G, G#), octave offset from center C (-1 is one octave down, +2 is two octaves up, etc.). Leave the octave field blank if there is no offset. Notes are separated by commas. For example,

```
8c#+1,4a,2d-1
```

would play a C# eighth note one octave up, an A quarter note at the default octave, and a D half note one octave down.

When you are finished entering notes, click the “Enter Value” button. The entered notes will be played by the lead instrument from the selected instrument set.

4.4. Saving Your Script

When you script is complete, you can save it by clicking “Download ISML File” at the bottom of the page. Depending on your browser settings, this may display the contents of the script in a new page. To *save* the script file instead of viewing it, right-click on the “Download ISML File” link and select “Save Link As” (or your browser's equivalent option). Then give the file an appropriate name, navigate to the directory where you wish to save it, and click the “Save” button.

4.5. Loading Scripts

Due to the limitations of Javascript, it is impossible to load files directly. Instead, locate the script that you want to load, open that script in a plain text editor (such as Notepad or Gedit), select all of the text inside the script file, and copy it. Then launch the ISML Generator, click inside the text box next to the “Load script” button, and select paste. Then check the confirmation box and click the “Load script” button to load the script for editing.

5. Advanced ISML Scripting / Developer Guide

If you have some experience programming or writing script files by hand, you may find it more efficient to write ISML files by hand rather than using the ISML Generator. For users who wish to give this a try, here is a quick guide to writing ISML scripts by hand. Developers who wish to write a parser for ISML scripts should also find this section useful.

ISML uses a format similar to XML or HTML. Sections are denoted by tags in angle brackets (e.g., `<tag>`) and are terminated by the same tag, except with a slash in front of the text (e.g., `</tag>`). Following is a complete list of all the supported tags in ISML:

```
<item> </item>
<activity> </activity>
<if> <or> <then> </if>
```

Everything within an item should be contained between corresponding `<item>` and `</item>` tags. The execution order of items is not defined; thus, items should be written to be independent of one another. Everything within an activity should be contained between corresponding `<activity>` and `</activity>` tags. Every activity within an item is checked and executed in sequential order. The content of each activity is in the following form:

```
<if>
condition_1
condition_2
…
<or>
condition_1
condition_2
…
<or>
…
<then>
action_1
action_2
…
</if>
```

Or alternatively, simply:

```
action_1
action_2
…
```

Each condition and each action is contained on a single line. In order for an activity's conditions to evaluate as true, *every* condition in *any* condition set must evaluate as true. If this is the case, all of the associated actions are then executed in sequential order. A condition set has any of the following forms:

```
<if>
conditions
<then>
```

```
<if>
conditions
<or>

<or>
conditions
<or>

<or>
conditions
<then>
```

Note that having conditions in an activity at all is optional. An activity may contain only actions, in which case they are always executed.

There are 26 available variables, from a to z. Variables have *item scope*, meaning two variables with the same name in two different items are considered to be different variables. Variables are assumed to contain numeric values. These numeric values may contain a decimal component.

Conditions may have either of the following 2 forms:

```
object_type
comparator value_1 value_2
```

Note that in the actual ISML script, the comparator comes *before* the 2 values being compared instead of between them; e.g., `less_than cur_velocity_x 1.0` instead of `cur_velocity_x less_than 1.0`. This is known as *prefix notation*, contrasted with *infix notation*.

Valid object types are the following:

```
left_hand right_hand either_hand both_hands
left_foot right_foot either_foot both_feet
head
```

Valid comparator tokens are:

```
equal_to not_equal_to greater_than less_than
```

Values may be any valid number; a variable name (a-z), or any token from the following list:

```
cur_velocity_x cur_velocity_y cur_velocity_z cur_velocity_composite
avg_velocity_x avg_velocity_y avg_velocity_z avg_velocity_composite
acceleration_x acceleration_y acceleration_z acceleration_composite
avg_proximity x_position y_position z_position elapsed_time_once
elapsed_time_repeatable bpm
```

Actions may have any of the following forms:

```
assignment
set key signature
```

```
set time signature
set instruments
play notes
```

Set key signature is denoted by any token from the following list:

```
c_major g_major d_major a_major e_major b_major fsharp_major csharp_major
a_minor e_minor b_minor fsharp_minor csharp_minor gsharp_minor dsharp_minor
asharp_minor
```

Set time signature is denoted by any token from the following list:

```
1/2 2/2 3/2 4/2 2/4 3/4 4/4 2/8 3/8 4/8 6/8 8/8
```

Set instruments is denoted by any token from the following list:

```
electronica rock orchestral
```

Assignments take the following form:

```
= value_1 operator value_2 <value_3>
```

The third value is optional. In the case of simply transferring one value to another, `assign` is used as the operator; e.g.,

```
= a assign cur_velocity_x
```

Values 2 and 3 may be any valid number; a variable name (a-z), or any token from the token list for conditions (`cur_velocity_x` and so on). Value 1 may only be a variable or bpm because it does not make sense to assign to other kinds of properties, such as velocity, which are read-only.

Valid operators include:

```
+ - * / % ^ abs
```

“Play notes” has the following form:

```
play_notes:note_list
```

where note_list must follow the format described in section 3.2, under “Play Notes.”

Example Script

The following example script demonstrates what raw ISML script looks like.

```
<item>
<activity>
<if>
equal_to elapsed_time_once 1000
<then>
rock
c_major
4/4
= bpm assign 120
</if>
</activity>
<activity>
<if>
greater_than cur_velocity_composite 4
<or>
greater_than acceleration_composite 5
<then>
electronica
= bpm assign 180
= a + b c
= d abs e
</if>
</activity>
</item>
```

This rather simplistic script has a single item which contains 2 activities. The first activity sets the instrument set to rock, the time signature to 4/4, the key signature to C major, and the BPM to 120 exactly 1 second into the program running. The second activity checks to see if the user is moving faster than 4 meters per second or accelerating more than 5 meters per second squared. If either condition is met, the instrument set is set to electronica and the BPM is increased to 180. The script then performs some simple arithmetic operations on some variables, purely to demonstrate the format, since in this case these operations accomplish nothing useful.