iisuOscBridge

Quick Start Guide

February 2012

iisuOscBridge allows to use iisu without requiring complex programming. This is made possible by streaming the iisu data via the OSC network protocol.

**Antoine Rennuit**

Table of Contents

[1. Introduction 1](#_Toc316277114)

[2. Principle & architecture 1](#_Toc316277115)

[3. Basic use case 2](#_Toc316277116)

[3.1. Interaction designer scripts 4](#_Toc316277117)

[4. Reading the data 5](#_Toc316277118)

[4.1. Composite types 5](#_Toc316277119)

[4.2. Arrays 6](#_Toc316277120)

[4.3. Arrays of composite types 6](#_Toc316277121)

[5. Conclusion 7](#_Toc316277122)

1. Introduction

iisuOscBridge is a tool that comes on top of iisu. It provides OSC streaming for most types handled by iisu. That is it is able to read most data types handled by the iisu engine directly and send it over a network via the OSC protocol.

iisuOscBridge is designed as an end application for iisu. This means it uses iisu internally and the users do not have to manipulate the iisu SDK directly in order to get access to the 3D markerless motion capture engine. Put another way: with iisuOscBridge there is no need for complex programming in order to use iisu. The data from iisu is available via an OSC stream and you can use it as you would use any other OSC stream.

Note that **iisuOscBridge 1.0 or newer versions only works with iisu 3.0 or newer versions**.

Also, currently it **only works under windows**. There may be a linux version at some point in the future, if the need is raised but there will probably not be a mac version. This is because iisuOscBridge uses the iisu dlls and there is no mac version of iisu at the time when this document was written.

This software comes with no guarantee. Use it at your own risk.

1. Principle & architecture

Many different data come out of iisu. There is so much information that it would not make sense to stream everything out and it would be totally inefficient. So we made it **possible to tailor what is streamed out** in order to fit users’ specific needs.

In OSC, the information is packed into bundles. Each bundle is composed of several messages and each message carries an atomic piece of information.

At each time tick of iisu (that is each time iisu computes new data), an OSC bundle is computed with data registered in iisuOscBridge. Using an OSC bundle makes it clear that **all the data in the bundle originates from the same time tick** in iisu. This is because a unique timestamp is associated to each OSC bundle and thus all the data inside of a bundle has the same timestamp.

As mentioned earlier a bundle is composed of several messages. **In the stream of a session all bundles have the same structure.** That is the bundles are composed of the same message types. Bundles can be edited for the need of other sessions though.

Note that OSC messages are identified by a path which is organized in a tree. That is a path is composed of “path bits” which work like folders and sub-folders (which we will also call “structuring” bits). Each bit can contain data or sub-bits in the same way that folders can contain files or sub-folders. This will become clear in the next section. Examples of OSC path:

*The data 1 is directly at the root.*

*/dataPathBit1*

*/pathBit/subPathBit1/subSubPathBit/dataPathBit2*

*Both data 2 & 3 are in the same folder.*

*/pathBit/subPathBit1/subSubPathBit/dataPathBit3*

*Data 4 is in a different sub-folder.*

*/pathBit/subPathBit2/dataPathBit4*

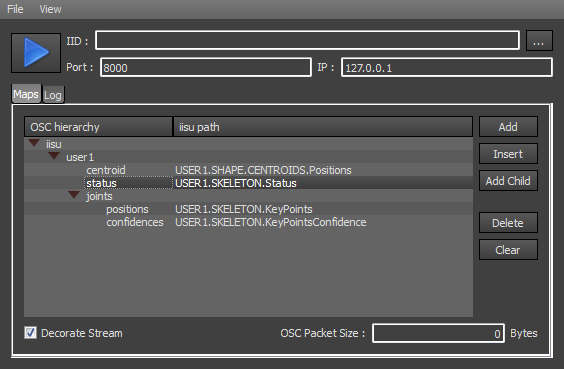
*Example of OSC paths.*

*Paths are composed of “bits” which can be seen as names for folders, sub-folders or data (data bits are underlined).*

Bundles in iisuOscBridge are setup in the main window (the *Maps* tab). They can be saved together with other information for later reuse.

1. Basic use case

The iisuOscBridge application looks like what is shown on the image below. There are not many windows and it will be obvious to most users to perform the basic operations.



*Maps tab of the iisuOscBridge application.This is the central tab or window to set up a bundle of messages and start streaming data.*

After first installing the application, the *Maps* tab (shown on the above image) will be empty – unlike what is shown here, where the maps for a bundle of messages are already in place. Then on later openings of the application the last opened project will be automatically loaded for you. Of course, you can always reset to a new project when needed.

**The user is due to setup his bundle of messages as a set of maps between OSC message paths and iisu data paths** (we will explain how in a following paragraph) or modify an existing one. **A map is the association of an OSC path together with and iisu data path.** This map or association is very important. It makes it possible to identify an iisu data within an OSC stream: each iisu data streams in an OSC message which path is provided by the map. In the *Maps* tab each map is represented by a line in the tree structure.

The above image shows seven line, four of which are associated an iisu data path. This means four iisu data are due to stream in this project. The three remaining lines are only there to structure the OSC messages in a folders-like hierarchy. These structuring elements could also be associated data (ie an iisu data path) to stream but most often it is cleaner not to use this possibility. If we keep to this convention, the analogy is that the structuring elements or structuring maps can be looked at as folders in a file system and the data maps can be seen as files.

The hierarchy of maps and OSC messages in the bundle is created using the buttons on the right of the *Maps* tab and that appear on the following image.



*Buttons allowing the creation of new maps (structuring maps or data maps, associating an OSC path to an iisu data message).*

The *Add* button adds a map at the bottom of the list of siblings of the selected map, the *Insert* button allows inserting a map just before the selected one and the *Add Child* button allows giving the OSC messages its hierarchical structure as it creates a child to the selected map.

You may be surprised that the *OSC hierarchy* column only shows OSC path bits when you may expect a full OSC path. This is because the **OSC path of a map inherits the path bits of its parents**. For example, the map /*positions* in the tree corresponds to an OSC message which full path is actually:

*/iisu/user1/joints/position*  and not /*position* only.

This is this very same full path which will identify the data in OSC. This is because */iisu*, */user1* and */joints* are structuring parent maps of map */position*.

**Once you have added a new map using the buttons mentioned above, you can associate it with iisu data**. This is done by specifying the iisu data path in the right column. You can type the iisu data path if you know it by heart but most often you will select it in the drop down menu that pops up when you double click on the map in the column. Each piece of data registered in iisu is listed in this menu.

Before beginning the stream you need to check the IP address and port are correct. The default IP address *127.0.0.1* corresponds to the localhost, that is you stream data to your own machine. The default IP port *8000* corresponds to the default IP port for OSC as mentioned in the standards of OSC.

You then only have to **press the big blue *play* toggle button in order to start streaming the data**. The tabs view changes to *log* which makes it easier to understand what is going on. The *play* button toggles to a *stop* button when pressed. This is obviously the button to use in order to stop the stream.

## Interaction designer scripts

Interaction designer (or IID) is an application which makes it able to script behaviors in a language call LUA, get of graphical/visual display for feedback and output the result of these computations as normal iisu data that you can read as any other iisu data. This **allows tailoring the data coming out of iisu to fit the user’s need**. A basic example of use would be to use the data coming out of iisu to build a natural interface slider, to control say… sound.

Let us imagine you actually control the sound volume. You want no sound when hands are joined and full sound power when hands are extended. Programmatically we assume this would translate into a variable called *SoundPower* being 0 for no sound and 1 for full power.

Then we can use IID to get the position of the hands, compute the distance between them and rescale the value in an output data variable called *SoundPower* so that when hands are joined *SoundPower* is null and when hands are extended, *SoundPower* is unitary.

The live value of *SoundPower* is automatically registered for you in iisuOscBridge and can be selected for streaming as any other data coming out of iisu. This makes the full pipeline easily customizable.

As in any iisu session, **iisu has to know about the script if you want it to be processed**. In iisuOscBridge this is done by specifying the script’s file path in field *IID* next to *start*/*stop* toggle button, as seen on the image below.



*Field IID, used to specify the file path of the IID script we want iisu to process. At the right of the field there is a square button which displays a file selection dialog box to automatically fill the field.*

Specifying the file path by hand would be tedious. There is a square button which shows three points (“…”) at the right of the *IID* field and that displays a dialog box in which you can select the script file you like. It fills the *IID* field accordingly.

**When a script file is selected, the variable data it outputs is added to the iisu data available for streaming**. You can then associate these iisu data paths added by IID to an OSC path in a map as you would do with any other normal iisu data.

1. Reading the data

When streaming simple, non-composite data, like a float value, an integer, a string or whatever getting the data is very straightforward and is done the usual way applications use OSC streams. We are not going to get into the details of that because depending on the client platform things work much differently. Look at our examples or the examples of the client application you use for more details.

For simple data the overall idea though is that if you know the OSC path associated to a message then there is a way you can get to know the live value of data associated. The data is available out of the box with no trick.

This is not the case anymore when the data becomes more complex. That is **when the data is composite and has several components you need to know about the conventions we used in order to read the data**. Think about a vector with component x, y, z for example, or about an array of size *n*, or even worse: an array of vectors.

In this case we decided to implement two different modes in order to stream the data. In the first case, called the *Decorated Mode* (**the *Decorated Mode* is the default mode**), the data is simple to read and use. In the other case, called the *Un-Decorated Mode*, the stream is more compact but the downside is that the data is more complex to read.

You can toggle from one mode to the other using the *Decorate Stream* check box at the bottom of the *Maps* tab.



*The* Decorate Stream *check box at the bottom left of the* Maps *tab.*

In the *Un-Decorated Mode* the bundle is about five times more compact than in the other mode. Which means you can stream more data on the same network.

**The difference between the two modes is about how the data is arranged in the bundle and how the messages’ file path is built.**

## Composite types

In the *Decorated Mode*, the bundles are much more numerous because the composite data (vector and arrays) are split. That is **in the *Decorated Mode* there is one message per component of the data**. Eg, a *Vector3* which OSC path is */iisu/myVector3* actually streams as three expanded or decorated OSC messages namely:

*/iisu/myVector3****/x***

*decoration of the OSC path for a Vector3:*

* *the addition bit is marked as bold.*

*/iisu/myVector3****/y***

*/iisu/myVector3****/z***

The same kind of decoration occurs for *Quaternions*, *Frames*…

**In the un-decorated mode and for the same *Vector3*, the three components would stream in the same message**, as OSC allows.

## Arrays

For arrays the principle is roughly the same. In the *Decorated Mode* each single component of the array streams in a message of its own. Note though that **only the arrays which have the same size as the number of keypoints in iisu (21 in iisu 3.0) can stream decorated** this is because the decoration corresponds to the labels associated to the keypoints. The labels used are the public keypoints labels in iisu: *Pelvis*, *Waist*, *LeftShoulder*...

Eg, an array of floats which OSC path is */iisu/myFloatArray* actually streams as twenty one expanded or decorated OSC messages namely:

*/iisu/myFloatArray****/Pelvis***

*/iisu/myFloatArray****/Waist***

*decoration of the OSC path for an array of floats.*

*…*

*/iisu/myFloatArray****/LeftFoot***

The order of the components in the array is the order of the joints in the iisu API. This order is reminded in the table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | *Pelvis* | 7 | *Right Wrist* | 14 | *Left Elbow* |
| 1 | *Waist* | 8 | *Right Hand* | 15 | *Left Wrist* |
| 2 | *Collar* | 9 | *Right Hip* | 16 | *Left Hand* |
| 3 | *Neck* | 10 | *Right Knee* | 17 | *Left Hip* |
| 4 | *Head* | 11 | *Right Ankle* | 18 | *Left Knee* |
| 5 | *Right Shoulder* | 12 | *Right Foot* | 19 | *Left Wrist* |
| 6 | *Right Elbow* | 13 | *Left Shoulder* | 20 | *Left Hand* |

*Order of the joints in the streams of arrays.*

In the case the array streams in the *Un-Decorated Mode* (that is the *Decorated Mode* is off or the array size is not twenty one), then all the components of the array would stream in the same message as OSC allows.

Note that **in the *Un-Decorated Mode*, the size of the array is available in the first component of the message**. Hence an array of *n* floats actually streams in a message of *n + 1 components* with the first component being the size.

## Arrays of composite types

**In the case of arrays of types which are composites such as *Vector3*, and in the *Decorated Mode*, both the decorations for the array and the decoration for the types are used in a row**. Eg, an array of Vector3 which OSC path is */iisu/myVector3Array* actually streams as sixty three expanded or decorated OSC messages (21 messages for the array components multiplied by 3 components per *Vector3*) namely:

*/iisu/ myVector3Array* ***/Pelvis/x***

*/iisu/ myVector3Array* ***/Pelvis/y***

*/iisu/ myVector3Array* ***/Pelvis/z***

*decoration of the OSC path for an array of Vector3.*

*/iisu/ myVector3Array* ***/Waist/x***

*…*

*/iisu/ myVector3Array* ***/LeftFoot/y***

*/iisu/ myVector3Array* ***/LeftFoot/z***

In the case the array streams in the *Un-Decorated Mode* (that is the *Decorated Mode* is off or the array size is not twenty one), then all the components of the array would stream in the same message as OSC allows and in the same order. The first component in the message would be the size of the array, ie **the number of elements in the array, not the overall number of components**.

That is for an array of 30 *Vector3*, the size is 30, not 90. This message would look something like this:

*30, value of component x for the 1st Vector3, value of component y for the 1st Vector3, value of component z for the 1st Vector3, Value of component x for the 2nd Vector3,…,Value of component z for the 30th Vector3.*

1. Conclusion

Although coded by a member of SoftKinetic, this project was done in my spare time and is not part of the SoftKinetic products.

The responsibility of SoftKinetic is not involved at all as for any idea in this document or in iisuOscBridge itself.

The ideas or positions expressed in this document only reflect my own state of mind and not SoftKinetic’s opinion.