RSVP, a preset system solution for Pure Data

José Rafael SUBIA VALDEZ
Edinburgh University
Alison House
Edinburgh
EH8 9DF
Scotland
Rafael.Subia@ed.ac.uk

Abstract

This paper describes the logic and process behind the development of the RSVP preset library for the Pure Data programming environment. The library aims to tackle the lack of a native preset system in Pure Data. Projects like Kollabs[1], CREAM[2], sssad[3] and others, have produced different solutions for this issue. However, after experimenting with these, it became clear that a different approach was required to fit personal needs. This led to the creation of the RSVP library which will be described in detail. During the development of this project, a feature request for PD was identified, and that will also be shared here. This paper will offer a detailed description of how the system works, but will not go into extensive Pure Data patch descriptions. Instead it will focus on how the code is structured and will describe how the system functions with the users' own projects.

Keywords

state-saving, GUI, interpolation, external, abstraction

1 Introduction

The flexibility that Pure Data [Puckette, 1996] has as a programming environment is immense; the fact that a *Graphical User Interface* or "GUI" is part off its workflow concept is very interesting as a programming language. Pure Data, and its "prettier sibling", MAX [Zicarelli, 1990], allow users to program in a different style than Supercollider [McCartney, 1996] or ChucK [Wang and Cook, 2002] to name a few. PD incorporates the idea of "connection" that is well known among musicians with stage experience. However, it does not contain an easy and rapid preset mechanism such as the one found in MAX. Having to tackle this issue led to the development of other ways of interacting with a patch. Consequently, this required the use of some interesting tricks to overcome the lack of this particular feature. Nevertheless, a preset system is very helpful for musical purposes even if not used extensively.

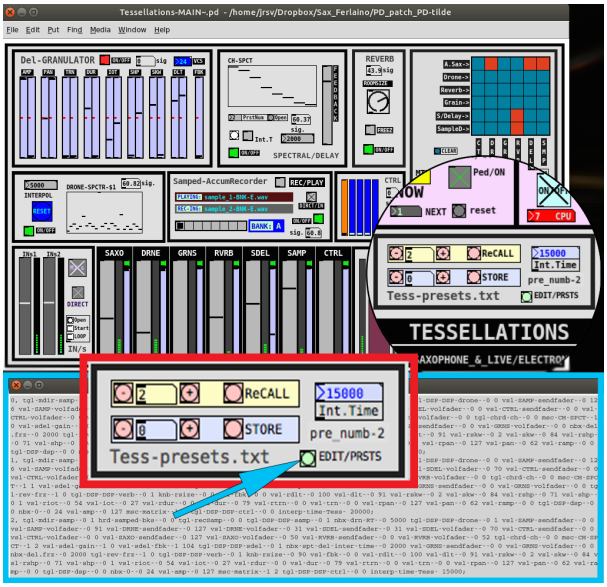

Figure 1: Patch of *Tessellations* for alto sax and computer, developed with RSVP

Over the years, different techniques implemented by other users were tested and incorporated in complex patches produced for personal use. There are some complete and powerful libraries like Kollabs [Weger, 2014], that allow different types of interpolation between current and to-recall values. The CREAM [Guillot, 2014] library and its GUI programmed as *externals*, also offers interesting interpolations, even working with its *c.breakpoints*[4] object. The design is very similar to the one in MAX, including the commands on the *c.preset* such as "shift + mouse-click" to *save* a preset and "mouse-click" to *recall* it.

[1] `https://github.com/m---w/kollabs`
[2] `https://github.com/CICM/CreamLibrary`
[3] `http://puredata.info/downloads/sssad`
[4] c.breakpoints is a GUI external that allows the creation of different breakpoint functions

However, the method developed by *rodrigo@anorg.net*[5] was the closest to the type of preset management envisioned. This solution used the *pool*[6] object to recall data into the patch. Although it had no interpolation methods, and needed to be used in "looped" (see Fig. 2) connection with the GUI, its structure was a great starting point for this project. Still, these discoveries were never completely adequate, as they are workarounds to a problem that ideally should be solved in the source code itself.

Moreover, the testing and experimentation of the different solutions was done when implementing them in specific projects. The judgement made on each was based entirely on particular situations. This means that these libraries could be better implemented or may run "smoother" if the programming had been done on a faster system or with more time available. Issues based on installation and implementation, fast editing as well as CPU or GPU consumption in the equipment available, played an important part on the amount of usage they received. Consequently missing key features were identified and it was decided that a new and custom solution was required. This resulted in the creation of the RSVP library.

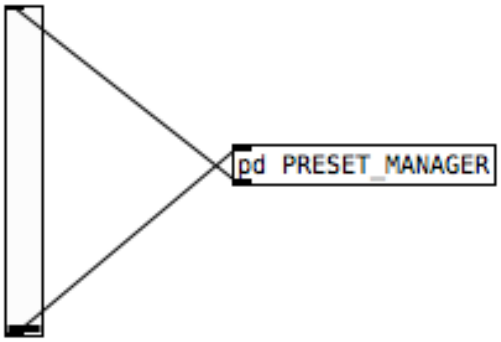

Figure 2: looped connection that some preset managers offer

2 How it works

The main idea when designing RSVP was to develop a "light and flexible (as possible)" library to meet general needs. The library had to easily be incorporated in projects by avoiding the "looped" connections (see Fig. 2), it had to implement a way to edit presets from within the patch and include a basic interpolation method between values. Another goal was to include a single click call and recall strategy with a simplified interface. This way the user would not have to struggle with loading and naming files, as well as opening n number of subpatches of settings. To accomplish this, the project was divided in three main parts:

- GUI/single click saving
- Rapid patching
- Managing the presets with automatic creation/loading of files containing the data

2.1 GUI/single click saving

The design of GUI objects, which contain the ability to store and recall presets, must be based on the easy creation of the objects and easy recall of the presets. Thus it was decided to link the Data and the GUI, as opposed to Kollabs, which is based on the principle that separates the GUI from the data processing [Weger, 2014]. A mechanism of state saving based on native vanilla GUI objects was programmed by creating a wrapper around these. The wrapper would save the state of a variable when it received a global "save preset" type message or bang that would register the value into a *coll* [7] object with the unique ID of the abstraction that generated it. This would simplify the recording of the data by the values inside *coll*, and allow easy recalling of the values by routing it to the abstraction based on an "ID" given when created.

2.2 Unique ID (keep score of data with iemguts)

A fundamental component for the development of RSVP was the iemguts[8] library and the *stat*[9] object. After experimenting with *dollarsign-zero*[10] to create unique IDs, it was understood that *dollarsign-zero* number is only unique for each session; once the file is closed and opened again, that unique number changes. Consequently, it would make the already saved data useless if saved in a previous session. Using the *canvasname* external of the iemguts library, allows the query of window names and arguments,

[5] only remaining information found on the author

[6] `https://grrrr.org/research/software/pool/`

[7] `https://puredata.info/downloads/cyclone`

[8] `https://puredata.info/downloads/iemguts`

[9] `https://puredata.info/downloads/hcs`

[10] A mechanism to create a unique ID inside Pure Data to help with the creation of abstractions

and enables the creation of unique IDs to save the data. The *canvasname* external provides the name of the parent patch, and by using *canvasargs* and providing the necessary information, it is possible to have multiple instances called (see Fig. 3). On the other hand, the *canvasindex* external provides a way to keep count of the number of instances. This is crucial for the deletion of GUI abstractions and the synchronisation of all the Data to be stored inside the *coll*.

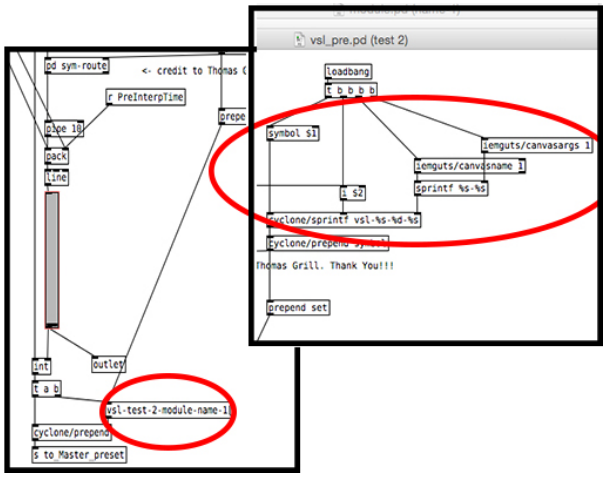

Figure 3: use of *canvasname* & *canvasargs* in RSVP

The system works by adding the name of the parent patch to the initial unique name created, either by hand or using the *GUI-creator* that comes with RSVP. If there is no parent patch, a unique name is expected but not necessary, although that code, if no unique name is provided, will not work correctly if multiple instances of it are used in the same patch. The unique name (using dollarsign number/local variable) allows the correct use of multiple copies of code in a patch. By using this standard method to handle the appointment of IDs, RSVP allows flexibility to use presets in different ways, including nesting or multiple instances.

2.3 Track of Instances and Deletions

One of the biggest challenges of creating a library that records states, is to correctly map values if an instance of the object recorded is deleted. Many basic preset solutions made with Pd involve the use of an array to record values. Although this is a very fast and efficient way to build a preset system, this method does not take object deletion into account. When objects are deleted, the array is resorted and those deleted are "popped" out, causing the values in the array to shift their positions thus corrupting the data. In order to solve this, it was decided to implement a way in which all values are recorded every time a preset is saved. Consequently, the list of values in the *coll* object with values of the new object IDs created in the patch is updated, without any that may have been deleted.

RSVP builds a unique message in Pure Data -with all the variables recorded- that is later pushed into a preset slot in the *coll* object. This is achieved by using the "add2" message to a blank message box that receives all the values when the "SaveMaster" message sent from the *PresetManager* abstraction is received in each RSVP object. The challenge in this part of the program is to know when to push the message into the *coll*. To accomplish this, the *canvasindex* from iemguts library is extremely important. The object keeps track of all the number of RSVP abstractions being used. In this way, the patch knows when all values have reached the empty message as its length must be double the number of instances. Consequently, the message can be pushed into the *coll* object in the *PresetManager*. The presets are saved in a text file in the same directory where the preset manager is being called from. It creates a text file with the SUFFIX "-preset" added.

2.4 Recalling

Every GUI abstraction accepts only values that correspond to the ID tag given for the preset. The *sym-route*[11] abstraction routes messages like the built-in *route* object, but accepts *symbols* instead of *floats* as the *type* of data to process. This abstraction receives each pair of values from the *coll* object when a preset is recalled. The value is routed properly when the ID values match and advances to the interpolation stage achieved with the *line* object.

3 GUI Abstractions

The RSVP library mirrors all vanilla GUI objects plus the *breakpoints*[12] and the *knob*[13] externals. With this selection, most needs of a typical simple patch are covered. Every abstraction has the functionalities of the original "wrapped" object plus an interpolation time that will be sent from the PresetManager. Their names also try to resemble the object native to

[11] This abstraction was coded by Thomas Grill
[12] Part of the tof library
[13] Part of the flatgui library

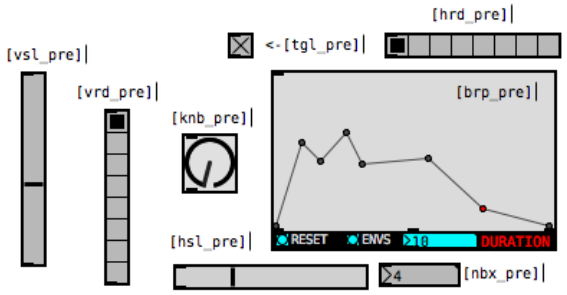

Figure 4: GUIs available in RSVP

vanilla in order to create the objects more easily (see Fig. 4). The suffix "_pre" to the available objects creates the wrapped version with the exception of the breakpoints and knob object, which are part of other libraries and are abbreviated to brp_pre and knb_pre respectively.

3.1 The Breakpoints Abstraction

The breakpoints abstraction brought some complications to the saving and recalling technique that was being implemented. While the *PresetManager* handles pairs of messages formed by the abstraction's unique ID and the value, the breakpoints object allows the creation of an envelope with a list of values. By adding a *coll* object to the brp_pre abstraction, lists can be stored and saved independently thus allowing the storage and recalling of objects that use more than one value The values of the internal *coll* are stored in a different text file with the file extension ".brp".

3.2 The Miscellaneous Abstraction

In addition to the GUIs offered, RSVP includes a "msc_pre" abstraction which can be used to save different values in a sub preset and be recalled by the PresetManager. This abstraction allows the use of RSVP to write other types of data in case the native RSVP abstractions cannot fulfil certain needs. The msc_pre abstraction was initially created to store variable amounts of points of the breakpoints abstraction explained above. It was later duplicated as an independent abstraction to offer a way of recalling data in objects not native to RSVP. The "msc_pre" abstraction creates an additional textfile, with the file extension ".msc", that records the presets assigned specifically to this abstraction.

The abstraction is linked to the *PresetManager* by receiving the number of the internal preset to recall, in the same way as the brp_pre abstraction. The main purpose for the creation of this abstraction is offer the possibility of a modular preset systems, but it also allows the use of the library with other abstractions or externals from different developers. In the example (see Fig. 5), the msc_pre object is used with the *matrixctrl*[14] object. Different ways of using the library with the msc_pre abstraction and a new "local" feature, are currently being tested and are discussed further on this paper.

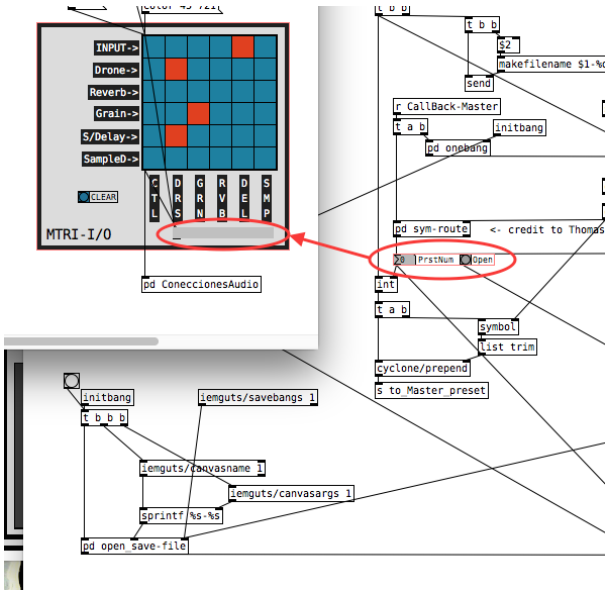

Figure 5: RSVP msc_pre object working with jmmmp's *matrixctrl*

4 Usage

4.1 Rapid Patching with the Help of the *GUI-creator* Abstraction

Initially, the intention was to hack Pd's Tcl/TK frontend and link the "put" action of the main menu to the creation of every GUI that comes with RSVP. Eventually, it was concluded that developing a Dynamic Patching abstraction named *GUI-creator*, would be the best solution[15] to develop the idea quickly (the initial idea is still being researched with the use of the Tcl/Tk plugin API).

The abstraction creates the RSVP GUIs with the click of a button. It takes care of the sequential SUFFIX that is entered for the unique ID and allows for the quick creation of abstractions

[14] `puredata.info/downloads/jmmmp`

[15] I decided to wait until having good results once RSVP was finished to start thinking on further developments.

if developing something that needs a matrix of knobs or toggles[16], for example.

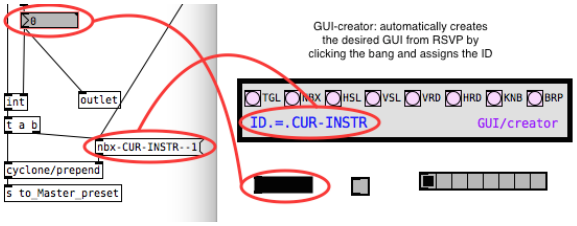

Figure 6: *GUI-creator* assigns the ID and increments the SUFFIX as it creates objects

An interesting problem surfaced while developing the *GUI-creator* abstraction. In situations when the user deletes the *GUI-creator* (it is intended to be deleted after use), and for some reason needs to add more RSVP objects with it. The *GUI-creator* first queries how many instances of that object already had been created previously and then continues to increment the SUFFIX number from that value on (see Fig. 6). To provide the abstraction with this number, the same abstraction creates a text file and stores the number and type of RSVP abstractions it creates. This record is used to initiate a new count, starting from this number plus one, when a new instance of *GUI-creator* is called.

The *GUI-creator* abstraction, only needs to keep count of instances created. If the number of instances recorded is different than the number present in the patch, then an instance or instances of RSVP abstractions were created but later deleted. In this case the value in the SUFFIX is more than the true number of RSVP objects used. However, RSVP uses that number as the SUFFIX of the ID allowing an infinite amount of unique IDs by incrementing on the previous known total.

RSVP works by keeping count only of instances created and values saved the moment a preset is recorded. The library was developed around the idea of how to discard the data that becomes obsolete when a preset is rewritten. RSVP takes care of this by using the destructive editing feature of the *coll* object to purge obsolete data. Furthermore, if obsolete data exists because *coll* has not been updated and this data is recalled, then the values are not processed as there are no instances of the sym-route abstraction linked to that ID.

[16]Video Examples: `http://www.jrsv.net/pure-data-preset-system`

4.2 *PresetManager*

The *PresetManager* is the module that controls the state saving and the value recalling of the stored data. The module consists of two main parts that take care of the saving and recalling of the values. The abstraction contains a GUI with visual feedback when an action is taken. It also allows the recording and recalling of any given position in the *coll* and contains the interpolation time control in a number box. Finally, the patch also allows the user to display the values for fast queries and/or editing in a "popup" window (see Fig. 7),. The *PresetManager* will save the contents of the *coll* object every time the patch is saved.

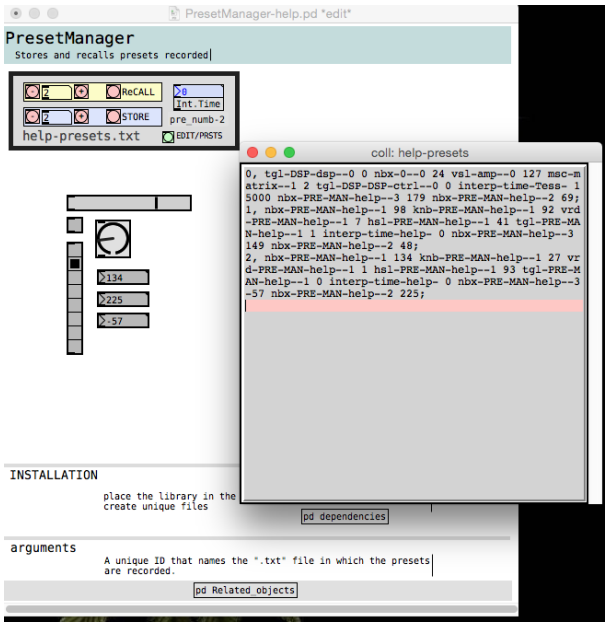

Figure 7: PresetManager help file with the popup winddow displaying the presets

4.3 Dealing with Multiple Instances

The user can call multiple instances of projects using RSVP abstractions by allowing the creation of an instance ID. This works following the way that Pd uses \$0 and \$*n* to create local and global variables. This feature was created in case the user needs two modules that use RSVP in the same master patch. A similar type of use can be observed in projects like *Automatonism*[Eriksson, 2017] or *Context*[Goodacre, 2017] that allow the creation of modular instruments to connect as desired in a patch. RSVP takes care of this by allowing an argument to set a name on creation.

4.4 Customizing RSVP

The current version of RSVP works as a local library inside a project. This means that the folder containing RSVP should be copied and placed in the main directory of the file being used as a main patch. The reason for this is that the library uses GUI objects that alter the source code of the abstractions if modified; causing the GUI to change for all files calling the library. For this reason, RSVP is intended to work as a local library letting the users customize the abstractions source code with the colors and sizes set for each specific project.

The flexibility that RSVP offers is based on the ability to modify the graphical properties of the abstractions. The modifications are as extensive as what a user can modify to the wrapped GUI objects. Extending the objects available is as easy as duplicating the source file of the GUI and using it as a template to be modified. The user can then apply all changes and save the personalised abstraction under a custom name or keep the changes to the original. If the object uses a new name, it will still be compatible with the RSVP preset system when called.

4.5 Modularity: implementation of "Local Presets"

The RSVP library offers different ways to have local presets stored in modular projects. This provides added flexibility as RSVP can be used with the user's own GUI design. With the use of msc_pre abstraction, it is easy to achieve any type of modular state saving. To make this user friendly, a "local" method inside msc_pre and brp_pre (see Fig. 8) was programmed thus giving a straight forward way of using RSVP like this. The objects can receive a message with a "local" flag and the values 1 or 0 to turn on/off the ability to read and write the values of their local *coll* object. This means that the user can program modules using their own GUIs and write their local presets such as timbres of a synthesizer in a msc_pre that will not be controlled by the *PresetManager* in the parent patch.

5 What is next? ...the "to-do" list

The RSVP library was created in such a way that improvements could be made later according to structural areas of the system. Different ideas are already being tested, including the possible switch from native GUI to native lookalikes done with Data Structures. This move to

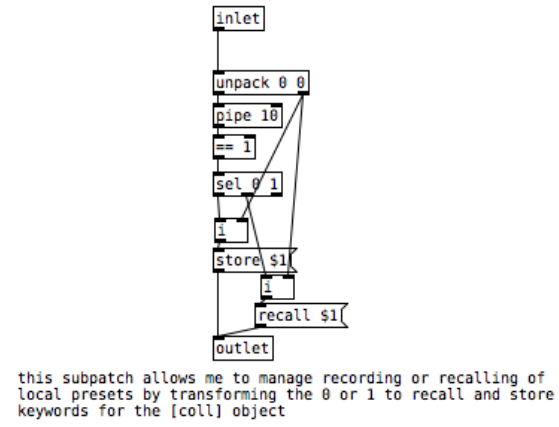

Figure 8: part of the patch that allows the recalling and recording of local presets

data structures might help develop a GUI with design changes and more importantly, have the data stored in the data structure itself.

Additional modifications are being considered to parts of the library in charge of the data storing. The implementation of the *text* object instead of the *coll* external in RSVP would make the library close to being vanilla-friendly.

The last improvement currently considered for the RSVP library is the implementation of the *propertybang* object of the iemguts library. By using this object, the library could be modified to run as a global library and be installed as any other offered, making the customization of the GUI easier to implement.

6 Envisaging a META section in the ".pd" file

During development of RSVP, it became evident that having the information stored as simple text is very useful. Future development is trying to use less files to store and recall information consequently centralizing everything into a single file. Other ideas include saving data inside the ".pd" file itself. This way the RSVP data could be accessed using the *text* object within the patch. Being able to store in a META section, opens additional possibilities for Pd users. Information such as credits, licensing and state saving could be stored with the patch. Other options may include building an abstraction by running a patch or a place to store scripts for externals like py/pyext, pdlua or pdlisp.

It is possible to read the Pd file as text inside Pd. Unfortunately, when Pd loads a file

with extra information on it, it produces warnings every time text that is not part of a normal Pd file is found. This means that any information saved in the patch as simple text to the ".pd" file is ignored. Using a simple text editor it is easy to write anything in the file and successfully save it, however when the same file is opened again in Pd, it will produce warnings and ignore that information if saved from Pd. This is why a META section for the file could be implemented. One solution could have an "EOF" (End of File) to stop Pd from reading the information that is stored in a META section.

The META section of the file would not be read by Pd when loading. It would instead be accessed as a plain text file inside Pd, with a *text* object and a message/method META sent to one of its inlets. Once the META section is accessed, the user could read it line by line and modify the information retrieved as needed. This feature would allow projects like RSVP and others to extend Pd to fulfil other needs easily.

7 Conclusion

The RSVP library offers a rapid way of saving different states in Pure Data. The design of the library was based on other solutions, but there was a desire to simplify and modify them to better solve various needs in different musical projects. RSVP offers a number of tools that help with the creation and performance of Pd patches and intends to have a solid, simple and fast way of managing presets.

The advantage of using a wrapper in its design allows the system to be easily modified and used in other versions of Pd. It can improve tools that other flavors of Pure Data have and gives space for quick development in modular areas of the system.

While this project was an attempt to solve something that Pd has been missing, it also proved that Pure Data is a flexible system that can take care of complex programming challenges like the one described. Nevertheless the library still lacks key feayures such as different types of interpollations and vanilla *friendlier* code.

Unfortunatelly, RSVP cancells usefull functionalities found in the "Properties" menu of its native GUI Objects. It is crucial for further development to address these setbacks and reinstate this functionalities *via* the wrapper. Finally, the code needs to be optimized as there are some processes that could be done more efficiently.

The RSVP library is in constant change and currently in an *Alpha Stage*. Anyone is welcome to download and use it in projects, but the developing changes according to user experience. Because RSVP is a local library, backwards compatibility is not a serious problem but more testing is needed to offer proper support. The library can be found in the URL address:

- `https://github.com/JRSV/RSVP`

Some videos introducing its features are hosted on the following website

- `http://www.jrsv.net/pure-data-preset-system`

8 Acknowledgements

RSVP uses third party libraries developed by people from the community. I wish to acknowledge the developers of libraries used in this project and individuals that helped answer questions on the pd mailing list and social media, including Matt Barber, Liam Goodacre and Thomas Grill. RSVP uses the following libraries that can be downloaded and added through the "deken" manager:

- HCS
- iemguts
- iemlib
- flatgui
- cyclone
- tof
- zexy

RSVP was developed thanks to the support of the University of Edinburgh in Scotland during the course of my PhD. I wish to thank my supervisor Dr. Michael Edwards for advice and guidance.

References

Johan Eriksson. 2017. Automatonism. `https://www.automatonism.com/the-software/`.

Liam Goodacre. 2017. Context Sequencer. `https://contextsequencer.wordpress.com`.

Pierre Guillot. 2014. CreamLibrary: A set of PD externals for those who like vanilla... but also want some chocolate, coffee or caramel. https://github.com/CICM/CreamLibrary.

James McCartney. 1996. SuperCollider SuperCollider. https://supercollider.github.io/.

Miller Puckette. 1996. Software by Miller Puckette. http://msp.ucsd.edu/software.html.

Ge Wang and Perry Cook. 2002. ChucK => Strongly-timed, On-the-fly Music Programming Language. url-http://chuck.cs.princeton.edu/.

Marian Weger. 2014. Kollabs / DS - a state-saving system with scene morphing functionality for Pure Data. https://iem.kug.ac.at/fileadmin/media/iem/projects/2014/weger.pdf.

David Zicarelli. 1990. Max Software Tools for Media | Cycling '74. https://cycling74.com/products/max/.