**Documentation (Script *PyoPlug*) :**

Cette documentation sera utile pour tout développeur souhaitant créer des scripts pour mes plugiciels. S’il y a besoin d’étudier plusieurs exemples, veuillez vous référer aux scripts du dossier des préréglages (« Presets »). Veuillez noter que le *serveur Pyo* est initialisé à l’interne du plug-in et ne devrait pas être manipulé à l’intérieur d’un script. Lors de l’initialisation, le plug-in vérifie si *Pyo* a été installé en version 64-bits (module *Python*: *pyo64*). Si c’est le cas, la version 64-bits de *Pyo* est chargée. Sinon, ce sera la version 32-bits (module *Python*: *pyo*).

Serveur - Entrées/Sorties (E/S) :

Les entrées et les sorties fonctionnent de la même manière que tout autre type de pilote audio *Pyo*. Cependant, le nombre d’entrées et sorties du plugiciel dépend de l’initialisation de celui-ci par le séquenceur ou logiciel utilisé. Ainsi, un plug-in mono à stéréo aura une seule entrée et deux sorties. Ces E/S peuvent être utilisés à l’intérieur du script de cette façon :

*# Entrées:*

*# Remarque : Bien que monoIn et stereoIn soient indiqué ici à titre*

*# d’exemple, ces objets sont prédéfinis à l’interne du plug-in et peuvent*

*# être utilisé tel quel; cela a été fait afin de simplifier et d’accélérer*

*# l’écriture de scripts*

monoIn = Input(chnl = 0)

stereoIn = Input(chnl = [0, 1])

*# Traitement Audio*

dist = Disto(stereoIn)

*# Sorties:*

*# La méthode "out()" peut être utilisée normalement*

out = dist.out()

*# Ou afin de bien gérer les sorties, il est possible de*

*# spécifier les numéros de sorties voulues*

out = dist.out(chnl=[0, 1])

*# De même, la class Mix peut être utilisé*

out = Mix(dist, voices=2).out()

Paramètres du séquenceur :

Quelques variables et objets *Pyo* créent d’intermédiaire afin de retrouver des paramètres du séquenceur dans le script.

**Variables :**

sr : Fréquence d’échantillonnage

bufferSize : Nombre d’échantillons audio par mémoire tampon

**Objets *Pyo* :**

dawSamplePos : Position sur la ligne de temps (en nanoseconde)

dawQuartetPos : Position sur la ligne de temps (en noir)

dawBPM : Nombre de battement par minute

dawTimeSigNum : Numérateur de l’indication de mesure

dawTimeSigDenom : Dénominateur de l’indication de mesure

dawIsPlaying : Détermine si le séquenceur est en mode lecture ou non

Valeurs : 1 = Lecture, 0 = Arrêt

Peu être utilisé afin de mettre un script au silence

Ex : out = Sine(freq=400, mul=dawIsPlaying).out()

Compatibilité Cecilia :

Afin de faciliter la conversion des modules Cecilia, la fonction *duplicate* fut ajoutée à l’interne du plug-in et peut donc être utilisée par tout script. Cette fonction duplique les éléments d’une séquence d’après l’argument « num ». Cette fonction peut être utilisée afin de créer des listes qui correspondent au nombre de canaux multiplié par le nombre de voix.

def duplicate(seq, num):

tmp = [x for x in seq for i in range(num)]

return tmp

Chemin du script :

Le chemin du script peut être accédé avec la variable : « pathToPyoScript *»*

Message d’erreur :

Lorsque le plug-in tente de charger un script et qu’une erreur dans ce dernier l’empêche de bien l’exécuter, une onde sinusoïdale se fera entendre pendant deux secondes. Cela permet de facilement s’apercevoir d’un problème avec le script au lieu de se demander pourquoi il y a que du silence.

Ordre et nombre de scripts et dossiers :

Puisqu’il est seulement possible d’utiliser des chiffres dans les formats de plug-in *VST* et *AU*, l’ordre des fichiers et dossiers a une grande importance puisqu’ils pourront seulement être appelés par index. Ainsi, afin de toujours garder le même script ou dossier à la même valeur d’index, un chiffre a été ajouté avant chaque nom. Cela est particulièrement important si nous voulons que des sauvegardes de paramètres puissent être constantes malgré l’ajout de scripts dans le dossier. De plus, un nombre maximum de dossiers et de scripts par dossiers a été établi. Cela permet de facilement trouver un script avec l’interface graphique et de le voir toujours attribuer le même index; par exemple, le premier script dans le dossier aura la valeur 0.00, le second aura la valeur 0.01 et ainsi de suite.

Définition des paramètres à contrôler par le plugiciel :

La fonction principale qui permet l’interaction entre le plug-in et le script est définie par : *defineUI()*. Celle-ci est la fonction clé afin de définir les paramètres du plug-in en spécifiant son nom affiché à l’utilisateur, le minimum et maximum des valeurs ou une liste de valeurs ou fichiers à utiliser, l’unité à afficher, ainsi que la linéarité des valeurs par le potentiomètre à glissière de l’interface. De plus, elle s’occupe de la transformation des messages *OSC* reçus en un objet *Pyo* avec le nom spécifié par le développeur du script. De même, elle permet au développeur de spécifier une fonction à appeler lorsqu’un paramètre est modifié.

***La fonction :***

**defineUI**(id=1, name='paramName', func='none', label='Param', file=False, path='none',

unit=' ', min=0., max=1., init=0., rel='lin', power=0., value=['none'],

\*args, \*\*kwargs)

***Définition des arguments :***

id : (int) Numéro du paramètre à définir

name  : (string) Nom de l’objet *Pyo* à créer pour la réception des messages

*OSC* avec ce paramètre afin des utiliser dans le script

func  : (string) Nom d’une fonction à appeler à chaque modification d’un paramètre

label : (string) Nom de paramètre a afficher dans l’interface graphique. Si le format

*VST* est utilisé, notez qu’un maximum de 8 caractères peut être utilisé.

file  : (bool) Spécifie que des fichiers seront utilisés comme valeurs à ce

paramètre

path  : (string) Conjointement avec l’argument *file*; Indique le chemin du dossier

contenant les fichiers à utiliser. Si celui-ci n’est pas spécifié, le dossier

des fichiers sons (« 0-Sounds ») inclus avec le plug-in sera utilisé. Les sons

du dossier avec leur chemin d’accès sont accessibles avec la liste nommé

« filesList+id » où *id* correspond au numéro du paramètre.

Ex : filesList5[3] donnera le 4e son dans la liste du paramètre 5.

unit : (string) Nom d’unité à afficher

min : (float) Valeur minimale du paramètre

max : (float) Valeur maximale du paramètre

init  : (float ou string) Valeur initiale du paramètre lors de l’initialisation

du script. Si une liste de valeurs est spécifiée à l’argument *value*, la

valeur donnée à l’argument *init* pourra être l’index de la valeur à

utiliser ou la valeur elle-même dans la liste.

rel : (string) Linéarité des valeurs.

Valeurs possibles : 'lin' – linéaire

'log' – logarithmique

'exp' – exponentiel

power : (float) Spécifie la base logarithmique ou la puissance exponentielle à

utiliser. Si non spécifié : si *rel* = 'log', une base 10 sera utilisée

si *rel* = 'exp', une puissance 2 sera utilisée

value : Spécifie les valeurs possibles du paramètre. Une liste de caractère doit

être spécifiée. Cependant, l’index dans la liste sera retourné comme

valeur du paramètre.

\*args, \*\*kwargs : Ces arguments ne sont pas utilisés, mais sont spécifiés afin

d’empêcher le plug-in de planter s’il y a des erreurs d’arguments

Important: Les valeurs retournées par le plug-in vers le script sont passées par des objets *Pyo*. Par ce fait, les valeurs seront toujours reçues en tant que nombre à virgule flottante (*float*). Ainsi, il est au développeur du script de s’assurer de convertir les valeurs au type voulu, au besoin. Par exemple, la conversion en nombre entier (*int*) pourra se faire avec : int(objPyoDuParametre.get())

Afin de mieux comprendre le fonctionnement de la fonction *defineUI()*, voilà quelques exemples :

*# Simple définition avec quelques arguments utilisant les valeurs par défaut; ex: min = 0 et max = 1*

defineUI(id=1, name="env", label="Gain", unit="x", init=.8)

*# Spécification de minimum et maximum*

defineUI(id=2, name="transp", label="Transpo", min=-24, max=24, init=0, rel="lin", unit="cents")

*# Modification de la linéarité vers une fonction logarithmique*

defineUI(id=3, name="cut", label="FiltFreq", min=100, max=18000, init=2000, rel="log", unit="Hz")

defineUI(id=4, name="filterq", label="FiltQ", min=0.5, max=10, init=0.707, rel="log", unit="Q")

*# Spécification d’une fonction à appeler et d’une série de valeurs avec initialisation sur le nom*

*# d’une valeur dans la liste*

defineUI(id=5, name="filttype", func="filttypefunc", label="FiltType", init="Lowpass", value=["Lowpass","Highpass","Bandpass","Bandstop"])

*# Spécification d’un dossier contenant des fichiers audio*

defineUI(id=6, name="sndidx", func="sndchoice", label="SndTable", file=True, init="ounkmaster.aif", path=os.path.join(os.path.expanduser('~'), "Library/Audio/Presets/PyoPlug/0-Sounds/"))

*# Exemple d’utilisation des paramètres dans le script :*

*# Initialisation de variables; le chemin vers un son est défini simplement afin que le script soit*

*# chargé sans erreurs, il sera ensuite initialisé au paramètre voulu par le plug-in*

usrPath = os.path.expanduser('~')

sf = SfPlayer(os.path.join(usrPath, "Library/Audio/Presets/PyoPlug/0-Sounds/ounkmaster.aif"), loop=True, mul=0.3)

*# Utilisation de l’objet Pyo* « transp »; celui-ci sera modifié en temps réel par le plug-in

harm = Harmonizer(sf, transpo=transp, winsize=0.05)

*# Les objets Pyo « cut » et « filterq »* seront modifiés en temps réel. Cependant, l’argument « type »

# nécessite une conversion en nombre entier. Ainsi, l’appel d’une fonction sera nécessaire. Ici, 0 est

# spécifié afin d’initialiser à une valeur quelconque

filt = Biquadx(harm, freq=cut, q=filterq, type=0, stages=2).out()

# Fonction appelée à chaque fois que le paramètre « FiltType » est modifié. Notez la conversion en

# nombre entier.

**def** **filttypefunc**():

filt.type = int(filttype.get())

def sndchoice():

sf = SfPlayer(filesList6[int(sndidx.get())], loop=True)

harm.setInput(sf)

Variables et objets réservés :

Quelques variables et objets sont utilisés à l’intérieur du plug-in, ils sont par le fait voués au bon fonctionnement de celui-ci. Ainsi, bien qu’ils puissent être utilisés à l’intérieur d’un script afin de retrouver une information, ces variables et objets ne devraient pas être modifiés par ce dernier. Ceux-ci sont :

**Variables et objets réservés :**

pyoPlugServer : (objet *Pyo*) Serveur *Pyo*

pyoPlugOscPort : (objet *Pyo*) Port *OSC* utilisé

NBR\_PARAMS : (int) Nombre maximum de paramètres dans le plug-in

pathToPyoScript : (string) Chemin d’accès au script chargé

varsDefault : (list) Liste des objets dans le contexte courant avant

qu’un script soit chargé

paramAddrList : (liste) Liste des adresses *OSC*

pyoPlugParam : (objet *Pyo*) Réception des messages OSC - OscReceive*()*

pyoPlugParamUI : (dict) Dictionnaire *Python* utilisé pour passer les

valeurs de la fonction *defineUI()* au plug-in