

**GARETH OLUBUNMI HUGHES**

**Student ID Number: 0944776**

**A Submission for the Award of a PhD Qualification in Music  
at Cardiff University**

**Title of Thesis: “A Portfolio of Acoustic/Electroacoustic  
Music Compositions & Computer Algorithms that Investigate  
the Development of Polymodality, Polyharmony, Chromaticism  
& Extended Timbre in My Musical Language”**

**VOLUME #2 (of 2): “Academic Commentary”**

**August 2015**



# Abstract

The emphasis of this PhD is in the field of original/contemporary musical composition and I have submitted a portfolio of original compositions (volume 1/2, comprising of music scores of both acoustic and electroacoustic music compositions [totalling c. 114:30 minutes of music] as well as written material relating to notation and artistic motivation), along with an academic commentary (volume 2/2 [totalling c. 19,500 words], which places the original compositional work in the portfolio in its academic context).

The composition works in first volume are varied and broad ranging in scope. In terms of pitch organisation, the majority of works adopt some form of modality or polymodality, whilst certain works also incorporate post-tonal chromaticism and serialism into their syntax. Certain key works also explore extended timbre and colouration (in particular for bowed strings, voices, flute and electronics) and adopt the use of timbral modifications, harmonics, microtones, multiphonics, sprechgesang (i.e. ‘speech-song’), phonetics and the incorporation of electroacoustic sampling, sound synthesis and processing.

The academic commentary in the second volume sets out several initial theoretical pitch organisation models (namely relating to modes, polymodes, rows, serial techniques and intervallic cells), with a particular emphasis placed on the formation of a melodic/harmonic language which is fundamentally polymodal, polychordal and polyharmonic.

The commentary then takes a closer look at various works within the portfolio which adopt modal, polymodal and chromatic forms of pitch-organisation (whilst intermittently discussing wider musical parameters, such as rhythm, counterpoint, timbre, structure etc...). Separate chapters also discuss a work for flute and electronics and a lengthy work for string quartet (inspired by urban dystopian film) in greater depth.

The commentary also discusses my style of writing, placing individual works within the portfolio in their academic context alongside key influences as well as contextualising non-musical aesthetics and sources of artistic inspiration relating to my work.





# Contents

<i>Pages 6–8</i>	<b>Project Aim</b>
<i>Pages 9–27</i>	<b>Chapter 1:</b> Pitch Organisation Models
<i>Pages 12–16</i>	Modes
<i>Pages 16–21</i>	Polymodes
<i>Pages 21–25</i>	Rows & Serial Techniques
<i>Pages 26–27</i>	Intervallic Cells
<i>Pages 29–47</i>	<b>Chapter 2:</b> Modality
<i>Pages 49–59</i>	<b>Chapter 3:</b> Explicit Polymodality
<i>Pages 61–70</i>	<b>Chapter 4:</b> Electronic & Computer-Generated Sonorities
<i>Pages 71–82</i>	<b>Chapter 5:</b> “ <i>Urban Wilderness</i> ” for String Quartet
<i>Pages 83–88</i>	<b>Chapter 6:</b> Style, Influence & Context
<i>Pages 89–98</i>	<b>Bibliography</b>
<i>Pages 99–100</i>	<b>Appendix 1:</b> Table of Diatonic Modes Referred to in the Academic Commentary
<i>Pages 101–2</i>	<b>Appendix 2:</b> A Chart of Natural Harmonics on the Contrabass
<i>Pages 103–7</i>	<b>Appendix 3:</b> Handwritten sketches for <i>Human Visions: “Civilisations”</i> for symphony orchestra
<i>Pages 109–21</i>	<b>Appendix 4:</b> Handwritten sketches for #2: “ <i>Utopian Mirror</i> ” from “ <i>Urban Wilderness</i> ” for string quartet
<i>Pages 123–33</i>	<b>Appendix 5:</b> Handwritten sketches for “ <i>Amber on Black</i> ” for solo SATB singers
<i>Pages 135–58</i>	<b>Appendix 6:</b> A User Guide for <i>Eternal Owl Call</i> ’s Electronic Performance Interface
<i>Pages 136–7</i>	Contents
<i>Pages 159–308</i>	<b>Appendix 7:</b> SuperCollider Documentation & Source Code for the <a href="#">Z_Library</a>
<i>Page 161</i>	Contents
<i>Pages 163–219</i>	SuperCollider Class Extension Library Documentation
<i>Pages 220–308</i>	SuperCollider Class Extension Library Source Code

## Project Aim

The portfolio of compositions submitted (i.e. vol. 1), along with the academic commentary contained in this second volume of the PhD investigate the possibility of creating a hub between modal and chromatic forms of composing music, as well as investigating the possibility of incorporating extended timbre, colouration and electronics into a style of composing which remains inherently modal or polymodal. Key exponents of post-tonal theory and early 20<sup>th</sup> modernism are important to this progression within my work: in particular Debussy, Ravel, members of *Les Six*, Messiaen, Bartók, Starvinsky and members of the Second Viennese School. Compiling the portfolio, whilst being influenced by other key composers and theorists, has taken me on a journey towards an individual musical language which, despite bearing certain characteristics of the above mentioned influences, has a voice of its own nonetheless.

One particularly unique theoretical system which operates in my work is been that of ‘stacking’ chromatically related modes on top of one another in different pitch regions and then unifying these modes through the use of a ‘polymode’ (a continuous interrelated combination of modes which do not repeat at the octave – leading to a deliberate avoidance of any sort of octave doubling in the majority of my work). This concept is discussed in detail in the *Polymodes* section of chapter 1 (vol. 2, pp. 16–21) and arguably stands as the most significant contribution to music theory within this PhD project.

In addition, a few of the most experimental works within the portfolio engage with more specific influences of later modernists who developed more radical approaches to composing contemporary music during the second half of the 20<sup>th</sup> century and the 21<sup>st</sup> century. Of particular note, *Utopian Mirror* (the second movement of *Urban Wilderness* for string quartet [vol. 1, pp. 72–99]), *Amber on Black* for four a cappella singers (vol. 1, pp. 253–89) and *Eternal Owl Call* for bass flute and electronics (vol. 1, pp. 291–300) are, perhaps, the most experimental works within the portfolio: the use of extended string timbre and unorthodox string harmonics in *Utopian Mirror* can boast influences as diverse as composers such as Geroage Crumb, Kajia Saariaho, Helmut Lachenmann and Salvatore Sciarrino; the use of extended vocal techniques and phonetics in *Amber on Black* can cite the vocal writing of Luciano Berio, György Ligeti and Karlheinz Stockhausen as influences; in addition, the incorporation of electronic sound synthesis, live electronic processing, stochastic procedures,<sup>1</sup> extended flute timbre and symbolism with the natural world in *Eternal Owl Call* are partially indebted to the influence of composers such as Toru Takemitsu, Stockhausen, Iannis Xenakis and Saariaho.

---

<sup>1</sup> As pertaining to some of the random and stochastic generative procedures described in Iannis Xenakis’ *Formalized Music: Thought and Mathematics in Composition* and relating to similar techniques adopted in Xenakis’ electronic and/or computer-generated music compositions.

The order in which the individual works occur in the portfolio (i.e. vol. 1) is explained and laid-out as shown in the table below (i.e. starting with large-scale works and then eventually working its way through to more intricate and specialist chamber and electroacoustic works):

### Order of Works in the Portfolio of Compositions

Category	Order #	Title of the Piece
<b>Large Scale: Instrumental</b> [in order of magnitude (i.e. the number of instruments employed)]	<b>1</b>	<i>Human Visions: “Civilisations”</i> for symphony orchestra
	<b>2</b>	<i>“Urban Wilderness”</i> for string quartet
<b>Choral</b> [in order of magnitude (i.e. the number of individual vocal lines employed)]	<b>3</b>	<i>“Ynys Afallon” (“Isle of Avalon”)</i> recomposed for SSAATTBB choir a cappella
	<b>4</b>	<i>“Ynys Afallon” (“Isle of Avalon”)</i> for SATB choir & pianoforte
<b>Chamber Works: Instrumental</b>	<b>5</b>	Arrangement/Re-composition of <i>“Cwyn y Gwynt”</i> ( <i>“The Wind’s Lament”</i> ) for flute & harp
	<b>6</b>	<i>“Twilight Impulse”</i> for clarinet, cello & pianoforte
<b>Instruments &amp; Voices</b>	<b>7</b>	<i>“Cwyn y Gwynt” (“The Wind’s Lament”)</i> for contralto, vibraphone & cello
	<b>8</b>	<i>“Y Gwyllanod” &amp; “Iâr fach yr haf”</i> ( <i>“Butterfly...”</i> & <i>“The Seagulls”</i> ) for soprano, flute/piccolo & pianoforte
<b>A Cappella Voices</b>	<b>9</b>	<i>“Amber on Black”</i> for solo SATB singers
<b>Mixed Electroacoustic</b>	<b>10</b>	<i>“Eternal Owl Call”</i> for solo bass flute with live electronic processing

Since *Urban Wilderness* is a work for string quartet, it would normally be classed as a chamber work, however, due to the very long structure of both movements and the vision that this piece be combined with live video projection at some point in the future it has been categorised as a large-scale instrumental work as indicated.

Chapter 1 of the commentary sets out several initial theoretical pitch organisation models (namely relating to modes, polymodes, rows, serial techniques and intervallic cells), with a particular emphasis placed on the formation of a melodic/harmonic language which is fundamentally polymodal, polychordal and polyharmonic. Chapters 2 and 3 then take a closer look at various works within the portfolio which adopt modal, polymodal and chromatic forms of pitch-organisation (whilst intermittently discussing wider musical parameters, such as rhythm, counterpoint, timbre, structure etc...). Chapter 4 discusses the use of the SuperCollider audio synthesis computer programming language before taking a closer look at *Eternal Owl Call* for bass flute and electronics (which uses this language for its computer algorithms). Chapter 5 discusses *Urban Wilderness* for string quartet (a work inspired by urban dystopian film) in greater depth. Chapter 6 then further discusses my style of writing, placing individual works within the portfolio in their academic context alongside key influences as well as contextualising non-musical aesthetics and sources of artistic inspiration relating to my work.

# CHAPTER 1

## Pitch Organisation Models

# **Pitch Organisation Models**

The style of pitch organisation in my work has developed and evolved perpetually since I was an undergraduate student at King's College London (graduating in the year 2000). In the years that have followed, I have experimented with several recognised styles of composition and pitch organisation, ranging from modal and post-tonal theory through to the use of serialism, microtones and electronics. Several unique and elaborate systems of pitch organisation have emerged from this, forming an essential part of my style of composition. Although some of my works adopt a number of methods which look outside of the realm of the 12-tone Pythagorean system adopted in Western art-music (i.e. by using natural harmonics, microtones, electronics etc...), the core of my most recent work has centred on a number of methods which act as a hub between modal and chromatic styles of writing.

I have attempted to compile a portfolio which is broad ranging in scope and demonstrates a high level of variety and versatility. Some of the pieces are unashamedly modal in style; however, the majority adopt some form of polymodality (operating in several chromatically related modes simultaneously<sup>1</sup>). I have categorised the pitch organisation style in such pieces as being either 'explicitly polymodal' or 'chromatically polymodal': explicit polymodality is where the individual modes which form each polymodal or polyharmonic block are transparent and can clearly be related back to a more conventional modal style of writing; chromatic polymodality is where the individual modes which form each polymodal or polyharmonic block are disguised and diluted by high levels of chromaticism – in such instances the modal origins are not immediately obvious and some passages might even come close to atonality. In addition, some pieces in the portfolio adopt various Schoenbergian or Stravinskian methods of manipulating chromatic note-rows or dividing them into smaller modes or pitch-collections, which can be linked to the polymodal systems in my work.

I have classed the various methods of pitch organisation relating to individual works within the portfolio as being either modal (without the use of any form of polymodality), explicitly polymodal (as defined in the above paragraph), chromatically polymodal (ibid.) or using some form of serial rotation. In addition, certain works also incorporate extended timbre and colouration or electronics into their respective soundworlds. The table overleaf categorises the pitch/timbral organisation methods in operation in each individual movement within the portfolio according to the six methods defined above (namely Modal, Explicitly Polymodal, Chromatic/Polymodal, Serial Rotation, Extended Timbre & Colouration, Electronics):

---

<sup>1</sup> This differs from "polytonality" which shares multiple diatonic/non-modal 'tonal centres' (i.e. shares multiple major/minor keys).

## Pitch Organisation Chart

- Primary Feature
- Secondary Feature

Individual Movement	Modal	Explicitly Polymodal	Chromatic/ Polymodal	Serial Rotation	Extended Timbre & Colouration	Electronics
<i>Human Visions: “Civilisations”</i> for symphony orchestra			●	●	○	
<i>Urban Wilderness #1: “Apathetic Machines”</i> for string quartet		●				
<i>Urban Wilderness #2: “Utopian Mirror”</i> for string quartet			●	●	●	
<i>“Ynys Afallon”</i> for SSAATTBB a cappella	●					
<i>“Ynys Afallon”</i> for STAB singers & pianoforte		●	●	○		
<i>Cwyn y Gwynt #1: Llwydnos Gwynfannus</i> for flute & harp		●	○			
<i>Cwyn y Gwynt #2: Galargan “Dagrau ddaw...”</i> for flute & harp	●	○				
<i>Cwyn y Gwynt #3: Breuddwyd</i> for flute & harp		●	●	○		
<i>“Twilight Impulse”</i> for clarinet, cello & pianoforte		●	○			
<i>Cwyn y Gwynt #1: Gwylltineb</i> for alto, vibraphone & cello		●	○			
<i>Cwyn y Gwynt #2: Galargan</i> for alto, vibraphone & cello	●	○				
<i>Cwyn y Gwynt #3: Breuddwyd</i> for alto, vibraphone & cello		●	●	○		
<i>“Iâr fach yr haf ”</i> for soprano, flute & pianoforte	○	●				
<i>“Y Gwylanod”</i> for soprano, piccolo & pianoforte	○	●	○			
<i>“Amber on Black”</i> for SATB singers a cappella			○	○	●	
<i>“Eternal Owl Call”</i> for bass flute & electronics			○	○	●	●

As indicated, most works adopt multiple methods in either a primary or secondary capacity.

Prior to analysing the individual pieces in my portfolio, it is important that I first illustrate some of the core theoretical ideas and models of pitch organisation which are adopted in my work:


## Modes

Both the Lydian<sup>2</sup> and Dorian mode are adopted predominantly in my modal and polymodal systems. I effectively treat the Dorian mode as the ‘relative minor’ equivalent to its Lydian mode counterpart. This is the by-product of many years of composing with these modes as well as working with them in jazz, folk and other styles of improvisatory music. I have always liked the bright sound which these modes possess and prefer them to the, perhaps more conventional, Ionian and Aeolian modes adopted predominantly in western art-music prior to the twentieth century.

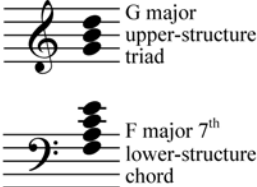
I regard the Lydian mode to be brighter than the Ionian mode due to the augmented fourth interval which exists from its root (as opposed to a perfect fourth) and consequently the greater number of sharpened intervals which it possesses (i.e. C Ionian is neutral whereas C Lydian has one sharp, G Ionian has one sharp whereas G Lydian has two sharps etc... similarly of their respective relative minor modes: A Aeolian is neutral whereas A Dorian has one sharp, E Aeolian has one sharp whereas E Dorian has two sharps etc...).

Both modes can also be used to construct the bright and colourful polychords shown in the examples based on white-note modes below:

F Lydian



F major 13<sup>th</sup>  
polychord



**Fig 1.1.1**

<sup>2</sup> A table containing the 7-note and 5-note diatonic modes discussed in this commentary [namely modes of the major, melodic/harmonic minor, major (b6) and pentatonic scale respectively] is supplied in **Appendix 1** (vol. 2, p. 100).





Fig 1.1.2

Another feature in my modal and polymodal systems is the idea of upper/lower-structure chords or patterns being in operation within a specific pitch region as illustrated above.

It is often a characteristic of my style that one mode floats chromatically to the next without a conventional modulation or cadence (as would be seen in western art-music prior to the twentieth century), thus conventional cadences are relatively infrequent; however, some examples of harmonies with a conventional dominant function do occasionally occur and I adopt special chromatic modes for this purpose. Such modes might similarly be seen in the style of impressionist or post-impressionist composers (i.e. Debussy, Ravel, Messiaen), Bartók, Stravinsky or various styles of modern jazz. Below is an example of three modes with harmonic blocks that might form the basis of typical cadences:

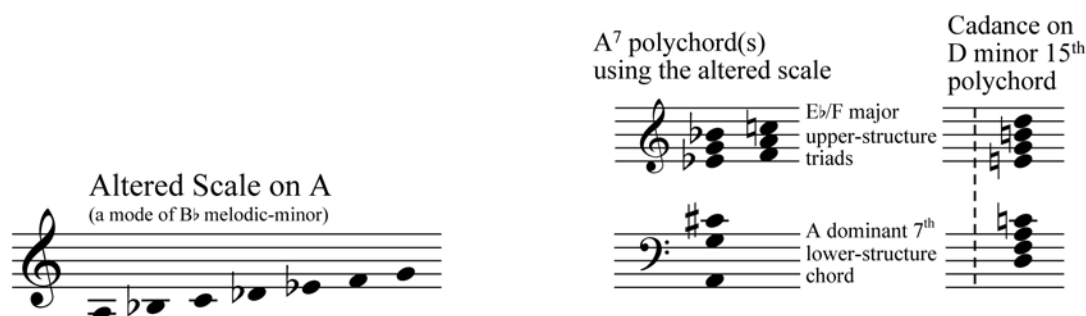


Fig 1.1.3

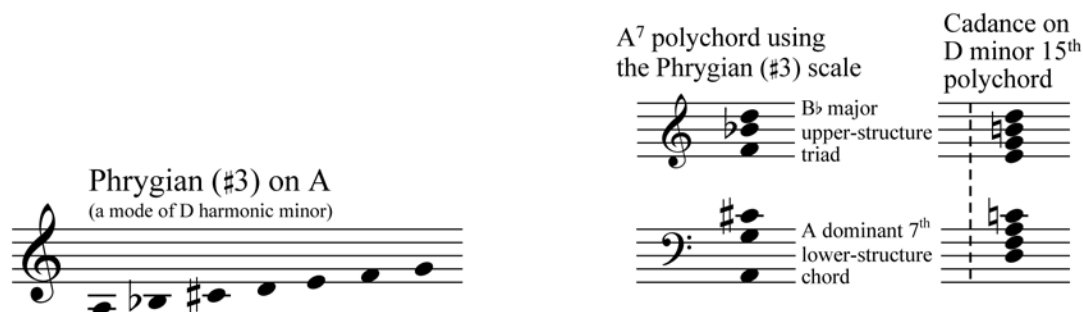

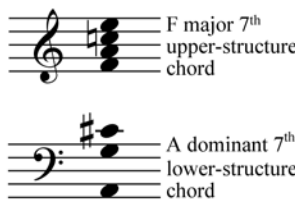


Fig 1.1.4

Phrygian ( $\flat 4$ ) on A  
[a mode of F major ( $\flat 6$ )]



A<sup>7</sup> polychord using the Phrygian ( $\flat 4$ ) scale



F major 7<sup>th</sup> upper-structure chord

A dominant 7<sup>th</sup> lower-structure chord

Cadance on D minor 15<sup>th</sup> polychord

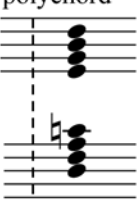
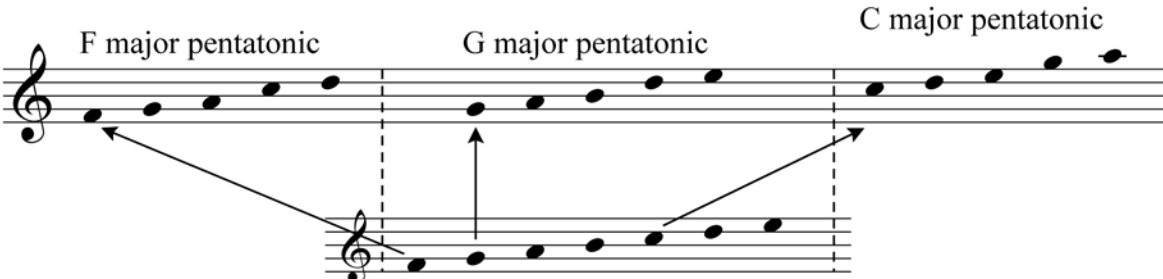


Fig 1.1.5

Note that the accidental spellings on the upper/lower-structure chords occasionally differ from the logical spelling used on the mode shown in isolation on the left (i.e. in **Fig 1.1.3** & **1.1.5** the lower-structure chord uses C $\sharp$  rather than D $\flat$  – this is to clearly show that the lower-structure chord is an A dominant 7<sup>th</sup>, which should normally be spelt with a C $\sharp$ ). Slight modifications of this nature are often made so that the upper/lower-structure chords within a polychord are spelt in a natural way.

Pentatonic modes are also frequently used in my work. If we look at a Lydian mode on F (or any 7-note white-note scale for that matter) 3 separate major pentatonic scales can be extracted from it as shown below:



F major pentatonic

G major pentatonic


C major pentatonic

Basic pentatonic scales extracted from F Lydian

Fig 1.1.6

Pentatonic modes work well when used for upper-structure melodic/harmonic patterns as illustrated in the two following examples:

G major pentatonic as upper-structure melodic mode



Perfect fifth on the root of F Lydian as lower-structure chord

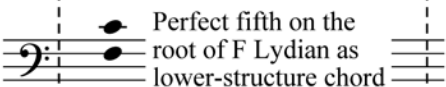



Fig 1.1.7

E $\flat$  major pentatonic as upper-structure melodic mode



A dominant 7<sup>th</sup> lower-structure chord

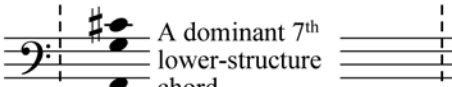


Fig 1.1.8

**Fig 1.1.8** uses all the pitches of the altered scale on A as shown in **Fig 1.1.3**

There is a subjective avoidance of symmetrical scales and arpeggios in my work (in particular the chromatic, whole-tone<sup>3</sup>, hexatonic, octatonic<sup>4</sup> scales and diminished 7<sup>th</sup> arpeggios). This is a decision which I have consciously made as a composer as I generally find the sound of repeated symmetrical patterns to be somewhat monotonous and lacking in subtlety – that said, highly effective use of symmetrical patterns has been seen in the work of composers such as Debussy and Stravinsky: The whole-tone scale (i.e. C–D–E–F#–G#–A#–C) is utilised very successfully in works such as Debussy’s solo piano piece *Cloches à travers les feuilles* (from *Images*, Set 2, 1907) and his operatic work *Pelléas et Mélisande*<sup>5</sup> (1898), as well as Paul Dukas’ *Ariane et Barbe-bleue*<sup>6</sup> (1906); of the two latter works, Messiaen states in *The Technique of My Musical Language* that both works “have made such remarkable use of it [the whole-tone scale] that there is nothing more to add. Then we shall carefully avoid making use of it, unless it is concealed in a superposition of modes which renders it unrecognizable”.<sup>7</sup>

In a similar vein, remarkable use of the octatonic scale (i.e. C–C#–D#–E–F#–G–A–Bb–C) has already been seen extensively in much of Stravinsky’s work, as seen in works such as his ballet *Petrushka* (1910–11) – this includes the recurrence of its (in)famous ‘Petrushka chord’ extracted from the pitches of a single octatonic scale, which juxtaposes two major triads a tritone apart and recurs in various horizontal or vertical formations throughout the work. Bartók has made remarkable use of the hexatonic scale (i.e. [C–D#–E–G–Ab–B–C], otherwise known as the ‘magic hexachord’) in pieces such as the third movement of the *Concerto for Orchestra* (1943) marked “*Elegia*” *Andante non troppo* to create an evocative and mystical musical ambiance.

However, one intention with the use of modes in much of my own work is to avoid any obvious cliché that might cause the music to sound as if it could have been composed a hundred years ago and the overuse of explicit symmetrical modes, such as the whole-tone or octatonic scale might lead to this. One of the features of diatonic and pentatonic modes is their asymmetry and this is what gives them their subtlety; this is a feature of modality that I am interested in exploring further as I develop my (poly)modal methods and style. It is, however, worth noting that the first four notes of the Lydian mode make up part of a whole-tone scale, similarly, the first five notes of the Lydian (#5) mode are whole-tones, therefore

<sup>3</sup> Defined in Olivier Messiaen’s *The Technique of My Musical Language* (Paris: Alphonse Leduc, 1956) as the ‘first mode of limited transpositions’, vol. 1, p. 59.

<sup>4</sup> Ibid., defined as the ‘second mode of limited transpositions’, vol. 1, pp. 59–60 & vol. 2, p. 50.

<sup>5</sup> Ibid., vol. 1, p. 59.

<sup>6</sup> Ibid.

<sup>7</sup> Ibid.

patterns using these modes will frequently make use of their whole-tone portions and might occasionally give the impression of using the whole-tone scale without actually doing so in an explicit fashion.

## Polymodes

**Fig 1.1.1** illustrates how a Lydian mode is used as the basis for an F major 13<sup>th</sup> 7-note modal polychord. Similarly, **Fig 1.1.2** illustrates how a Dorian mode is used as the basis for a D minor 15<sup>th</sup> 8-note polychord. A feature of both of these chords is that their consecutive intervals always alternate between a major and minor third. However, if one were to continue this pattern on the F major 13<sup>th</sup> chord the next pitch in the series would be an F $\sharp$  rather than an F $\natural$  – a pitch which does not belong to the Lydian mode on F. This idea creates a polychord which effectively has two different pitch regions in two different keys simultaneously. This new pattern can be used as the basis for forming a polymode which combines two different Lydian modes in two differing pitch regions:

Two octave polymode stacking G/F Lydian

F Lydian      auxiliary note      G Lydian

F major 15<sup>th</sup>  
polychord

G major 7<sup>th</sup>  
upper-structure  
chord

F major 7<sup>th</sup>  
lower-structure  
chord

**Fig 1.2.1**

This polychord (which is a signature chord in my work and frequently occurs in some guise) now features an interesting ‘false relationship’ between F $\sharp$  and F $\natural$ . In this particular instance it is logical to describe the interval between these pitches as an ‘augmented octave’ or ‘augmented fifteenth’ rather than a compound minor 9<sup>th</sup> or minor 2<sup>nd</sup>, because of the modal relationship which exists between these two pitches (i.e. as the upper structure of the polymode/polychord is based on G Lydian it is normally logical to think of the top pitch as an F $\sharp$  rather than a G $\flat$ ). It is also interesting to note that despite the false relationship the polychord retains a consonant and neutral sound.

The polychord effectively stacks two major 7<sup>th</sup> chords a major 9<sup>th</sup> apart on top of one another and the polymode created from this chord stacks two Lydian modes a major 9<sup>th</sup> apart on top of one another (with an auxiliary note added to link the two modes). This pattern can be continued for a further two octaves to produce an extended polymode and polychord as shown below:

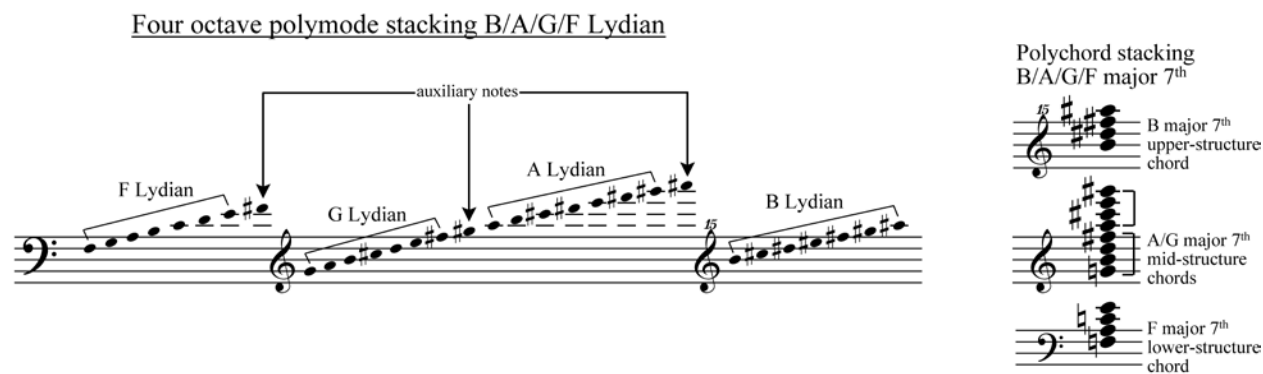


Fig 1.2.2

All twelve chromatic pitches operate simultaneously in this extended polychord. It also passes through each respective interrelated major/minor 7<sup>th</sup> chord in a cycle of fifths as it moves upwards through the pitch spectrum. This is illustrated in the diagram below, which uses colours to help with visually differentiating between each chord:

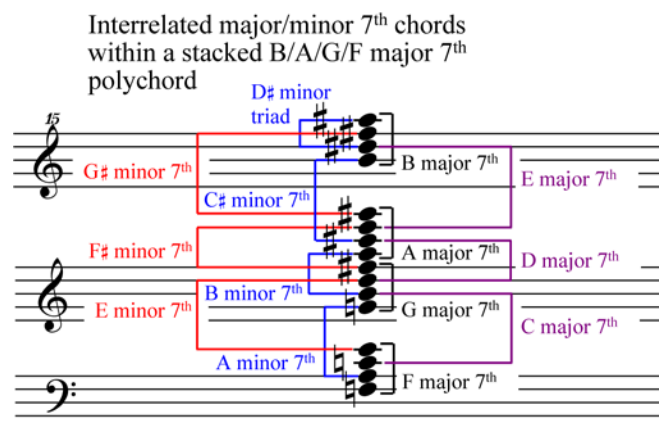


Fig 1.2.3

Similarly the extended polymode passes through each respective interrelated Lydian/Dorian in a cycle of fifths as it moves upwards through the pitch spectrum:

Interrelated Lydian/Dorian modes within a stacked B/A/G/F Lydian polymode

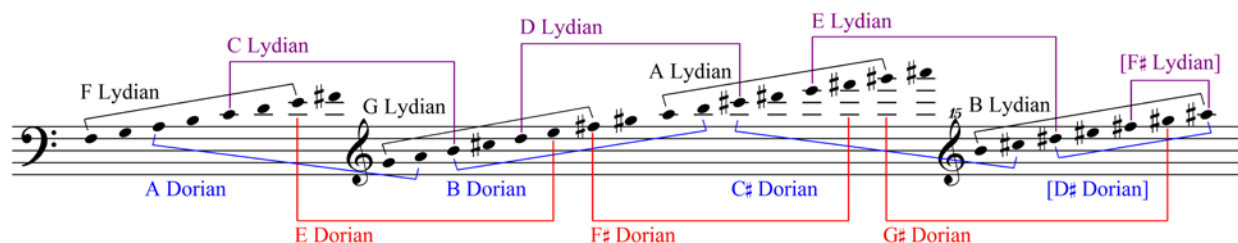


Fig 1.2.4

If we think of F3 as being the 'root' of the polymode above, it is an interesting feature of this polymodal system that transposing the polymode up or down an octave would effectively 'change key', thus each individual pitch within the pitch-spectrum has, in effect, its own unique key with an 'absolute root'. As a

consequence, there is a subjective avoidance of consecutive octaves in my style of composing. I also frequently (but not exclusively) try to avoid doubling the same pitch class at the octave within an individual harmonic block; however melodic or contrapuntal material based on each block might include auxiliary notes which double at the octave without forming part of an individual block.

This polymodal system differs from the technique described as “polymodal chromaticism” by Bartók<sup>8</sup> in relation to his own polymodal style, where multiple chromatically related modes operate simultaneously whilst sharing a common root (e.g. in *Mikrokosmos No.80* [“*Hommage à Robert Schumann*”, from *Book III*, 1926–39]) C-Phrygian/Lydian are employed simultaneously comprising all twelve pitches of the chromatic scale [C–D $\flat$ –E $\flat$ –F–G–A $\flat$ –B $\flat$ /C–D $\sharp$ –E $\sharp$ –F $\sharp$ –G–A $\sharp$ –B $\sharp$ ]. Here, Bartók juxtaposes chromatically related modes within the same pitch regions, which differs from my own polymodal style in which chromatically related modes are superimposed (or ‘stacked’) in different regions of the pitch spectrum.

Another well-known early twentieth century work to adopt this form of polymodal chromaticism was Poulenc’s *Mouvements Perpetuels, I.* (1918),<sup>9</sup> which includes passages which stack a B $\flat$  major (or Ionian mode) ostinato pattern in the left-hand below a B $\flat$  Phrygian or Locrian hexachord-based melody (i.e. employing B $\flat$ –C $\flat$ –D $\flat$ –E $\flat$ –G $\flat$ –A $\flat$ ) in the right hand. This comes closer to the stacked polymodal system which I adopt in my own work because there are chromatically related modes in operation in different regions of the pitch spectrum; however, a key difference to emphasise is that the Poulenc example employs two modes which share a common root of B $\flat$ , whereas in my own system I frequently employ modes in different pitch regions which do not share a common root. To illustrate this, consider the polymode in **Fig 1.2.1**: I regard F $\sharp$ 3 to be the absolute root of this polymode; however, the upper structure mode is G Lydian (which does not contain an F $\sharp$ ) and the auxiliary note which links F Lydian and G Lydian is F $\sharp$ 4, therefore it *cannot* properly be said that various octave transpositions of F $\sharp$ 3 (namely F $\sharp$ 4 or F $\sharp$ 5) are also roots because they do not feature in the polymode. However, the fact that G Lydian forms the upper-structure to this polymode does not mean that multiple roots are in operation, instead G Lydian is an interrelated collection which forms part of larger synthetic/extended polymode with a common absolute root of F $\sharp$ 3.

The stacked Lydian/Dorian polymodes illustrated in **Figures 1.2.1–4** are employed exhaustively in my style and might be regarded as typical type of ‘pure-form’ polymodes. If we consider these polymodes, the interval from one adjacent pitch to the next is always either a major or minor second but it

<sup>8</sup> Benjamin Suchoff, *Bartók’s Mikrokosmos: Genesis, Pedagogy and Style* (Lanham, Md: Scarecrow Press, 2002), 115.

<sup>9</sup> John Vincent, *The Diatonic Modes in Modern Music* (Berkeley: University of California Press, 1951), 272.

avoids the inclusion of two adjacent minor second intervals – this is a characteristic which such polymodes share with conventional diatonic scales. As a consequence I subjectively avoid the inclusion of trichords which consist of two adjacent minor seconds within any individual harmonic block, as illustrated below:

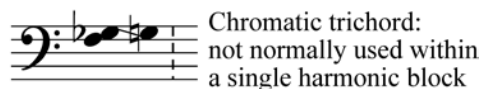


Fig 1.2.5

However, the following trichords are frequently seen within individual harmonic blocks:

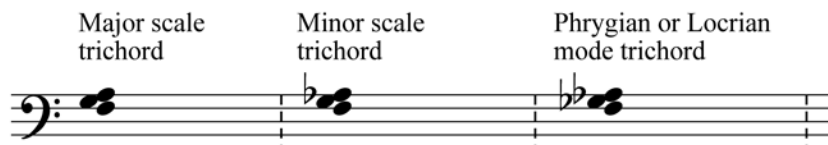


Fig 1.2.6

It is, of course, logical to add harmonic variety by deviating from the pure-form Lydian/Dorian polymodes as any musical passage runs its course and one way of doing this is by ‘omitting’ a particular pitch-region within the pure-form series and then shifting other regions within the series up or down a specified number of octaves. For example, if I were to remove the G Lydian and B Lydian portions from the polymode in **Figures 1.2.2–4** and then shift the A Lydian portion down an octave (adding a couple of auxiliary notes to link the two Lydian modes) I might produce the following new polymodal pattern:

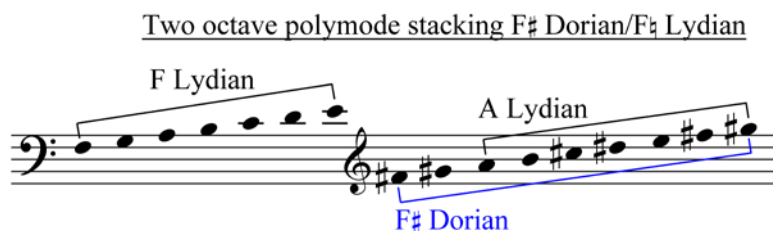
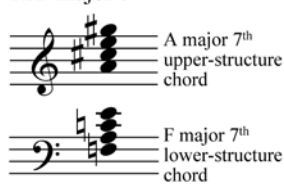


Fig 1.2.7

Polychord stacking  
A/F major 7<sup>th</sup>



Similarly, if I were to remove the G Lydian and A Lydian portions and then shift the B Lydian portion down two octaves I might produce the following pattern:

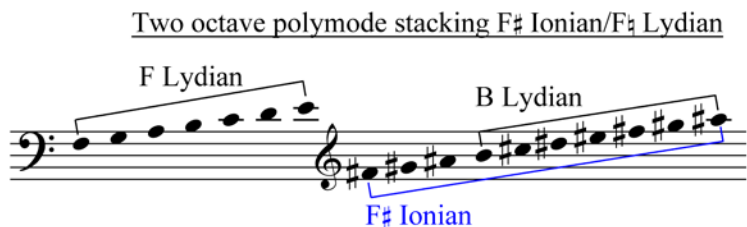
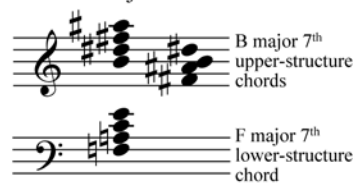


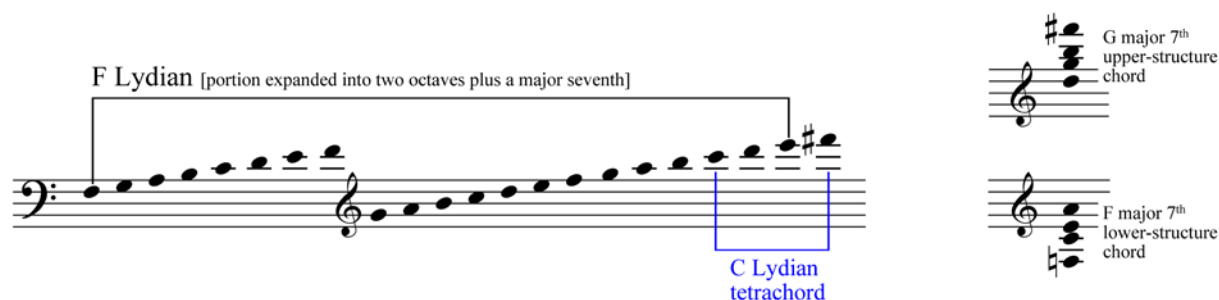
Fig 1.2.8

Polychord stacking  
B/F major 7<sup>th</sup>



The above diagram also illustrates the possibility of re-voicing the upper structure B major 7<sup>th</sup> chord (so that it is in 2<sup>nd</sup> inversion position) as a means of avoiding the slightly unnatural gap between the two root position major 7<sup>th</sup> chords.

Another method of changing the ‘shape’ of a pure-form polymode is to vary its range of an interralted mode within the series (e.g. I might change mode after a fifth or two octaves rather than every octave). Thus the rate of change of the stacked Lydian/Dorian modes will be compressed or expanded respectively from its pure form. The following example shows an expansion of the polymode seen in **Fig 1.2.1**:



**Fig 1.2.9**

Note that the above polymode is very close to actually being modal and the false relationship does not occur until the final high F# in the series. Another point to clarify here is that when a root mode (in this case F Lydian) is expanded in this way, it opens up the possibility of multiple roots being shared on octave transpositions of the same pitch-class: because F Lydian has been extended for two and a half octaves in the above polymode, this means that F3, F4 and F5 can all be regarded as roots but F6 cannot because it does not feature in the polymode.

The following example shows a compression of the polymode seen in **Fig 1.2.1**



**Fig 1.2.10**



Generally, expanding a polymode will reduce the perception of chromaticism and shift it towards modality whereas compressing a polymode will do the opposite and increase the perception of chromaticism.

These methods of generating and modifying polymodes open up the idea of visualising an ‘invisible’ (or hidden) polymode/polychord which spans the entire musical pitch spectrum. Then when we have harmonic sequences each individual harmonic block within the sequence contains a new invisible polymode/polychord. Examples of this technique will feature later on in the commentary relating to a harmonic analysis of sections within some of the pieces in the portfolio.

There is a complexity to the theory behind such a system, however I have found that adopting and developing these theories over a continuous period of time has meant that extracting interesting pitch material from a corresponding invisible polymode has become second nature to me, leading to a system of pitch organisation which is both unique and elaborate but also practical enough to work with nonetheless.

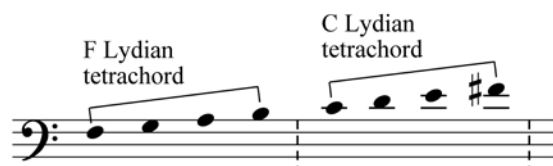
## **Rows & Serial Techniques**

When adopting rows in my work, I will generally think of them as defining the individual pitch-classes used for the purposes of constructing individual harmonic/melodic blocks, without necessarily defining the exact octave within the pitch spectrum where that particular pitch-class might occur – this differs from the interrelated polymodal system previously illustrated which does define the exact octave for each individual pitch within it. Any individual pitch-class should not be repeated within a row, therefore when adopting rows which contain more than 7 individual pitch-classes I will always be working with an initial set which ventures outside of the realms of any individual 7-note diatonic mode – thus the row is treated as being part-modal and part-chromatic/polymodal.

Rows are rarely adopted explicitly in my recent compositional work; by this I mean that one would be unlikely to find many examples of rows which are repeated, retrograded, inverted and transposed continually (as is the case in serial and other row-based works by pioneers in this field such as Schoenberg or Webern). Instead, rows are normally adopted as tools (in an auxiliary fashion) to help me create individual polymodal harmonic/melodic blocks and there is often a new row for each constituent block rather than a repetition of the same prime row or some transformation of it.

A row might be constructed from the first eight pitches of the polymode in **Fig 1.2.1** (also equivalent to the individual pitch-classes in the stacked G/F major 7<sup>th</sup> polychord on the right-hand side of the same figure):

### Polymodal 8-note Row



Example polychords  
using the pitch-classes  
in the 8-note row:

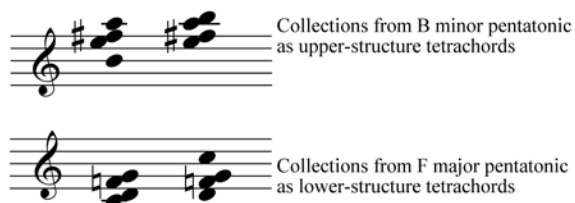
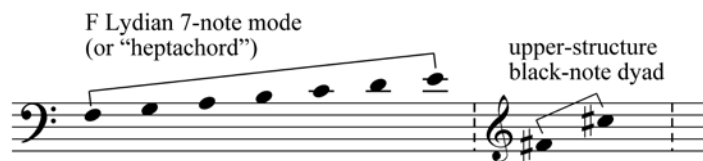


Fig 1.3.1

The above 8-note row (on the left-hand side) possesses palindromic characteristics: If inverted about F#4 it produces a retrograde of exactly the same row; when inverted about any other pitch it will produce a transposed retrograde; in addition, when the leftmost polychord is inverted about B#5 it produces the rightmost polychord, which contains two tetrachords with exactly the same notes as the former.

Similarly a 9-note row might be constructed from all nine of the individual pitch-classes which occur in the playmode in **Fig 1.2.1** (adding a C# to the row in the previous example):

### Polymodal 9-note Row (Prime)



Example polychord  
using the pitch-classes  
in the 9-note row:

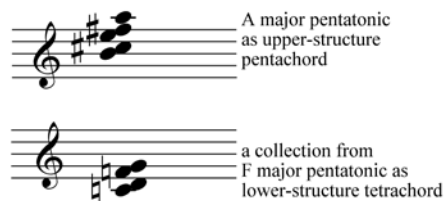
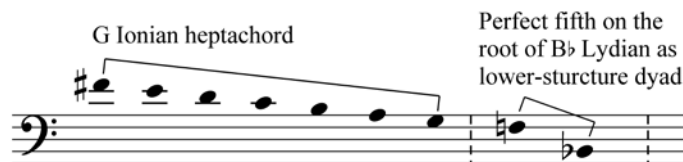


Fig 1.3.2

If the above row is inverted about F#4 it produces an interesting result with a perfect fifth at the bass end of the row:

### Polymodal 9-note Row (Inversion about F#4)



Example polychord  
using the pitch-classes  
in the 9-note row:

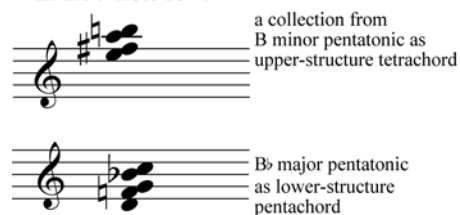


Fig 1.3.3

A 10-note row might be constructed from the individual pitch-classes of a stacked A/G/F major 7<sup>th</sup> polychord and inverted as in the two following examples:

### Polymodal 10-note Row (Prime)

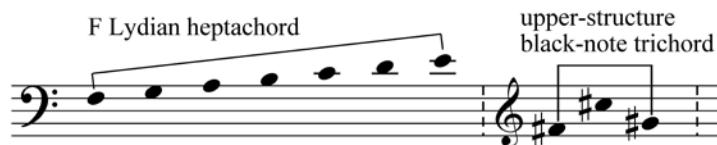


Fig 1.3.4

### Polymodal 10-note Row (Inversion about F#4)

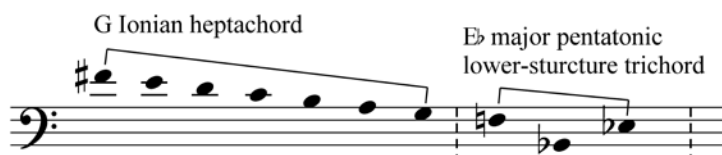


Fig 1.3.5

Similarly, an 11-note row might be constructed from the individual pitch-classes in a stacked E/A/G/F major 7<sup>th</sup> polychord and inverted as shown in the two following examples:

### Polymodal 11-note Row (Prime)



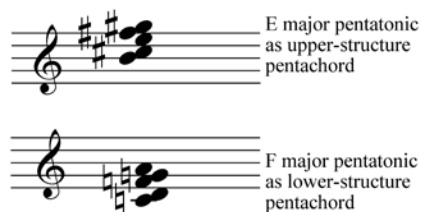
Fig 1.3.6

### Polymodal 11-note Row (Inversion about F#4)

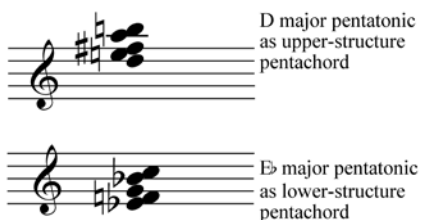


Fig 1.3.7

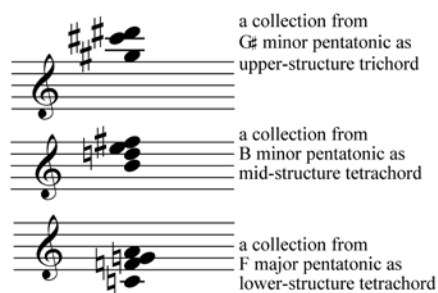
Example polychord  
using the pitch-classes  
in the 10-note row:



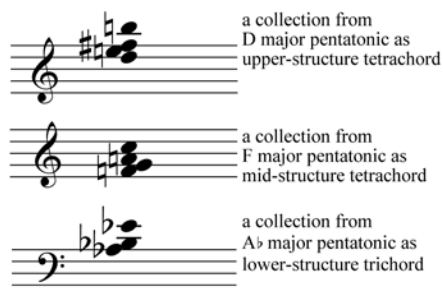
Example polychord  
using the pitch-classes  
in the 10-note row:



Example polychord  
using the pitch-classes  
in the 11-note row:



Example polychord  
using the pitch-classes  
in the 11-note row:



When the above trend is extended to a 12-note row a typical result would be a row with a 7-note diatonic mode as one set (thought of as a heptachord) which leaves a 5-note pentatonic mode (or pentachord) when rotated as shown in the next two examples:

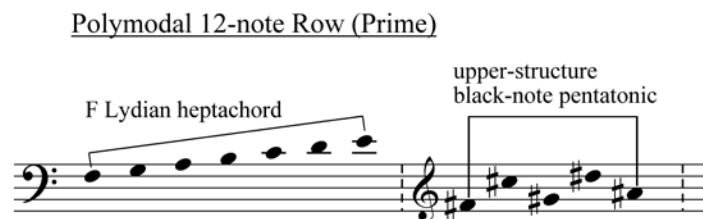


Fig 1.3.8

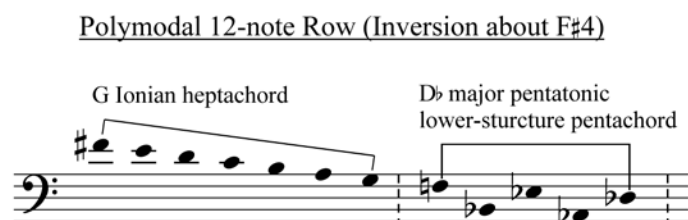
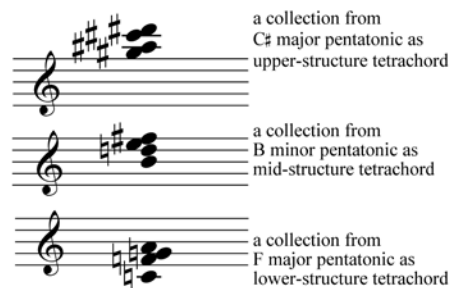


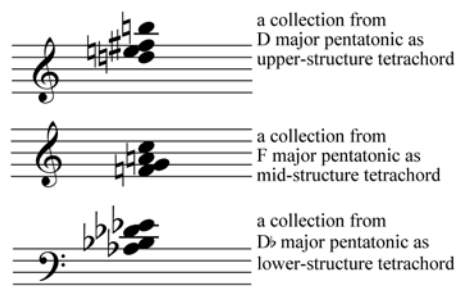
Fig 1.3.9

This way of arranging 12-note rows is a form of ‘literal pitch-class complementation’ (where an ‘aggregate’ is divided into smaller subsets, each containing pitch-classes absent from the other(s)<sup>10</sup>) and is similar, in principle, to the technique of hexachordal rotation seen extensively in Schoenberg’s works, as well in many of the late works of Stravinsky, Krenek and more recently in the works by contemporary British composers such as Oliver Knussen and Julian Anderson. One clear differentiation to make here is to emphasise my tendency to extract an asymmetrical 7-note or 5-note mode (such as a diatonic or pentatonic mode) from the row. As previously mentioned, there is a subjective avoidance of symmetrical scales in my work and when working with hexachordal rotation the resulting hexachords tend to imply symmetry (e.g. 6-note modes/scales such as the whole-tone scale, the hexatonic scale, or six-note collections from the octatonic scale). Rotating seven or five notes instead tends to lead to the resulting heptachords/pentachords being asymmetrical in one way or another. Summarily I would describe the serial rotation techniques which I adopt in my work as either “heptachordal rotation” or in some cases “pentachordal rotation.”

Example polychord using the pitch-classes in the 12-note row:



Example polychord using the pitch-classes in the 12-note row:



<sup>10</sup> Arnold Whittall, *The Cambridge Companion to Serialism* (Cambridge: Cambridge University Press, 2008), 272.

I also often work with hybrids of some of the patterns in the row-based examples above, for example if I were to create a type of 12-note hybrid of the rows in **Figures 1.3.8–9** I might produce the following result:

Hybrid Polymodal 12-note Row

Split Pentachord

Example polychord using the pitch-classes in the 12-note row:

a collection from E Lydian as upper-structure pentachord

F major pentatonic as mid-structure pentachord

A♭ perfect fifth as lower-structure dyad

**Fig 1.3.10**

Note how the pentachord in **Fig 1.3.8** has been split into a dyad plus a trichord and some of the black notes have been shifted below the register of the white-note heptachord. Similarly, the heptachord in **Fig 1.3.8** could be split as illustrated in the following example:

Hybrid Polymodal 12-note Row

Split Heptachord

Example polychord using the pitch-classes in the 12-note row:

a collection from C♯ Locrian or Phrygian as upper-structure hexachord

a collection from F Dorian or Aeolian as lower-structure hexachord

**Fig 1.3.11**

An interesting feature in the above example is that the polychord on the right-hand side is made up of two hexachords, which imply asymmetrical diatonic modes, illustrating the fact that it is, of course, possible to do this with hexachords in certain circumstances.

As illustrated in **Figures 1.3.1–7**, I will frequently be working with rows which contain less than twelve individual pitch classes but the principle of rotating asymmetrical heptachords or pentachords will still apply in the same way. Similarly, I might sometimes extend a row to more than twelve pitches for various pragmatic reasons. As an example, the stacked B/A/G/F polychord in **Figures 1.2.2–3** contains all twelve pitch-classes and four of the pitch-classes are also doubled at the octave (namely A♭, E♭, B♭ and F♯) – in appropriate circumstances this type of harmonic/melodic block might be thought of as a type of 16-note row.

## Intervallic Cells

In order to create a logical interface between (poly)modality and chromaticism in my work, I have devised a set of intervallic cells which are commonly seen in some of my most recent compositional work. This idea owes itself, in part, to the ideas of theorist David Lewin, who was a pioneer in approaching the analysis of works by Schoenberg, Webern and Alban Berg in this way, deducing typical intervallic cells adopted in the work of each respective composer. Lewin's analysis of Berg's opera *Lulu* derived several intervallic cells typical in the composer's work (i.e. z-cell, y-cell, x-cell), which could be linked to the post-tonal, post-chromatic origins which are transparent in Berg's work. A similar approach was also applied to the analysis of intervallic cells linked to chromatic polymodality in Bartók's work by scholars and theorists such as George Pearle and Benjamin Suchoff, in particular the z-cell (which combines two tritones a semitone apart [i.e. C–D♭–F♯–G]) and the x-cell (four adjacent semitones [i.e. C–D♭–D–E♭]).

The typical intervallic cells which I have applied to my own work are based on pitches and the exact intervals which occur in the polymodes which I adopt, but are not closely related to more conventional modal or tonal melodic/harmonic patterns. I am particularly fond of the major 7<sup>th</sup> and augmented octave intervals which occur in the pure-form stacked Lydian polymode (i.e. if we consider the first eight pitches within this polymode: F–G–A–B–C–D–E–F♯, I often extract the major 7<sup>th</sup> which occurs between F–E and the augmented octave which occurs between F♯–F♯), thus I divide each cell into a lower and upper structure pitch collection, where the lowest pitch in the upper-structure is either a major 7<sup>th</sup> or an augmented octave above the lowest pitch in the lower-structure. The lower and upper portion each contain between one and four individual pitch-classes (thus the maximum number of individual pitch-classes possible within the cell is eight [i.e. a lower-structure tetrachord plus an upper-structure tetrachord]). If a lower or upper-structure portion is a dyad then its two pitches must be either a major 2<sup>nd</sup> or a major 3<sup>rd</sup> apart (i.e. F+G or F+A); if it is a trichord then it must contain two adjacent whole-tones (i.e. F+G+A); if it is a tetrachord then it must contain three adjacent whole-tones (i.e. F+G+A+B, making it a Lydian mode tetrachord over a tritone).

A table containing these intervallic possibilities is provided overleaf (**Fig. 1.4.1**, on an F♯ root). I have also given each cell its own unique label (e.g. 3+2e) where the leftmost digit represents the number of pitches in the lower-structure, the rightmost digit represents the number of pitches in the upper-structure and the letter at the end is unique to that exact cell. These labels are useful for the purposes of analysis and deducing these cells from the intervallic patterns in my work.

# Fig. 1.4.1: Typical Intervallic Cells (on an F $\sharp$ root)

## simple dyads:



## 2+1 trichords:



## 1+2 trichords:



## 3+1 tetrachords:



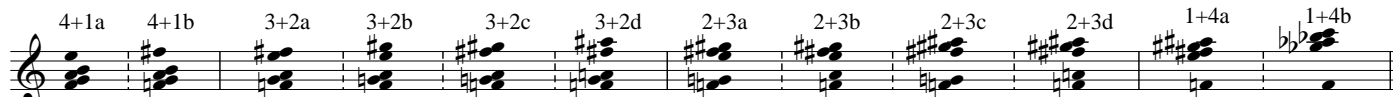
## 2+2 tetrachords:



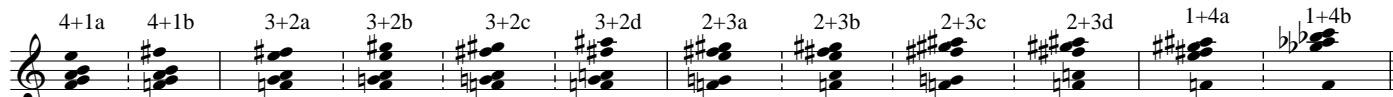
## 1+3 tetrachords:



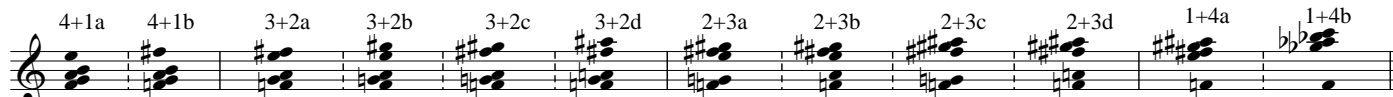
## 4+1 pentachords:



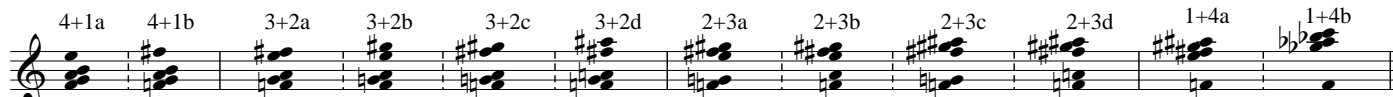
## 3+2 pentachords:



## 2+3 pentachords:



## 1+4 pentachords:



## 4+2 hexachords:



## 3+3 hexachords:



## 2+4 hexachords:



## 4+3 heptachords:



## 3+4 heptachords:



## 4+4 octachords:







# CHAPTER 2

## Modality

## Modality

*Ynys Afallon* for SSAATTBB choir a cappella and the second movement of *Cwyn y Gwynt* (both the flute/harp and contralto versions) are, perhaps, the most conventional sections of music within my portfolio. These pieces are composed in style which is predominantly modal, providing insight into how I employ the modal models previously discussed (as pertaining to **Figures 1.1.1–8**). Despite this, occasional passages within these pieces do hint, albeit in a somewhat oblique fashion, at the more elaborate form of polymodality which is present in much of my most recent compositional work.

*Ynys Afallon* for SSAATTBB (vol. 1, pp. 108–29) was composed so that it could be entered for the composer's medal at the 2013 Welsh National Eisteddfod in Denbighshire, in response to a call for a new work for a cappella choir set to Welsh-language text. It was important for this new work to be suitable for performance by a good level amateur or semi-professional choir, thus I wanted to avoid a style of vocal writing which was overtly chromatic or complex in terms of pitch-organisation. It is in fact a top-to-bottom re-composition (rather than an arrangement) of a previous setting of the same poem for SATB choir and pianoforte (also included in the portfolio), which was in a polymodal style and deemed not to be suitable for performance by amateur choirs. There are certain structural and motivic similarities between the two versions, however in reality, they are rather different from one another.

The first verse is atmospheric and harmonic, beginning with 4-part alternations between F and D major 7<sup>th</sup> harmonies (in second and third inversion respectively) in the soprano and alto parts, before a tenor melody enters in m. 3. The bass line is absent in the first few bars but there is a harmonic suggestion of an alternation between F and D Lydian modes for three bars, then an F Lydian to D<sup>b</sup> Lydian (or Ionian) to F Lydian cadence in m. 4 before the bass line finally emerges with a drone on F. The same four-bar progression is then re-ordered, starting with 4-part harmonies in the alto and tenor lines before a soprano melody enters in m. 7, this time announcing the first verse of the poem (rather than the phonetic vowel sounds previously heard), before the bass drone once again emerges at the end of m. 8.

Following this, mm. 9–19 assumes the format of an 8-voice chorale, with text from the poem being set in all parts, a greater variety of harmonic/modal changes and a metric modulation from 4/4 to 6/8 time, before a cadence and a pause on a C<sup>#</sup> Dorian harmony in mm. 18–19. The F–D Lydian modal progression from m.1 then recurs in the tenor and bass from m. 20 onwards; a harmonic/modal progression model for mm. 20–35 is provided overleaf:

**Fig. 2.1.1:** harmonic/modal model for mm. 20–35 of “Ynys Afallon” for SSAATTBB

**[END OF VERSE 1]**

**Slowly, Atmospheric, Celestial** ♩ = c.60

20 21 22 23 24

*pp e poco a poco cresc.*

ALLOS 3 3

SOPRANOS & ALTOS

TTBB

F Lydian D Lydian F Lydian E Mixolydian F Lydian D Lydian F Lydian (h) F Aeolian F Lydian

D major 7 in fourth inversion

**[START OF VERSE 2]**

**Warm, Expressive** ♩ = c.60

25 26 27 28

F# major pentatonic

melodic anacrusis on final ♩ or ♩. of each bar

SOPRANOS

D major pentatonic + G♯ dyads in perfect fourths

pppp

(mf)

p

D Lydian Altered Scale on B#C [a mode of C# melodic minor] F Lydian C Lydian

SOPRANOS & ALTOS

C major pentatonic major/quartal triads

D major pentatonic + G♯ major/quartal tetrachords

29 30 31 32

F# = chromatic auxiliary note

SOP. I

F Lydian C Lydian A Dorian Altered Scale on E [a mode of F melodic minor] E Phrygian (b4) [a mode of C major (b6)]

B♭ major pentatonic + E♭ major/quartal triads

33 34 35

A Dorian B Aeolian C Lydian D Ionian Altered Scale on E A Dorian G Lydian C Lydian

The passage from mm. 20–26 ends with the equivalent of an imperfect cadence in F on a B#/C dominant 7<sup>th</sup> harmony which uses pitches from the altered scale on B# (a mode of C# melodic minor) and uses an F# major pentatonic mode as its upper-structure collection. This is identical to the modal concept illustrated in **Fig. 1.1.8**, where an upper-structure pentatonic collection is extracted from the altered scale.

Following a pause, the second verse then begins on a quaver-pulse anacrusis at the end of m. 26 with polyphonic and somewhat fugue-like material. This alternates between F Lydian and C Lydian in adjacent bars (corresponding with the question–answer/tonic–dominant principle seen in fugues of J.S. Bach), with upper-structure pentatonic collections which relate to these modes being employed. Further examples of dominant-harmony substitution modes occur in the passage that follows, namely the altered scale on E in m. 32 and m. 34 and a brief glimpse of E Phrygian (b4) at the end of m. 32 (both cadence on A Dorian) and an altered scale on D in m. 39 which cadences on G Dorian.

A further harmonic/modal progression model for mm. 46–53 is provided overleaf:

**Fig. 2.1.2:** harmonic/modal model for mm. 46–53 of “Ynys Afallon” for SSAATTBB

[END OF VERSE 2]

**Warm, Expressive** ♩ = c.60

(*poco a poco allargando*)

46 *p* D Dorian

47 *mp*

a chromatic mix of D Dorian and D Mixolydian, the entire passage uses the exact pitch-classes in the polymode below:

48 [a mix of D Dorian and D Mixolydian]

49 *mp* A Phrygian ( $\flat 4$ ) [a mode of F major ( $\flat 6$ )]

Upper-structure 4-note chords

50 *mf* D Dorian E Phrygian F Lydian

51 A Phrygian ( $\flat 4$ ) [a mode of F major ( $\flat 6$ )]

Upper-structure 4-note chords

52 D Dorian E Phrygian F Lydian E Dorian

53 *ff* Altered Scale on A [a mode of  $B\flat$  melodic minor]

Upper-structure triads

The melismatic passage in mm. 46<sup>3</sup>–48<sup>3</sup> uses a chromatic mix of pitches from both D Dorian and D Mixolydian (above a bass pedal point on D), thus containing a chromatic alternation between a modal set containing either F $\natural$  or F $\sharp$  respectively. I have added a polymode below this passage in my diagram to show the exact collection of pitches in use. There are changes between different harmonic blocks every semiquaver beat, some of which contain F $\natural$  and some of which contain F $\sharp$ , however, there is never a simultaneous clash between these two pitches, which corresponds to the principle illustrated in **Fig. 1.2.5** (which explains that there is a subjective avoidance of trichords which might consist of two adjacent minor seconds within any individual harmonic block in my style).

Dominant-harmony substitution modes with a root of A are then used to emphasise the D Dorian-based modality in this section whilst building to the movement's climax in the third verse: Firstly in mm. 48<sup>4</sup>–49 using A Phrygian ( $\flat 4$ ) and then from the final quaver beat of m. 50 to m. 51 using the same mode (relating closely to the example A Phrygian ( $\flat 4$ ) cadence illustrated in **Fig. 1.1.5**). The altered scale on A is then employed in mm. 52<sup>3</sup>–53 with upper-structure triads in the soprano and alto lines (similarly relating closely to the example altered scale cadence illustrated in **Fig. 1.1.3**).

Finally, a harmonic/modal progression model for mm. 54–57 is provided overleaf:

**Fig. 2.1.3:** harmonic/modal model for mm. 54–7 of “Ynys Afallon” for SSAATTBB

**[START OF VERSE 3]**

**Broad, Powerful, Majestic** ♩ = c.84

54 *f* D melodic minor D Dorian Altered scale on D G A D G A Dorian

55 D Dorian C Lydian Altered scale on B E F# G# A D E F# Dorian Lydian Dorian

56 *mf* D E♭ D Dorian B♭ C D A♭ B♭ B♭ C# D# E♭ G♭ Lydian Dorian Lydian

57 *ff* this passage uses the exact pitch-classes in the polymode below (hinting at a stacked C/F Lydian or A/D Dorian polymode): F Lydian Used on the final chord only D Dorian C Lydian

The majestic third verse is highly chromatic and frequently modulates or changes quickly from one chromatically related mode to another. It starts in 5/4 time (with long drawn-out bars and greater rhythmic irregularity than previously seen) and modal changes or modulations often occur on adjacent semiquaver, triplet or quintuplet beats. There are brief dominant-harmony substitution modes (namely the altered scale on D in m. 54 and the altered scale on B in m. 55) which add rhythmic as well as modal emphasis and m. 56 is perhaps the most chromatic bar in the entire movement.

Following this, m. 57 sees a similar idea to that previously discussed in mm. 46<sup>3</sup>–48<sup>3</sup>, where there are chromatic alternations between a modal set containing either F $\flat$  or F $\sharp$  respectively; where an F Lydian harmony is displaced either upwards or downwards by a G Ionian or E Aeolian-based harmonic block. Here, there are some changes between different harmonic blocks on the 32<sup>nd</sup>-note beat, but a polyphony reminiscent of patterns seen in madrigals or in some forms of folk music is retained. The polymode added below m. 57 is very similar to the pure-form stacked Dorian/Lydian modes previously illustrated except that what would be C $\sharp$ 5 has been replaced with C $\flat$ 5 and G $\sharp$ 5 has been replaced with G $\flat$ 5, thus the C Lydian portion of the polymode has been expanded. However, as is the case with the passage in mm. 46<sup>3</sup>–48<sup>3</sup>, every individual harmonic block contains pitch-classes from just one diatonic mode at a time; there are no simultaneous false relationships between say F $\flat$  and F $\sharp$  within an individual block, thus the passage merely hints at polymodality without actually being polymodal in a strict sense.

In mm. 58–59 there is a brief two bar recurrence of the initial atmospheric idea seen in mm. 1–8 and mm. 20–23 (transposed down a major fourth such that the alternation is now between C Lydian and A Lydian), before a 6/8 meter recurs in m. 60 and this final section of the third verse contains further hints at polymodality: in m. 60 there are alternations between C Lydian and D Ionian (emphasising a false relationship between C $\flat$  and C $\sharp$ ); in m. 62<sup>2</sup> (♩ beat) there is an alternation between A $\flat$  Lydian and B $\flat$  Lydian (emphasising false relationships between A $\flat$ /E $\flat$  and A $\sharp$ /E $\sharp$  respectively); similarly in m. 63<sup>1</sup> between E $\flat$  Lydian and F Lydian (emphasising E $\flat$ /B $\flat$  and E $\sharp$ /B $\sharp$ ); finally, in m. 65<sup>1</sup> there are alternations between F Aeolian/D $\flat$  Lydian and E $\flat$  Lydian (emphasising D $\flat$ /A $\flat$  and D $\sharp$ /A $\sharp$ ) before the section cadences chromatically on an A Lydian harmony.

The coda section in mm. 69–83 recapitulates material seen in the first verse, starting with dyads in major 7<sup>th</sup> intervals between the altos and tenors and then building harmonically in cannon form.

*Cwyn y Gwynt* for flute & harp (vol. 1, pp. 161–77) is based on a previous version of the same piece composed for alto voice, vibraphone and cello (also included in the portfolio: vol. 1, pp. 201–15). The original alto-voice version of this movement was set to the text from John Morris-Jones's Welsh-language poem, shown at the beginning of the score (vol. 1, p. 164). The three movement flute & harp version of this piece won the composer's medal ("Tlws y Cerddor") at the 2012 Welsh National



Eisteddfod in the Vale of Glamorgan, where the requirement was to compose a sonata for flute & harp. I felt that I could make a transcription/re-composition of the original alto-voice version work well for flute & harp (by occasionally adding some higher pitched flute material to the lower pitched song lines that already existed), but as the vocal part in the original version goes down to F#3, I needed to transpose it upwards so that it would be in appropriate range for the C-flute. This piece has also been well-received by contemporary flautist Carla Rees (who specialises in new repertoire for alto/bass flute) and plans are in place to transcribe the C-flute version for alto flute & harp so that it can be performed and recorded by the Rarescale contemporary music ensemble at some point in the near future.

As the harp can only be tuned to seven modal/chromatic pitches at any given time, it was necessary for me to rethink and scale-down much of the pitch material which had existed in the original version (where greater levels of harmonic/modal chromaticism were present in the original vibraphone line in particular), which would not have been possible to achieve in the same way on the harp. As a consequence, some of the polymodal elements of this piece are more explicit in the original alto-voice version and it was necessary for me to sit down and carefully consider how to translate the original harmonic language so that it would work as a harp accompaniment. The structure in the two different versions is pretty much identical, however the harmonic/chromatic language is rather different, thus I have appropriately described the flute & harp version as a re-composition of the original rather than a transcription.

The second movement of the flute & harp version (vol. 1, pp. 170–4) is composed in song structure, which corresponds to the song which was set to the text from the poem in the second movement of the original alto-voice version (as opposed to the first and third movements in the original version which are described as ‘abstracts’ and are both set to melismatic words which have been extracted from the original poem). When analysing the second movement of the flute & harp version one should not pay too much attention to the keys written below the harp part (i.e. D $\flat$  melodic minor, C $\flat$  major, etc..) as these are intended to represent harp tunings and make it easy for the harpist to check that the correct pedal configuration has been set when some sort of a conventional tuning is in operation. They *do not* represent the actual modes which are in operation musically.

The movement has a root mode of A $\flat$  Dorian and a key signature of six flats. This is the only movement within the portfolio to employ a key signature, but this has been done to make the accidental spellings in the harp line part easier to read (as the majority of tunings employ several flattened pitch-classes). In addition, flat tunings have a richer sound on the harp, hence my decision to employ a root mode of A $\flat$  Dorian (with six flats) rather than G $\sharp$  Dorian (with six sharps). The most prominent modes in operation in the movement are A $\flat$  Dorian, F $\flat$ /E Lydian and A Lydian, which frequently recur and there is

a melancholy relationship between the harmonies that are created from these three modes. The occasional flutter-tongues on minim beats in the flute part provide an effective timbral modification and add something of a melancholy edge to the flute melody – these characteristics correspond with the poetic representation of an elegy, tears, longing and despair which relates to the subjectivity of the poem, which this movement is based on.

A harmonic/modal progression model for the equivalent of the first verse in mm. 1–8 is provided overleaf (in **Fig. 2.2.1**). Most of this movement is modal (in a conventional sense), however, there are some clear pointers towards polymodality at times. The harmonies/modes employed in m. 3 are facilitated by chromatic harp tunings, with the tuning in m.  $3^{1-2}$  (♩ beat) of  $D\flat-C\flat-B\flat/E\flat-F\flat-G\flat-A\sharp$  used to enharmonically facilitate an A Lydian polychord (minus the  $B\flat$ , which is chromatic to this mode). The same tuning is also used to facilitate an altered scale on  $E\flat$  in m. 4, which serves as a dominant substitution for an interrupted cadence on an  $F\flat$  Lydian harmony, as indicated in the diagram. The tuning in m.  $3^{3-4}$  is used to facilitate a stacked E/D Lydian polychord, as demonstrated below with a logical enharmonic spelling in sharps and naturals, with the implied polymode on the left-hand side:

D Lydian [expanded into a compound major seventh]

A Lydian [expanded into a compound fifth]

enharmonic harp tuning:  
 $D_3, C\flat, B\flat, E\flat, F\flat, G\flat, A\sharp$

logical re-spelling of polychord in m. 3 (♩ beats 3–4)

melodic pitches (flute)

A Lydian upper-structure harp tetrachord

D major pentatonic lower-structure harp tetrachord

**Fig. 2.2.2**

The *lasciare vibrare* indication above the harp part in m. 5 is intended to imply that the harpist blends both polychords into one another, once again hinting at polymodality. A sum of the resulting polymodal pitch material in this measure is shown overleaf (in **Fig. 2.2.3**).

**Fig. 2.2.1:** harmonic/modal model for mm. 1–8 of 2. *Galargan* from “Cwyn y Gwynt” for flute & harp

**Slow, Melancholy, Sustained**  
 $\text{♩} = \text{c.40}$  or  $\text{♩} = \text{c.80}$

Fl.

mp

2

3

harmonies facilitated by chromatic harp tunings

mp

A $\flat$  Dorian F $\flat$  Lydian B $\flat$  Dorian G $\flat$  Lydian

altered scale on C  
[a mode of D $\flat$  melodic minor]

F $\flat$  Lydian

stacked E/D Lydian

A Lydian

4

5

A $\flat$ /G $\sharp$  adds a harmonic/modal pitch-class not present in the current harp tuning

A major/B [or C $\flat$ ] major upper-structure triads facilitated by a chromatic harp tuning

merge of sustained harp resonance creating a polymodal cluster

l.v. - - - - -

altered scale on E $\flat$   
[a mode of F $\flat$ /E $\sharp$  melodic minor]

F $\flat$  Lydian

A Lydian

interrupted A $\flat$  Dorian cadence on F $\flat$  major 13

6

7 flz.

8

5

5

5

G $\flat$  Ionian C $\flat$  Lydian A $\flat$  Dorian F $\flat$  Lydian D $\flat$  Dorian A Lydian A $\flat$  Dorian

The diagram illustrates a complex harmonic and modal analysis. At the top left, a musical staff shows an **E Ionian** scale (E-F#-G-A-B-C#-D#) in bass clef. A blue bracket below it indicates an **A Lydian [expanded into a compound major seventh]** interval. To the right, a red bracket highlights an **E Lydian** scale (E-F#-G#-A-B-C#-D#) in treble clef. Above this, a note states: **A♭/G♯ moved to the flute when A♭ is returned to A♮ on the harp**. Further right, a treble clef staff shows a **logical re-spelling of merged harp polychord in m. 5 [sum of both modal blocks]**, with labels for **melodic pitch-classes (flute)**, **E Lydian upper-structure harp cluster**, and **E Ionian or A Lydian lower-structure harp pentachord**. At the bottom, **enharmonic harp tunings** are shown for **D♭, C♭, B♭, E♭, F♭, G♭, (A♭-A♮)** on a staff with a rhythmic pattern below.

**Fig. 2.2.3**

The equivalent of the second verse in mm. 9–18 is based on similar harmonic material with arpeggiated ostinato figures in the harp, which use quaver rhythms with some triplets and quintuplets adding rhythmic variety and something of an ethereal feel, adding further poetic meaning to the music.

A further harmonic/modal progression model for the equivalent of the third verse in mm. 19–28 is provided overleaf:

**Fig. 2.2.4:** harmonic/modal model for mm. 19–28 of 2. *Galargan* from “*Cwyn y Gwynt*” for flute & harp

19 20 21

Fl. *pp e misterioso*

*p* *pp e misterioso*

A $\flat$  Dorian G $\flat$  Ionian B $\flat$  Phrygian A $\flat$  Dorian

bass pedal-point on C $\flat$   $\rightarrow$  8 $^{va}$  C $\flat$  Lydian (#5)  
[a mode of A $\flat$  melodic minor]

22 23 24

Fl. *cresc.*

upper-structure polychords or polymodes  $\rightarrow$  F $\flat$  major 11

F $\flat$ /E $\natural$  melodic minor

Common pitches in F/E $\flat$  Lydian

Common pitches in C/B $\flat$ /A $\flat$  Lydian

stacked D $\flat$  [or C $\sharp$ ] Dorian/  
F $\flat$  [or E $\natural$ ] melodic minor

stacked F/E $\flat$  Lydian

stacked C/B $\flat$ /A $\flat$  Lydian

doubled D $\flat$  major/E $\flat$  major  
upper-structure triads

(8) C $\flat$  Ionian (#5)  
[a mode of A $\flat$  harmonic minor]

(cresc.) altered scale on G $\flat$   
[a mode of A $\flat$ /G $\sharp$  melodic minor]

25 26

Fl. *mf e cresc.* *f*

*l.v. sempre*

*mf e cresc.* *f*

G Locrian F Dorian C Dorian D $\flat$  Lydian (#5)

27 28

Fl. *dim.* *mp*

*dim.* *mp e dim.* *pp*

G $\flat$  Lydian (#2) C $\flat$  Lydian B $\flat$  Dorian

G $\sharp$  adds a harmonic/modal pitch-class not present in the current harp tuning

The four chords in m. 19 are all modal clusters which belong to the same harp tuning, before a pedal point begins on C $\flat$ , using the lowest string on the harp. The bass pitches on the harp are then rhythmically suspended from this point onwards (occurring on the second and fourth beat in each bar, whilst the polychords in the mid–upper range occur on the first and third beats), however I have simplified the bass rhythms in the diagram to make the harmonic/modal progressions easier to follow. The polychords from m. 22<sup>3</sup>–23 are polymodal, as demonstrated below with logical enharmonic spellings and implied polymodes:

Figure 2.2.5 illustrates the harmonic structure of a musical passage. The main staff shows the E melodic minor scale (A Lydian [expanded into a compound major seventh]) and the E Lydian tetrachord. Below this, the enharmonic harp tuning is shown with notes D $\flat$ , C $\flat$ , B $\flat$ , E $\flat$ , F $\flat$ , G $\sharp$ , A $\sharp$ . To the right, a logical re-spelling of the polychord in m. 22 (beats 3–4) is shown, consisting of melody pitches from E melodic minor (flute), stacked E/A Lydian upper-structure harp tetrachord, E melodic minor mid-structure harp tetrachord, and a bass pitch from E melodic minor (low harp) marked 8<sup>vb</sup>.

Fig. 2.2.5

Figure 2.2.6 illustrates the harmonic structure of a musical passage. The main staff shows the E $\flat$  Lydian scale (F Lydian) and the C Lydian hexachord. Below this, the enharmonic harp tuning is shown with notes D $\sharp$ , C $\flat$ , B $\flat$ , E $\flat$ , F $\flat$ , G $\sharp$ , A $\sharp$ . To the right, a logical re-spelling of the polychord in m. 23 (beats 1–2) is shown, consisting of melodic pitches common to F/E $\flat$  Lydian (flute), A major pentatonic upper-structure harp trichord, E $\flat$  Lydian mid-structure harp tetrachord, and a bass pitch from E $\flat$  Lydian (low harp).

Fig. 2.2.6

The figure illustrates the construction of polychords in measure 23. It features several musical staves:

- A<sup>b</sup> Lydian** and **B<sup>b</sup> Lydian** modes shown as ascending scales.
- C Lydian trichord** shown as a three-note chord.
- enharmonic harp tuning:** A staff showing two rows of notes:  $D\sharp, C\flat, B\flat$  and  $E\flat, F\flat, G\sharp, A\flat$ .
- logical re-spelling of polychord in m. 23 (♩ beats 3–4):**
  - melodic pitches common to C/B<sup>b</sup>/A<sup>b</sup> Lydian (flute):** A staff showing the common melodic pitches.
  - G major pentatonic upper-structure harp tetrachord:** A staff showing the upper-structure tetrachord.
  - A<sup>b</sup> Lydian mid-structure harp tetrachord:** A staff showing the mid-structure tetrachord.
  - bass pitch from A<sup>b</sup> Lydian (low harp):** A staff showing the bass pitch.

**Fig. 2.2.7**

Both of the polychords in m. 23 (i.e. **Figures 2.2.6–7**) are based on the pitches obtained from pure-form stacked Lydian modes (stacking C/F/E<sup>b</sup> and C/B<sup>b</sup>/A<sup>b</sup> respectively). The polychord in m. 22<sup>3</sup> (i.e. **Fig. 2.2.5**) is modified from its pure form and has an E melodic minor portion at its bottom end.

In the previous progression model I have only shown the harmonic blocks in mm. 25–28 and avoided including the arpeggiated acciaccaturas in the harp part such as to make the harmonic/modal progressions easier to follow. Here each modal harmony is changed chromatically by modifying a single pitch-class in the harp tuning every two minim beats – This reaches a relatively bright harp tuning on C Dorian (with 5 naturals) in m. 26<sup>1–2</sup> before the passage cadences in B<sup>b</sup> Dorian in m. 28.

The final section in mm. 29–38 is the equivalent of a re-emphasis of the third verse (in terms of the words used in the original alto-voice version), but uses harmonic material similar to that in the first verse in mm. 1–8. The modal clusters in m. 29 are an exact transposition up a perfect fourth of the figure in m. 19. The harmonic progression from m. 30<sup>3</sup>–31 is a diminution of that seen in m. 1–3, with the modal harp tunings changing every minim beat rather than every two minim beats as previously seen. The final polychord in m. 31<sup>4</sup> is based on the same polymodal harp tuning as that seen in m. 3<sup>3–4</sup> and **Fig. 2.2.2** as shown overleaf:

The diagram illustrates the relationship between different musical scales and their re-spelling for a polychord in measure 31. It features three main musical staves:

- D Lydian:** A scale starting on D, with notes D, E, F#, G, A, B, C#, D. A blue bracket below it indicates the **A Lydian [expanded into a compound fifth]** interval.
- enharmonic harp tuning:** A staff showing the notes D#, Cb, Bb, Eb, Fb, Gb, Ab, A# in a chromatic sequence. Below this staff is a harp diagram with 12 strings, with the first seven strings marked with '+' signs.
- logical re-spelling of polychord in m. 31 (beat 4):** This section shows three stacked staves:
  - The top staff shows **melodic pitches (flute)** with notes D, E, F#.
  - The middle staff shows the **E major pentatonic upper-structure harp trichord** with notes E, G#, B.
  - The bottom staff shows the **stacked A/D Lydian lower-structure harp tetrachord** with notes A, C#, E, G.

Arrows indicate the mapping from the D Lydian and enharmonic harp tuning staves to the re-spelled polychord in measure 31.

**Fig. 2.2.8**

The colourful flute figure in mm. 30<sup>3</sup>–31 is an embellishment of previous melodic material, with little runs and shorter time values to those previously seen. The harmonies in mm. 32–35 are then similar to those seen in mm. 4–7 above a steady walking bass line in the harp before a short coda with further flute embellishment in mm. 36–38.

The second movement of the version for contralto, vibraphone & cello (vol. 1, pp. 209–12) has a modal root of C# Dorian and a greater level of chromatic modulation than that seen in the version for flute & harp. As an example of this, m. 1<sup>2–3</sup> modulates from A Lydian to F Dorian, which can easily be achieved on a fully chromatic polyphonic instrument such as a vibraphone or piano but cannot be facilitated instantaneously on the harp in the same way and would be a problematic scoring issue in that respect. The melancholy vocal line introduces extended techniques such as *sprechgesang*, *sprechstimme* and *glissandi*, which serve to further enhance the musical representation of despair relating to the poem that is set.

There are occasional passages in this movement which are explicitly polymodal: mm. 5–6 sees polychordal harmonic blocks between the vibraphone and the cello, following a dominant cadence using the altered scale on G# in m. 4, as shown overleaf:



harmonic/modal model for mm. 4–6 of 2. *Galargan* from “*Cwyn y Gwynt*” for contralto, vibraphone & cello

**Lento Sostenuto**  
♩ = c.50

Alto  
Vib.  
Vc.

altered scale on G#  
[a mode of A melodic minor]

interrupted C# Dorian cadence on a polychord  
with a root mode of A Lydian

stacked G#/E/A Lydian

stacked E/D/G Lydian

stacked B Lydian/  
F# Dorian

stacked A Lydian/  
E Dorian

Fig. 2.3.1

This passage is set to the words “Wrth fy ffenestr yn gwynfannus” (“At my window restlessly”) and the switch into polymodality in this passage is intended to be a symbolic representation of the wind against the window. Another prominent harmonic feature demonstrated here is the idea of shifting a polymodal harmonic block down by a whole-tone, as seen between the two polychords in m. 5 and the two polychords in m. 6 above. There is further polymodality in mm. 15–16. An illustration of the modal/pitch collections in use in these two measures is shown below:

modal/pitch collections used in mm. 15–16 of 2. *Galargan* from “*Cwyn y Gwynt*” for contralto, vibraphone & cello

(Sprechgesang)

Alto  
Vib.  
Vc.

belonging to F# Dorian

belonging to E Dorian

stacked C#/F# Dorian polymodal block:  
false relationship between A#/A#

stacked B/E Dorian polymodal block:  
false relationship between G#/G#

Fig. 2.3.2

This passage is set to the words “Ar y gwydr yr hyrddia’i ddagrau” (“On the glass it hurls its tears”), once again using polymodality to emphasise the poetic symbolism which exists between the representation of rain/water on a glass/window pane and tears/despair. On this occasion, a stacked Dorian harmonic block is shifted down by a whole-tone between the two measures.

Finally, the movement's climax occurs in mm. 24–28, where there is a more extensive polymodal passage. The text that this passage is set to finally reveals that the despair of poem's subject is related to the death of a loved one, which is further represented musically by the use of consecutive explicit polymodal blocks in mm. 25–27. A dense, and atmospheric musical texture is generated here by using the sustaining pedal on the vibraphone to stack chromatically related polychords on top of one another. A modal/harmonic model showing the polychordal breakdown between voice, vibraphone and cello is shown overleaf. All of the individual harmonic blocks in mm. 25–27 consist of 9, 10, 11 or 12 separate pitch-classes and bear a similarity to several of the row-based polychords illustrated in **Figures 1.3.1–11** in the first chapter.

The movement ends above a pedal-point on a low C# in the cello. There are polymodal harmonic blocks above this in the vibraphone part in mm. 33–34 (equivalent to those seen in mm. 5–6) and then further modal (non-polymodal) harmonic blocks above the pedal-point in mm. 35–38, where the movement ends. In this final section, the pitch material in the vocal line is modally related to that which occurs in the vibraphone accompaniment.

**Fig. 2.3.3:** harmonic/modal model for mm. 24–28 of 2. *Galgargan* from “*Cwyn y Gwynn*” for contralto, vibraphone & cello

Alto

Vib.

Vc.

24

25

10 chromatic pitch-classes

10 chromatic pitch-classes

(Sprechgesang)

*mp*

*p*

*mp*

upper-structure tetrachords

B Lydian tetrachord

trichord belonging to G Lydian

trichord belonging to F Lydian

A Lydian tetrachord

C major triad

B♭ major triad

altered scale on B [a mode of C melodic minor]

stacked B/G/C Lydian

stacked A/F/B♭ Lydian

interrupted E Dorian cadence on a polychord with a root mode of C Lydian

polyharmonic block transposed down by a whole-tone

Alto

Vib.

Vc.

26

27

11 chromatic pitch-classes

12 chromatic pitch-classes

9/10 chromatic pitch-classes

9 chromatic pitch-classes

chromatic auxiliary grace-note

(Singing Voice)

belonging to C♭/B Lydian

belonging to B♭ Lydian

F Lydian trichord

E Lydian hexachord

F major pentatonic

F Lydian trichord

D Lydian pentachord

A♭ Lydian

F Dorian or D♭ Lydian tetrachord

B Lydian tetrachord

A major pentatonic trichord

(arco)

C Lydian pentachord

B♭ major pentatonic

stacked E/F Lydian

stacked D/E♭/A♭ Lydian

stacked D♭/B/A Lydian

stacked C/B♭ Lydian

polymode transposed down by a whole-tone and a perfect fifth added in the bass

root mode transposed upwards by three adjacent semitones

Alto

Vib.

Vc.

28

7 modal pitch-classes

6 modal pitch-classes

F Lydian (♭7) [a mode of C melodic minor or the altered scale on B]

E Lydian

modal block transposed down by a semitone and (♭7) modal element is neutralised



# CHAPTER 3

## Explicit Polymodality

## Explicit Polymodality

Much of the previous chapter has illustrated how works in my portfolio, which have been composed in a predominantly modal style, are either occasionally hinting at polymodality or contain occasional passages which are explicitly polymodal. This chapter will now take a closer look at works within the portfolio which are set in a polymodal style “where the individual modes which form each polymodal or polyharmonic block are transparent and can clearly be related back to a more conventional modal style of writing”, described as ‘explicit polymodality’ in the first chapter (vol. 2, p. 10, with the pitch organisation chart on p. 11 categorising several work within the portfolio in this category).

The two songs for soprano, flute/piccolo & pianoforte set to words by Sir John Morris-Jones (vol. 1, pp. 217–51) are, perhaps, some of the earliest examples of explicit polymodality in my portfolio. At this point in my development as a composer I was beginning to devise ways of linking modality with polymodality and wanted to experiment with various ways of making this possible in a pragmatic fashion.

“*Iâr fach yr haf*” (“*The Butterfly*”, vol. 1, pp. 222–36) is set to three verses and adopts an  $A_1A_2B_1B_2A_3A_4$  structure, where each respective verse is repeated once –  $A_1$ ,  $B_1$  and  $A_3$  are the initial settings of each respective verse, while  $A_2$ ,  $B_2$  and  $A_4$  are embellished settings with melismatic runs in the soprano line the second time around:  $A_1A_2$  (mm. 1–13 and 14–27 respectively) are both settings of the first verse with a modal root of F Lydian;  $B_1B_2$  (mm. 28–40 and 41–53) are both settings of the second verse with a modal root of D Lydian;  $A_3A_4$  (mm. 54–70 and 71–92) are both settings of the third verse,  $A_3$  has a modal root of B $\flat$  Lydian (following an initial bridge section in mm. 54–57) and  $A_4$  starts with a modal root of A Aeolian before working its way back to its modal equivalent of F Lydian.

The song begins with the 4-measure flute and piano sequence shown in the harmonic/polymodal model overleaf:

harmonic/polymodal model for mm. 1–4 of *lâr fach yr haf* for soprano, flute & pianoforte

**Leggiero Sempre**  
♩ = c.112

Flute

1 2

E major pentatonic

G major pentatonic

F major pentatonic

A major pentatonic

E major pentatonic

3 4

A major pentatonic

G major pentatonic

F major pentatonic

A major pentatonic

G $\flat$  major pentatonic

belonging to G $\flat$  Lydian

stacked G/F Lydian

Fig. 3.1.1

An ostinato pattern is formed, which juxtaposes different modally or chromatically related pentatonic blocks. Initially, an F Lydian texture is created by juxtaposing F major pentatonic block chords in the left hand with a broken G major pentatonic pattern in semiquavers in the right hand. The left hand also alternates to and from A major pentatonic on the off-beat to add chromatic/rhythmic emphasis and displaces its F major root downwards by a semitone on E major pentatonic at the end of m. 2 and upwards by a semitone on G $\flat$  major pentatonic at the end of m. 4. The flute line complements the E major pentatonic portion at the end of m. 2 before a sequence based on a triplet rhythm using A major pentatonic in mm. 3–4, where the resultant superposition of A/G/F major pentatonic scales equates to a pure-form stacked G/F Lydian polymodal block.

The vocal line then enters at the end of m. 4 with a G major pentatonic-based melody and the circular ostinato accompaniment relates to the poem and symbolises the butterfly which “hovers around” (“hofran o’i gylch”) the rose. The ostinato sequence is then transposed up a whole-tone in mm. 9–10 (such that the modal root becomes G Lydian) and up a further tritone in m. 11 (such that the modal root becomes D $\flat$  Lydian) before returning chromatically to F Lydian in mm. 12–13. A similar ostinato pattern is then seen from m. 14 onwards, with the piano line scored in the instrument’s upper register with the previous modal blocks slightly re-ordered and quicker broken chords in the left hand. Following the first melismatic vocal section, the first verse ends on a G $\flat$  Lydian root, which juxtaposes falling/rising G $\flat$ /A $\flat$  major pentatonic blocks in m. 25. There is, however, a polymodal twist in m. 26–27: the flute line sees the inclusion of G $\sharp$  on the trill in m. 26 and the E $\flat$  major pentatonic figure in m. 27, which hints at a stacked D $\flat$ /G $\flat$  Lydian polymodal block with a false relationship between G $\sharp$ /G $\flat$ .

The second verse starts in m. 28. A harmonic/modal model for mm. 28–31 is shown below:

harmonic/modal model for mm. 28–31 of *lâr fach yr haf* for soprano, flute & pianoforte

**Un poco meno mosso**  
♩ = c.96

Fl. 28 29 30 31

pp

D Lydian altered scale on D

G Lydian G $\flat$  Lydian F Lydian E Lydian

Lydian modes falling chromatically

B Lydian pentachord

A Lydian

stacked B/A Lydian

belonging to the altered scale on E

altered scale on E above an A $\flat$  pedal

A Dorian A Lydian

Fig. 3.1.2

If the piano line is treated in isolation in the above passage, all of its individual harmonic blocks are modal. The flute does, however, introduce a polymodal element as seen in m. 30, where a B Lydian pentachord is stacked above an A Lydian harmony. A harmonic/polymodal model for mm. 36–44 is shown overleaf:



**Fig. 3.1.3:** harmonic/polymodal model for mm. 36–44 of *Iâr fach yr haf* for soprano, flute & pianoforte

The score is divided into three systems, each with staves for Soprano (Sop.), Flute (Fl.), and Piano (Pn.).

**System 1 (mm. 36-37):**

- Sop.:** Lyrics: "e - os a gân ei mel - ys".
- Fl.:** Melody with *mf* dynamic. Boxed section (mm. 36-37) is labeled **B $\flat$  major pentatonic**.
- Pn.:** Accompaniment with *mf* dynamic. Boxed section (mm. 36-37) is labeled **A Aeolian**. A separate box (mm. 36-37) is labeled **F Mixolydian**. A red box (mm. 36-37) is labeled **E $\flat$  Mixolydian**. A blue box (mm. 36-37) is labeled **stacked E $\flat$ /A $\flat$  Lydian**.

**System 2 (mm. 38-40):**

- Sop.:** Lyrics: "taw - el ser - en yr hwyr?".
- Fl.:** Melody with *pp* dynamic. Boxed section (mm. 38-40) is labeled **D Lydian**. A blue box (mm. 38-40) is labeled **stacked E/D Lydian**.
- Pn.:** Accompaniment with *ppp* dynamic. Boxed section (mm. 38-40) is labeled **stacked E/D Lydian**.

**System 3 (mm. 41-44):**

- Fl.:** Melody with *pp* dynamic. Boxed section (mm. 41-44) is labeled **B Lydian pentachord**. A blue box (mm. 41-44) is labeled **belonging to the altered scale on E**.
- Pn.:** Accompaniment with *pp* dynamic. Boxed section (mm. 41-44) is labeled **stacked A/D Lydian**. A blue box (mm. 41-44) is labeled **altered scale on D**. A red box (mm. 41-44) is labeled **stacked Lydian modes falling chromatically**. A blue box (mm. 41-44) is labeled **stacked B/A Lydian**. A red box (mm. 41-44) is labeled **altered scale on E above an A $\sharp$  pedal**. A blue box (mm. 41-44) is labeled **A Dorian**. A red box (mm. 41-44) is labeled **stacked E/A Lydian**.

The embellished broken chord figures in the flute and piano lines in mm. 36–37 represent the “singing Nightingale” with its ‘sweet melody’ (“eos a gân ei melys gainc”). In m. 37<sup>2</sup> (♩ beat), the stacking of E♭ Lydian in the upper range of the piano line above A♭ major pentatonic figures in the soprano and flute line equates to a stacked E♭/A♭ Lydian polymodal block; however, the former polymodal block in m. 37<sup>1</sup> stacks E♭ Mixolydian in the upper range of the piano line above B♭ major pentatonic in the soprano and flute line – which does not relate closely to the pure-form polymodal models discussed in **Figures 1.2.1–10**. This is explained by the fact that at this stage in the development of my polyomodal methods I had not formulated a clear singular system for stacking polymodes and was experimenting with various differing stacked polymodal structures, such as that in m. 37<sup>1</sup>.

The passage in mm. 38–40 sees a sudden change in musical temperament, to a *pp* dynamic with delicate and gentle harmonic writing, this time representing the “silent Evening Star” (“tawel seren yr hwyr”). Here, a D Lydian modal block in mm. 38–39 is modulated into a stacked E/D Lydian polymodal block in m. 40 via the inclusion of a polymodal figure which emphasises the false relationship between A♯/A♭ in the flute line.

The B<sub>2</sub> section begins in m. 41 and the predominantly modal accompaniment previously seen at the start of the B<sub>1</sub> section (in **Fig. 3.1.2**) has now been thickened up and turned into a group of stacked polymodal harmonic blocks, as shown at the bottom of **Fig. 3.1.3**. These thicker harmonic layers continue above a melismatic repetition of the second verse in the soprano line and there is further emulation of birdsong and the nightingale call in mm. 49–50 with question/answer type dovetails between the melisma/coloratura-like figures in the soprano and similar regular/quintuplet semiquavers in the flute line, above modal 8/9-note block chords in the upper register of the piano.

The A<sub>3</sub> section begins with the 4-measure bridge section shown in the harmonic/polymodal model overleaf:

harmonic/polymodal model for mm. 54–57 of *Iâr fach yr haf* for soprano, flute & pianoforte

Leggiero (Tempo primo)

♩ = c.112

The musical score is divided into two systems, measures 54-55 and 56-57. The instruments are Soprano (Sop.), Flute (Fl.), and Piano (Piano). The tempo is Leggiero (Tempo primo) with a quarter note equal to approximately 112 beats per minute.

**Measure 54:**

- Soprano:** Melody starting on 'a' (A4), marked *mp*. A blue box labeled 'G major pentatonic' covers the first four notes.
- Flute:** Melody starting on 'a' (A4), marked *p*. A blue box labeled 'D major pentatonic' covers the first four notes.
- Piano:**
  - Right hand: Chordal blocks marked in red. One is labeled 'belonging to E Lydian' and another 'A major pentatonic'. A blue box labeled 'stacked G/C Lydian' covers the first four notes.
  - Left hand: Chordal blocks marked in red. One is labeled 'E major pentatonic' and another 'C major pentatonic'. A blue box labeled 'stacked G/C Lydian' covers the first four notes.

**Measure 55:**

- Soprano:** Melody starting on 'a' (A4), marked *mp*. A purple box labeled 'D# minor pentatonic' covers the last four notes.
- Flute:** Melody starting on 'a' (A4), marked *p*. A purple box labeled 'D# minor pentatonic' covers the last four notes.
- Piano:**
  - Right hand: Chordal blocks marked in red. One is labeled 'D minor pentatonic' and another 'altered scale on B'. A purple box labeled 'altered scale on B#/' covers the last four notes.
  - Left hand: Chordal blocks marked in red. One is labeled 'altered scale on B' and another 'altered scale on B#/'.

**Measure 56:**

- Soprano:** Melody starting on 'a' (A4), marked *mp*. A blue box labeled 'G major pentatonic' covers the first four notes.
- Flute:** Melody starting on 'a' (A4), marked *p*. A blue box labeled 'G major pentatonic' covers the first four notes.
- Piano:**
  - Right hand: Chordal blocks marked in red. One is labeled 'belonging to A Lydian' and another 'D major pentatonic'. A blue box labeled 'stacked C/F Lydian' covers the first four notes.
  - Left hand: Chordal blocks marked in red. One is labeled 'A major pentatonic' and another 'F major pentatonic'. A blue box labeled 'stacked C/F Lydian' covers the first four notes.

**Measure 57:**

- Soprano:** Melody starting on 'a' (A4), marked *mp*. A purple box labeled 'G# minor pentatonic' covers the last four notes.
- Flute:** Melody starting on 'a' (A4), marked *p*. A purple box labeled 'G# minor pentatonic' covers the last four notes.
- Piano:**
  - Right hand: Chordal blocks marked in red. One is labeled 'A minor pentatonic' and another 'altered scale on F#'. A purple box labeled 'altered scale on E#/' covers the last four notes.
  - Left hand: Chordal blocks marked in red. One is labeled 'altered scale on F#' and another 'altered scale on E#/'.

Red arrows point from the red chordal blocks to the text 'chromatic/dissonant rhythmic anticipations' at the bottom of each system.

Fig. 3.1.4

As shown, the chordal blocks marked in red are chromatic/dissonant rhythmic anticipations which, do not form part of the wider harmonic block in each respective measure. In m. 56 the rhythmic alternation from an A Lydian block to a stacked C/F Lydian block in the piano part corresponds to the alternations between A major pentatonic and F Lydian in the left hand of the piano part in m. 1 (as illustrated in **Fig 3.1.1**) and the same idea is seen transposed up a perfect fifth in m. 54. In m. 55 the altered scale on B#

block is displaced downwards by a semitone in the altered scale on B $\natural$  block in red and the D $\sharp$  minor pentatonic block is an upper-structure collection extracted from the altered scale on B $\sharp$ . This idea is transposed down a perfect fifth in m. 57, except that the altered scale E $\sharp$  block is displaced upwards (rather than downwards) by a semitone in the altered scale on F $\sharp$  block in red.

The remainder of the A<sub>3</sub> section, in mm. 58–70 is a recapitulation of the A<sub>1</sub> section (mm. 1–13) transposed up a perfect fourth and set to words from the third and final verse, with a modal root of B $\flat$  Lydian and double-tongued figures in the flute line to add timbral variety.

The A<sub>4</sub> section begins in m. 71 with a brisk ostinato figure in sextuplets in the piano part; this alternates between A Aeolian and a C $\sharp$  Aeolian modal blocks (relating closely to the alternation between F major pentatonic and A major pentatonic in m. 1 [i.e. F major pentatonic and A major pentatonic are collections from A Aeolian and C $\sharp$  Aeolian respectively]). In addition, the G $\sharp$  Aeolian block on the final crotchet beat of m. 72 relates closely to the E major pentatonic block on final crotchet beat of m. 2 and the B $\flat$  Aeolian block on the final crotchet beat of m. 74 relates closely to the G $\flat$  major pentatonic block on the final crotchet beat of m.4. When the soprano enters, there are some transparent false relationships which exist between the F $\sharp$  pitches in the vocal line at the start of m. 76 and m. 78 and the A Aeolian texture in the flute/piano (containing or implying F $\flat$ ).

The third verse comes to a head on the altered scale on C block (below a G $\flat$  major pentatonic upper-structure) in m. 81 and then mm. 82–83 sees a brief recapitulation of the “silent Evening Star” passage seen at the end of sections B<sub>1</sub> and B<sub>2</sub>, this time modulated to the song’s root mode of F Lydian. Following this the final two lines of the poem are repeated above the flute/piano accompaniment figure seen at the start of B<sub>1</sub> and B<sub>2</sub> (in mm. 84–85), before a further altered scale on C block in mm. 86–87 acts as a dominant cadence on F Lydian (once again stacking G/F major pentatonic) in the final five measures of the song.

“*Y Gwylanod*” (“*The Seagulls*”, vol. 1, pp. 238–51) is similarly set to three verses and adopts an A<sub>1</sub>A<sub>2</sub>B<sub>1</sub>B<sub>2</sub>A<sub>3</sub>A<sub>4</sub> structure, with each respective verse repeated/embellished once: A<sub>1</sub>A<sub>2</sub> (mm. 1–16 and 17–32 respectively) are both settings of the first verse and have a somewhat ambiguous modal root of F $\sharp$  Dorian; B<sub>1</sub>B<sub>2</sub> (mm. 33–46 and 47–60) are both settings of the second verse with a modal root of E Lydian; A<sub>3</sub>A<sub>4</sub> (mm. 61–77 and 78–95) are both settings of the third verse, returning to a modal root of F $\sharp$  Dorian, which is great deal more transparent in the cadences at the end of A<sub>3</sub> and A<sub>4</sub> respectively.

As with *Iâr fach yr haf*, *Y Gwylanod* is constructed using a stylistic mix of both modal and explicitly polymodal writing. There are frequent chromatic modal modulations throughout (sometimes on the off-beat or on irregular beats within a measure) and short musical gestures are used symbolise the

words in the poem. The most common time signature in the first and third verse ( $A_1A_2$  and  $A_3A_4$  respectively) is 6/8, but there are also several time signature changes to 4/8, 7/8 and 3/8 to create metric irregularity. The second verse ( $B_1B_2$ ) modulates to 8/8 (equivalent to 4/4 or 2/2), but retains some changes to 7/8 and 3/8.

A harmonic/polymodal model of mm. 1–4 is shown below:<sup>1</sup>

The figure shows a musical score for measures 1 through 4, featuring a Piccolo (Picc.) and Piano (Pfte.) part. The tempo is marked 'Colourfully, ♩ = c.156' and 'always' with a note symbol. The score is annotated with various modal and harmonic blocks:

- Measure 1:** The Piccolo line starts with a note 'belonging to F# Dorian'. The Piano part has a 'D Locrian' block (first two quavers), a 'D Aeolian' block (third quaver), and an 'F# Ionian' block (fourth and sixth quavers).
- Measure 2:** The Piccolo line is part of a 'stacked D/G Lydian' block, which is 'belonging to D Lydian'. The Piano part has a 'G major pentatonic' block.
- Measure 3:** The Piccolo line has an 'E Dorian' block (first two quavers, marked in blue) and an 'E# Ionian' block (next four quavers, marked in red). The Piano part has a 'Dorian' block (first two quavers, marked in blue) and a 'Lydian' block (next four quavers, marked in red).
- Measure 4:** The Piccolo line is part of a 'stacked C/B#E#D# Lydian' block, which is 'belonging to C Lydian'. The Piano part has a 'G' block (first two quavers, marked in blue) and an 'E# D# Lydian' block (next four quavers, marked in red).

Fig. 3.2.1

The first measure uses modal blocks and quite clearly changes mode chromatically on its third, fourth and sixth quaver beat respectively. Following this, the next three measures all possess polymodal characteristics: m. 2 points towards a pure-form stacked D/G Lydian; the first two quaver beats in m. 3 (marked in blue) hint at a stacked C#/B/E Dorian polymodal block and the next four quaver beats (marked in red) hint at a stacked F/E#A# Lydian block (with the D Dorian portion being closely related to F Lydian); m. 4 hints at a stacked C/B#E#D# Lydian (with the G Dorian portion being closely related to B# Lydian).

A harmonic/polymodal model of mm. 33–39 is shown overleaf. This diagram relates to the beginning of the second verse, where harmonic blocks tend to change on regular minim beats, rather than on irregular beats as seen in the first verse. The piano reduction shows the harmonic changes on each minim beat as block chords (rather than the arpeggiated figures in the full score) in order to make the polymodal relationships easier to decipher. The rapid 16<sup>th</sup> and 32<sup>nd</sup>-notes in the piano in mm. 35–39 symbolise words in the poem which describe the seagulls' wings "Spinning swiftly" ("Troelli'n ebrwydd ar yr adain"). There are interesting polymodal stackings of Aeolian and Ionian modes (in m. 33 and m. 38 respectively), which deviates from the more commonly seen stacking of Lydian/Dorian modes in my

<sup>1</sup> When analysing the polymodal stacking in operation in *Y Gwylanod*, it is important to bear in mind that the piccolo line will sound an octave higher than notated in the score. However, it should also be noted that the piccolo line is written at actual pitch in some of the analytical diagrams shown in this chapter (as specified).

**Fig. 3.2.2:** harmonic/polymodal model for mm. 33–39 of *Y Gwylanod* for soprano, piccolo & pianoforte

**Sustained & less articulate, but  $\text{♩}=\text{♩}$**

The score is divided into three systems, each corresponding to a measure of music. The Soprano part has lyrics in Welsh. The Piccolo part is marked with dynamics like *pp* and *mp*. The Piano part is marked with dynamics like *pp* and *mp*. The analysis includes various musical notations such as chords, intervals, and mode labels.

**System 1 (mm. 33-35):**

- Measure 33:** Piccolo has a trill. Piano has a stacked C#/F# Aeolian chord. Labels: *pp*, E major 7<sup>th</sup>, D major 7<sup>th</sup>, stacked C#/F# Aeolian.
- Measure 34:** Piccolo has a trill. Piano has a stacked C/Bb Lydian chord. Labels: *pp*, 8va, Bb Lydian, stacked C/Bb Lydian.
- Measure 35:** Piccolo has a trill. Piano has a stacked Bb Lydian chord. Labels: *mp*, Bb Lydian, stacked Bb Lydian.

**System 2 (mm. 36-38):**

- Measure 36:** Soprano: Troe - lli'n. Piccolo: (tr). Piano: stacked Bb/Eb Lydian. Labels: *mf*, Eb Lydian dyad, belonging to Bb Lydian, stacked Bb/Eb Lydian.
- Measure 37:** Soprano: eb - rwydd. Piccolo: (tr). Piano: stacked A/D Lydian. Labels: D Lydian trichord, belonging to A Lydian, D Lydian, stacked A/D Lydian.
- Measure 38:** Soprano: Wnaeth yr. Piccolo: (tr). Piano: stacked Db/Gb Lydian. Labels: *p*, Eb Lydian, stacked Db/Gb Lydian.

**System 3 (mm. 39-41):**

- Measure 39:** Soprano: a - dar. Piccolo: (tr). Piano: stacked Ab/F# Ionian. Labels: *p*, F# Ionian trichord, belonging to Ab Ionian, F# Ionian, stacked Ab/F# Ionian.
- Measure 40:** Soprano: llwyd - ddu. Piccolo: (tr). Piano: stacked D# Phrygian (b4). Labels: *p*, D# Phrygian (b4), stacked D# Phrygian (b4).
- Measure 41:** Soprano: hyn;. Piccolo: (tr). Piano: stacked E Lydian. Labels: *p*, E Lydian, stacked E Lydian.

**Additional Labels and Annotations:**

- Measure 33:** A Locrian, quartal harmony.
- Measure 34:** F# Dorian or Aeolian hexachord.
- Measure 36:** Eb Lydian.
- Measure 37:** D Lydian.
- Measure 38:** Gb Lydian.
- Measure 39:** F# Ionian.
- Measure 40:** D# Phrygian (b4) [a mode of B major (b6)].
- Measure 41:** E Lydian.

style. There is also a 2+2e-cell tetrachord on F in m. 34, with this song illustrating some of the earliest examples of intervallic cells being used in this way in my work.

A colourful return to the harmonic/melodic material seen in the first verse occurs in m. 61, with the piano line adopting broken chords which alternate between both hands in a percussive fashion. The first clear sight of the song's stacked G#/F# Dorian root is revealed in the cadence which takes place in mm. 75–77, as illustrated in the figure below:

The figure shows a musical score for measures 75, 76, and 77. The top staff is for Soprano (Sop.) with a *ff* dynamic. The second staff is for Piccolo (Picc.) with a *f* dynamic in measure 75 and a *pp* dynamic in measure 77. The bottom two staves are for Piano (P). The piano part features a 'stacked G#/F# Dorian polychord' in measures 76 and 77, indicated by a dashed box and a *mp* dynamic. A 'C# Phrygian (>4) [a mode of A major (>6)]' block is shown in measure 75. A 'C# Dorian hexachord' is indicated above the Soprano staff in measure 75. The piano part in measure 75 has a *f* dynamic.

Fig. 3.2.3

The C# Phrygian (b4) block serves as a dominant harmony in the cadence, however the high D# in the piccolo is interestingly chromatic to the corresponding dominant harmonic block below it. A similar cadence occurs in the song's final measures in mm. 92–95 with the harmonic blocks slightly thickened and re-voiced from that seen in the figure above and natural harmonics employed in the piccolo line in mm. 93–95. There is a further interesting polymodal block in m. 79, which sees a 3+2a-cell pentachord on D in the piccolo line, as shown below:

The figure shows a musical score for measure 79, labeled '(Very Colourful)'. The top staff is for Piccolo (Picc.) with a *p* dynamic. It features a '3+2a-cell pentachord on D' highlighted in a blue box, with a red line indicating an 'E Lydian' mode. The bottom two staves are for Piano (Pfte.). The piano part features a 'stacked E/D/G Lydian' block, with 'G Lydian' and 'D Lydian' modes indicated by purple and blue boxes respectively. The piano part includes a triplet of eighth notes marked with a '3' and a *p* dynamic.

Fig. 3.2.4





# CHAPTER 4

## Electronic & Computer-Generated Sonorities

# **Electronic & Computer-Generated Sonorities**

The project aims for this PhD (which are outlined on pp. 6–8 of vol. 2) explain how one of the aims is to investigate the possibility of incorporating extended timbre, colouration and electronics into a style of composing which remains inherently modal or polymodal. Prior to beginning the PhD I had ample experience in composing for instrumental forces and had gained experience of composing musique concrète and acoustmatic music (electroacoustic music for pre-recorded tape), whilst completing a master's degree at Birmingham University from 2001–3 under the supervision of electroacoustic composers Jonty Harrison and Erik Oña. Following this, I had also gained experience and familiarity with many of the principles of commercial audio engineering whilst producing commercial and underground styles of music in a freelance capacity following the completion my master's degree.

For the PhD project, I wanted to include at least one piece in the portfolio of compositions which merged live acoustic performance with electronics (through the use of live processing, sound synthesis, sampling and stochastic<sup>1</sup> procedures), as a means of increasing the timbral and sonic possibilities available to me as a composer. In order for me to be able to achieve this, it was necessary for me to choose a modern music/audio-specific computer programming language to use to develop the interfaces required for this new piece. It was then also necessary for me to learn the language chosen to an advanced level as well as gaining a technical understanding of the core principles of modern computer programming and software development.

In addition to my interests as a composer, I also have both an academic and practical background in mathematics, physics, computing and sound engineering. I wanted to develop this aspect of my knowledge alongside the completion of this PhD and integrate these additional skills into the formation of a new piece of 'live' electroacoustic music to include in the portfolio. During the course of the PhD, I also spent 18 months taking formal qualifications at night school in a variety of generic (non-musical) computer programming languages, namely C, C++, Java, Perl and UNIX shell-scripting and through this, I was also awarded the D. E. Evans Prize for outstanding work on the advanced-level C-programming course in 2013.

I have a strong interest in the current activities of various centres of excellence in electronic/computer music, in particular at BEAST (Birmingham Electroacoustic Sound Theatre, Birmingham University, which I worked in close collaboration with during my master's degree from 2001–3), IRCAM (Institut de Recherche et Coordination Acoustique/Musique, Paris) and CCRMA

---

<sup>1</sup> As pertaining to some of the random and stochastic generative procedures described in Iannis Xenakis' *Formalized Music: Thought and Mathematics in Composition* and relating to similar techniques adopted in Xenakis' electronic and/or computer-generated music compositions.

(Center for Computer Research in Music and Acoustics, Stanford University, USA). A variety of music/audio-specific computer programming languages are popular at these institutions, with three of the most commonly used being MAX/MSP, Pure Data (Pd) and SuperCollider.

The MAX/MSP audio signal-processing language (currently maintained by Cycling 74 and originally developed by Miller Puckette) has become very popular among composers of 'live' electronic music which requires real-time performance interfaces. I believe that this is largely due to the fact that it is a powerful and versatile programming language which runs in a graphical environment, thus it is intuitive to use and does not necessarily require conventional script-based programming skills, which traditionally have a steep learning curve and require high levels of mathematical precision.

I completed a MAX/MSP summer school course in 2010 at Goldsmith's University and experimented with this language, however, I eventually decided against using it for the new electroacoustic piece which I wanted to work on because I found that creating algorithms with high levels of mathematical complexity, although possible, can be problematic in MAX/MSP, especially when high levels of accuracy and efficiency are required. Therefore due to the skills that I had developed in script-based programming I felt that I would prefer to use a script-based programming language of some description. That said, I found that MAX/MSP is actually very useful for creating relatively sophisticated algorithms in a short space of time. This is especially useful for prototyping and for trial and error testing purposes.

Following further research, I discovered that CCRMA at Stanford University offer an open-sourced distribution (i.e. publicly owned, freely distributed) known as Planet CCRMA, that mirrors the systems which are configured at their institution in Stanford University and is built on a powerful open-sourced operating system called Linux Fedora (maintained by Red Hat – this differs from the Apple-Mac or Microsoft Windows-based systems frequently adopted by the majority of professional electroacoustic composers and music industry professionals). The Planet CCRMA distribution, which runs on Fedora (also known as Planet Fedora) modifies the operating system configuration so that it is optimised correctly for use with real-time audio and also provides a variety of audio-specific programming languages. One such language is Pure Data (or Pd), which is an open-sourced equivalent to MAX/MSP (currently maintained by the open-sourced community, but also originally developed by Miller Puckette; another is SuperCollider (developed by James McCartney, also maintained by the open-sourced community), which is a fully expressive script-based music synthesis language and has become very popular over recent years with sound synthesis specialists and post-Xenakisian<sup>2</sup> computer music specialists, as well as engineers and scientists who work with spectral analysis.

---

<sup>2</sup> Ibid.

I liked the mathematical precision which SuperCollider allowed me to work with and also liked the fact that it is a script-based language (which was familiar to me because of the background which I had developed in generic script-based languages such as C, C++ and Java during the course of my PhD), rather than a graphical programming language (as is the case with MAX/MSP and Pure Data). Despite being script-based, SuperCollider is also an interactive language to work with and allows the programmer to run or sonically ‘audition’ a program or part of a program without needing to compile it first (as is the case with most generic programming languages and can be rather time-consuming when extensive testing is required).

SuperCollider also allows the programming of hundreds (or even thousands) of efficient and complex synthesizers simultaneously – this can be done either manually with great accuracy, or via the use of a pre-programmed automated function which can be written to randomly or stochastically<sup>3</sup> generate each individual synthesizer in a fraction of a second! A graphical programming language such as MAX/MSP or Pure Data would struggle to allow the programmer to work in this way and it is a highly effective way of allowing a programmer or composer to synthesize complex, evocative and atmospheric sound-masses or clusters of sound. I believe that this a strong factor that has lead to the SuperCollider language becoming popular amongst post-Xenakisian composers in recent years.

Following careful consideration, I felt that SuperCollider would be an appropriate language for me to use to develop my real-time performance interfaces and felt that it would allow me to work in exactly the way I wanted to work. The electronic interfaces for *Eternal Owl Call* for Kingma system bass flute & electronics (vol. 1, pp. 291–300) were all coded using SuperCollider, the source code for which is supplied in **Appendix 7** (vol. 2, pp. 220–308), following explanatory class library extension documentation (vol. 2, pp. 163–219). A user guide for the software, written in layman’s terms, is also supplied in **Appendix 6** (vol. 2, pp. 135–58) and there is an example run script at the end of the user guide, which contains some commented-out information (in blue syntax on pp. 155–8) which provides background to the piece as well as technical specifications. The concept for *Eternal Owl Call*, along with its Celtic mythology-based subjectivity is also described in layman’s terms in the programme note on p. 293 of vol. 1. The flute writing in the piece incorporates a variety of unorthodox extended techniques and I worked very closely with avant-grade flautist Carla Rees in order to verify the practicality and playability of the techniques called for. Much of what is written in the flute line is specific to Carla Rees’s specialist technique, in particular the use of quarter tones, very precise microtonal pitch bends, timbral trills, natural harmonics, multiphonics, articulated air sounds (using phonetic syllables) and modifications to the instrument’s natural timbre.

---

<sup>3</sup> Ibid.

*Eternal Owl Call* follows an  $A_1BCA_2$  structure (mm. 1–16, 17–46, 47–64, 65–81 respectively). The flute writing in  $A_2$  is a retrograde of structural material in  $A_1$  with timbral modifications added. The flute line starts with a microtonal root of  $F\sharp_4$  without any clear suggestion of the (poly)modality seen in much of my previous compositional work. However, this does change as the piece progresses. In mm. 14–15 there is a first emergence of typical intervallic cells as shown below:



Fig. 4.1

The first software patch (labelled as sub-patch **1.01–1.05** in the score, mm. 1–16) sets an organic tone by fading in samples of neotropical ambience along with other nature sounds and a series of owl calls – which are particularly symbolic to this piece as it is a depiction of the fate of the Celtic mythological figure Blodeuwedd, a beautiful maiden who is conjured from flowers and oak to marry a prince but flees and is eventually transformed into an owl for all eternity as punishment for her sin. Random-value frequency modulation and pan (i.e. spatial, left–right) modulation is applied to nature sounds in sub-patches **1.02–1.03** at given time intervals, which relates to the type of Xenakisian<sup>4</sup> procedures previously described – the values generated are not completely random, but are instead set to certain mathematical limits, as a way of controlling the type of sound created whilst allowing a certain degree of indeterminacy. Sometimes the nature samples mutate/modulate in a subtle way, whilst at other times they become very electronic and sound quite alien – almost like something out of a science-fiction movie!

The nature sounds begin a long fade-out in m.15, which is then blended with the sound of both natural harmonics in the flute and electronically generated harmonics in the B section, relating to the second patch (i.e. sub-patches **2.01–2.11**). The microtonal pitch bends in the flute line in mm. 21–32 are closely related to partials from the harmonic series, which are related to the frequencies of the harmonics which also occur in the electronics. There is a much clearer glimpse of (poly)modality in the flute line in mm. 36–40, with intervallic cells which relate closely to the modes in operation, as shown overleaf:

<sup>4</sup> Ibid.

B. Fl. *mf* 36 *9:8* *A Lydian* *C Lydian* *3* *inverted 4+1b-cell* *Stacked B/E Lydian* *2+2a-cell* *5* *7/8*

B. Fl. *mp* 38 *4+3a-cell* *7/8* *39* *4/4* *3+3a-cell* *5* *3* *40*

Fig. 4.2

In addition, further cells occur at the end of m.46:

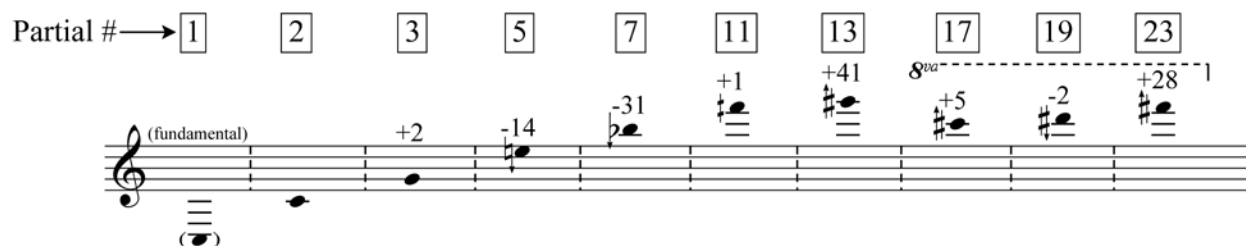
B. Fl. *mp* *mf* 46 *3+1b-cell* *ord.* *6* *2+1c-cell* *flz.* *35*

Fig. 4.3

The electronically generated frequencies use sine waves at a given frequency controlled by a square wave amplitude filter – the resulting sounds are reminiscent of little bells or chimes, which symbolise perpetual judgment, as is relevant to the narrative behind this piece. In addition, the looped 9-note row in m. 38 uses an electronically generated bell algorithm, created using SuperCollider’s “Klank” bell-modelling interface.<sup>5</sup> The frequencies in operation in sub-patches **2.01–2.07** and **2.09** are all based on natural harmonics of six separate musical pitch frequencies: namely D $\flat$ 2–E $\flat$ 2–F2–G2–C3–D $\sharp$ 3, which makes up a 4+2a-cell hexachord on D $\flat$ . The natural harmonics employed based on this intervallic collection are all prime-numbered partials of the relevant harmonic series (a prime number is an integer that can only be divided by itself or one, i.e. 2<sup>nd</sup>, 3<sup>rd</sup>, 5<sup>th</sup>, 7<sup>th</sup>, 11<sup>th</sup> partials of the harmonic series etc...). My reason for choosing prime-numbered partials was to avoid any octave doublings within that harmonic series in order to avoid any obvious and clichéd natural harmonic sequences (i.e. I wish to avoid the 4<sup>th</sup> and 6<sup>th</sup> partials as they are octave doublings of the 2<sup>nd</sup> and 3<sup>rd</sup> respectively, etc...). As a consequence of this, I am effectively creating a type of ‘microtonal serialism’ principle, where I avoid repeating any microtones at the octave, with the software-based sound synthesis techniques involved making the level of mathematical accuracy required to achieve this a possibility. To illustrate this, all prime numbered partials up to the 23<sup>rd</sup> partial of C3 (130.81 Hz) are shown overleaf, with the microtonal offset from the nearest musical pitch given in cents:

<sup>5</sup> In SuperCollider, the Klank bell-modelling class was preceded by the Klang class (which is the German word for “sound” and also shares its name with a well-known acousmatic work by electroacoustic composer Jonty Harrison [composed in 1982], which processes and transforms a recorded acoustic metallic/bell-like sound).

### Prime Numbered Partial of C3 (130.81 Hz)



**Fig. 4.4**

In addition, the sum of microtonal frequencies that ring simultaneously at the climax of the B section (in mm. 41–43) is shown overleaf (**Fig. 4.5**). This diagram illustrates the origin of the precise microtonal pitches indicated in the electronics part in the B section.

Breathy multiphonics are adopted predominantly in the flute line in the C section, starting in m. 47, marked “Atmospheric, Warm, Breathy”. In sub-patch **3.01** (m. 48) a long, atmospheric reverberation and 12-line delay algorithm is added to the microphone in order to sustain the flute multiphonics and build to a sound-mass. This technique is similar to that adopted in much of Kajia Saariaho’s ‘live’ electronic music, where a reverb algorithm of very long, or even infinite decay time is used to ‘freeze’ a sound without the necessity to use potentially problematic FFT<sup>6</sup> and spectral re-synthesis algorithms (which can lead to problems with computer memory consumption and speaker distortion in some circumstances). Here, the reverb/delay on the breathy flute sounds symbolises a ‘trapped’ human being, as is related to the work’s narrative. The bass pulse and the low-pitched modulated sine wave pulses which are initialised in sub-patch **3.02** (m. 54) are also symbolic – representing the heartbeat of a human.

Following a crossfade between the reverberated modular synths in the C section and the recurrence of nature sounds and neotropical ambience in m. 65, the ‘retrograded’ recapitulation, starting in m. 66 sees further examples of typical intervallic cells in the flute line, as shown overleaf (in **Fig. 4.6**):

<sup>6</sup> FFT (Fast Fourier Transform) is a quantised mathematical system for working with three dimensional coordinates. In audio engineering FFT is used for the purposes of the spectral analysis and re-synthesis of live or pre-recorded audio signals (with the three respective dimensions being time, amplitude and frequency). A three dimensional coordinate system such as FFT is necessary where frequency (i.e. pitch) variations need to be analysed or re-synthesised without changing the sample rate (i.e. the playback speed) of the signal.

**Fig. 4.5:** Sum of Electronically Generated Harmonics in mm. 41–43

Partial # of...

23

19

17

13

11

7

5

3

2

-31

-31

-31

-31

-31

-31

-14

-14

-14

-14

-14

-14

+2

+2

+2

+2

+2

+2

+28

+28

-2

-2

+5

+5

+41

+41

+1

+1

-31

-31

-14

-14

+2

+2

4+2a-cell

1+2a-cell

1+2a-cell

Looped 9-note row extracted from a stacked A $\flat$ /D $\flat$ /G $\flat$  polymode



### Tempo Primo (Very Subdued, Melancholy, Sustained)

The flute timbre in the final section is always either hollow sound (h.s.) or flutter tongued (flz.)

$\text{♩} = \text{c.}88$

Fig. 4.6

The long atmospheric reverb algorithm remains present on the microphone signal throughout the duration of the A<sub>2</sub> section, therefore as a result the flute line sounds rather different to that heard in A<sub>1</sub> with a haunting feel to the section. This is further enhanced in m. 72, where a series of owl call samples are also sent to the reverb channel, with slight pitch bends added for further effect.

Some of the modal and intervallic techniques in the solo flute writing in *Eternal Owl Call* can be closely related to classical 20<sup>th</sup> century works for solo flute, such as Debussy's *Syrinx* (1913, sharing its mythology-related subjectivity) and Edgar Varèse's *Density 21.5* (1936), however the organic feel of the writing, along with the adoption of several effective extended techniques (including multiphonics) moves it closer stylistically to some of Toru Takemitsu's writing for solo flute in works such as *Voice* (composed in collaboration with avant-garde flautist Pierre-Yves Artaud, 1971), *Itinerant—In Memory of Isamu Noguchi* (1989) and *Air* (1995). The application of electronics and live electronic interfaces to a piece for solo instrument has origins in some of Stockhausen's pioneering pieces of 'live' electronic music, including *Mikrophonie I* for solo tam-tam and electronics (Nr. 15, 1964) and *Solo* for any solo melodic instrument and electronics (Nr. 19, 1965–6).

During the composition of *Eternal Owl Call* I also liaised closely with electroacoustic composer Michael Oliva, who is a resident composer with the Rarescale contemporary music ensemble. Some of Oliva's well-known recent works for solo flute and electronic medium include *Apparition and Release*<sup>7</sup> for Kingma system alto flute and electronics (2005) and *Bereft Adrift*<sup>8</sup> for Kingma system bass flute and electronics (2007) which are similar both conceptually and stylistically to much of my writing, in particular the bass flute writing in *Bereft Adrift* which uses a number of multiphonics with exactly the same finger positions as those adopted in my own flute line. Oliva's approach to the live electronic component does, however, tend to be different and he tends to use a pre-recorded electronic backing track rather than a live 'on-the-fly' interface as I have created using SuperCollider.

<sup>7</sup> Discussed further in Carla Rees's recent PhD thesis, 'Developing a Repertoire of Extended Techniques for the Kingma System Alto and Bass Flute' (unpublished doctoral thesis, Royal College of Music, 2014), 51.

<sup>8</sup> Ibid., 52.

One composer who is also a SuperCollider programmer and has composed live electroacoustic works for Carla Rees is Scott Wilson (currently an academic at the University of Birmingham). Some of Wilson's recent works for flute and electronics include *Fluxion* for Kingma system alto flute, piano and electronics (2006) and *Vortically*<sup>9</sup> for Kingma system bass flute and electronics (2009). It should be noted that Wilson is also a leading authority on the SuperCollider language and is one of the editors of the de facto textbook for learning and studying this language, along with David Cottle and Nick Collins [*The SuperCollider Book* (Massachusetts: MIT Press, 2011)]. Wilson is also an active contributor to the development of the SuperCollider language.

The sound recordings of wildlife which were used in *Eternal Owl Call* have been obtained from the Macaulay Library at the Cornell Laboratory of Ornithology (CLO), Ithaca, New York, USA (via the recommendation of Dr Arlene Sierra, who also recently utilised birdsong recordings from this exhaustive audio/video file archive in her composition *Urban Birds* [2014] for three pianos and live electronics). A copyright agreement has been reached between myself and CLO for the use of these audio files in the corresponding piece.

---

<sup>9</sup> Ibid., 86–7.

# CHAPTER 5

*“Urban Wilderness”*

for String Quartet

## “Urban Wilderness” for String Quartet

As mentioned in the programme note on p. 37 of vol. 1, *Urban Wilderness* is part of a cycle of works for string quartet which I am currently in the process of completing. Its concept is inspired by Fritz Lang’s hugely influential 1927 cult science-fiction movie *Metropolis* (originally based on a novel under the same name by Thea von Harbou). At present the cycle contains two individual works:

#1: *Apathetic Machines* (representing apathy and subconscious machine control as portrayed in *Metropolis*’ “Shift Change” scene);

#2: *Utopian Mirror* (representing a machine-generated utopian ‘virtual reality’ or ‘false reality’).

In a strict sense, *Urban Wilderness* is not a film score. Instead, it is part of a cycle of individual works which are inspired, in one way or another, by films/videos which conceptualise the idea of futuristic urban technological dystopia, in particular *Metropolis*, Ridley Scott’s *Blade Runner* (1982), the Wachowskis’ *The Matrix* (1999) [as well as the sequels in its trilogy: *The Matrix Reloaded* (2003) and *The Matrix Revolutions* (2003)] and the music videos for Kraftwerk’s electronic music album *Man Machine* (1978). From a philosophical perspective, my intention with the idea for *Urban Wilderness* is also to draw attention to certain symmetries which exist between real-life modern urban existence and the fictional futuristic/dystopian vision which was present in Fritz Lang’s movie almost a century ago.

At the present time *Urban Wilderness* exists as a cycle of works for string quartet which would be performed without any film or video projection in the background; however, video projection is something which may be added to the cycle at some point in the future. In this respect, my intention is not that any particular film or video be shown chronologically from start to finish, instead, my intention is that scenes from multiple films or videos (which are particularly significant to the concept which an individual piece within the cycle conveys) be projected one after the other in the background whilst a particular piece is being performed (for example certain key scenes from the futuristic dystopian films *Metropolis*, *Blade Runner* and *The Matrix Trilogy*, as well as sections from some of the futuristic music videos for Kraftwerk’s *Man Machine* album would be projected consecutively in the background while the string quartet is performing; in addition, when the musical character changes the sequence of film/video scenes could be changed using some sort of an electronic controller).

The plot for *The Matrix* is quite clearly modelled on that of *Metropolis*: both feature futuristic urban dystopias; both feature the concept of humans being controlled by machines; both feature underground human communities that revolt and wage war against the machines; both feature a “prophecy” about the arrival of a “mediator” (*Metropolis*) or “chosen one” (*The Matrix*); both feature the concept of humans being replicated by machines whilst in a semi-conscious state.

In addition, I have a strong interest in other contemporary art-forms which have been influenced by *Metropolis*, in particular the link which exists with the albums of German electronic music band *Kraftwerk*.<sup>1</sup> *Kraftwerk* were heavily influenced by *Metropolis* and their 1978 album *Die Mensch-Maschine* (*The Man-Machine*) shares its name with the film's semi-human robot ("Maschinen-Mensch", as named in the starting credits). *Kraftwerk*'s album features six individual tracks (*The Robots*, *Spacelab*, *Metropolis*, *The Model*, *Neon Lights* and *The Man-Machine*). Their mechanical/repetitive style of electronic music, was intended to symbolise technological and futuristic concepts.

My reason for choosing to set music based on this concept to a work for string quartet was because I wanted to emulate a classical film score, such as is set to many of the American black-and-white films in the 1940s, 50s and 60s. *Metropolis* is, of course, German rather than American and predates such films. It is a silent movie with intertitles and various soundtracks have been added to it by various composers and bands over the years, however, its original score was composed by Gottfried Huppertz in an orchestral style inspired by Richard Wagner and Richard Strauss. Whilst Huppertz's score can be regarded to suggest and portray emotions in the film to good effect, I wanted to create a work which was more closely aligned with the styles of post-tonal modernist composers in the 20<sup>th</sup> and 21<sup>st</sup> century – moving away from tonality and towards the use of modes, chromaticism and serialism. The type of musical ambience which I wanted to convey in my own version was something which would initially be similar to some of the string writing in scores from later expressionist American films with some sort of an urban theme associated with them, such as Billy Wilder's *Double Indemnity* (1944, with the score composed by Bartók-influenced Hungarian composer Miklós Rózsa) and Alfred Hitchcock's *Psycho* (1964, with the score composed by Bernard Herrmann).

Billy Wilder advised Rózsa that he wanted a "restless string fugue" for *Double Indemnity*,<sup>2</sup> and consequently the score is composed in a style which is contrapuntal and chromatic, incorporates elements of jazz harmony, Bartókian (poly)modality and has a feel of tonal or modal ambiguity to it, akin to many works by post-tonal classical composers in the first half of the twentieth century. Bernard Herrmann's score for *Psycho* is composed for a string ensemble, with the music on the title credits (prior to the initial urban apartment scene) creating psychological suspense and intrigue with rhythmic, repetitive ostinato-based polyphony set in a contemporary modal style with several dissonant intervals. This thematic material reaches its head in the infamous/iconic 'shower scene', where screeching high-pitched violin dischords on a repeated crotchet figure accompany the premature murder of the film's heroine Marion Crane (played by Janet Leigh).

---

<sup>1</sup> This is a topic of research which I discussed in my final-year undergraduate dissertation, 'Germany, Psychedelia, Technology: A Consideration of the Interface Between Electronic and Commercial Music (1953–1986)', (unpublished undergraduate dissertation, King's College London, 2000).

<sup>2</sup> Miklós Rózsa, *Double Life: The Autobiography of Miklós Rózsa* (New York: Hippocrene Books, 1998), 119.

*Apathetic Machines* (the first movement of *Urban Wilderness*, vol. 1, pp. 52–71) is inspired by the “Shift Change” scene at the start of *Metropolis*, where large groups of frown-faced, apathetic and oppressed-looking, machine-controlled men walk into work all in a straight line, wearing identical uniforms and having identical facial expressions. *Apathetic Machines* follows a conventional sonata form with two separate motifs: firstly the “apathy motif” and secondly the “machine motif”. The “apathy motive” is slow, atmospheric and subdued, whilst the “machine motif” is lively, rhythmic and repetitive, with the repetition representing the subconscious, psychotic and trance-like state of the machine-controlled humans – this concept relates to the expressionist movement, which influenced the making of *Metropolis*.

In fact the structure was inspired by the evolved type of sonata form which is seen in Beethoven’s *Sonata Pathétique: Op. 13 in C minor* for solo piano, which starts with a slow, sombre section marked “Grave” before moving into an agitated rhythmic section marked “Allegro di molto e con brio”, which starts with contrapuntal right hand material above tremolos on pedal points in the left hand. Likewise, *Apathetic Machines* starts in a slow subdued fashion before changing to a lively section with repeated rhythmic figures.

On reflection, I would class this piece in the ‘explicit polymodality’ category and place it alongside other explicit polymodal works in the portfolio, such as the two John Morris-Jones settings (*Iâr fach yr haf & Y Gwyllnod*), the first movement of *Cwyn y Gwynt* (both the flute and contralto versions) and *Twilight Impulse*. It was completed in 2010, at a similar period in my development as a composer to the above mentioned pieces. I would, however, argue that it does begin to move away from an explicit/transparent form of polymodality and towards a style which begins to demonstrate a greater level of chromatic complexity.

*Apathetic Machines* starts with long sustained polymodal harmonies using double stops in the second violin, viola and cello whilst the first violin plays a soloistic figure above these harmonies. This idea is reminiscent of that seen in the third movement of Bartók’s fourth string quartet (the work’s ‘nucleus’, marked “Non troppo lento”), which starts with long sustained double stops in the violins and viola (forming an A–B–C#–E–F#–G# modal hexachord), whilst a cello melody plays a chromatic/polymodal passage below, starting with an alternation between D $\flat$  and D# (which implies a constant modal fluctuation between A Ionian and A Lydian). In this instance, Bartók’s polymodality is ‘crossed’ (i.e. the chromatically related modes ‘cross’ one another in the same pitch regions, as opposed to the ‘stacked’ form of polymodality which is typically seen in my own polymodal style), as is often the case in Bartók’s polymodal style. There are some instances of crossed polymodality in *Apathetic Machines*, however the overwhelming bias is towards stacked polymodality.

A table showing the structural breakdown of *Apathetic Machines* is provided overleaf:

### **Apathetic Machines Structural Breakdown:**

<b>Measures</b>	<b>Section/Subsection</b>	<b>Description</b>	<b>Concept</b>
mm. 1–52	<u>Exposition</u>		
mm. 1–12	first theme	subdued, marked: ♩ = c.66 “Atmospheric, Sustained”	“apathy motif”
mm. 13–18	transition section	changes the tempo to ♩ = c.144 “Intense, Articulate”	
mm. 19–32	second theme	repetitive machine-like rhythmic figures and contrapuntal lines	“machine motif”
mm. 33–52	codetta section	builds towards a climax with double stops & glissandi at <i>f</i> – <i>fff</i> dynamic, marked: ♩ = c. 132 “Broader, Aggressive”	
mm. 53–117	<u>Development Section</u>		
mm. 53–72	first episode	process-based rhythmic augmentation of the second theme	“malfunctioning machines” episode
mm. 73–87	second episode	pizz., Bartók pizz., nat., sul pont. rhythmic figures in 9/8, no melodic material	“clockwork” episode
mm. 88–117	third episode	process-based rhythmic diminution of the second theme	“overloading machines” episode
mm. 118–72	<u>Recapitulation</u>	intervallic inversion/ “mirror-image” of the exposition	
mm. 118–29	first theme	mirror image of mm. 1–12	mirror “apathy motif”
mm. 130–5	transition section	mirror image of mm. 13–18	
mm. 136–49	second theme	mirror image of mm. 19–32 2-note bowed figures replace rhythmic pedal-points	mirror “machine motif”
mm. 150–72	extended coda section	mirror image of mm. 33–52 expanded natural harmonics section and aggressive final cadence added	

A harmonic/polymodal progression model for the first theme is shown overleaf (**Fig. 5.1.1**):

**Fig. 5.1.1:** harmonic/polymodal model for mm. 1–12 of *Apathetic Machines*

**Atmospheric, Sustained**  
♩ = c.66

*molto espress.*

**F# major pentatonic**

**Bb Ionian**

**D major pentatonic**

**C# minor pentatonic**

**stacked E/D/G Lydian**

**F Mixolydian**

**G minor pentatonic**

**stacked Eb/Db Lydian**

*poco rall. -----*

*port.*

**A Tempo**

release the harmonic

**stacked E/D Lydian**

**E/D Lydian**

**D/Bb Dorian**

**D major pentatonic/ altered scale on D#**

**F/Bb/Ab Lydian**

**A/D/C Lydian**

**G/C/Bb Lydian**

**E/D Lydian**

**F#/E Dorian**

**E/A/G Locrian**

**F/Bb/Ab Lydian**

**B/E/D Lydian**

**Ab/C#/B Lydian**

**stacked C/F/Eb Lydian**



The “machine motif” first occurs at the start of the second theme in mm. 19–20 as shown below:

**Fig. 5.1.2**

Here, there is an example of a polymodal crossing between the  $G\sharp$  pedal-point in the viola line and the first melodic  $G\sharp$  in the first violin line in m. 20, however, once the first violin line reaches its upper register the music returns to a more common stacked polymodal orientation. The violin line in m. 20 consists of pitch-classes from the hexachord  $E-F\sharp-G\sharp-B-C\sharp-D\sharp$ , which, interestingly, is a transposition of the hexachord seen at the start of the nucleus section in Bartók’s fourth quartet. From the perspective of rhythmic tension and articulation, the second theme (i.e. the “machine motif”) includes accentuation on off-beats, a juxtaposition of 8/8 time signatures with 9/8 and repeated rhythmic repetition which is at times reminiscent of some of Stravinsky’s strata-based writing in *The Rite of Spring* (1913).

A harmonic/polymodal model for the climax of the first codetta section in mm. 42<sup>3</sup>–51 (♩ beat) is shown overleaf (**Fig. 5.1.3**).

Both the “malfunctioning machines” and the “overloading machines” episodes are intended to symbolise the “Moloch!” scene in *Metropolis* (named after the Ammonite god of sacrifice), where Freder (one the film’s central characters played by Gustav Fröhlich) is horrified to discover a shift worker collapsing from exhaustion whilst handling machinery, leading to an explosion, which kills other workers. The “malfunctioning machines” episode symbolises the exhausted, collapsing workers, whilst the “overloaded machines” episode symbolises the exploding machine.

The “malfunctioning machines” episode includes a process-based rhythmic augmentation of material from the second theme in mm. 58–72, where the repeated pedal-points mutate from [regular semiquavers]–[triplet quavers]–[regular quavers]–[crotchet triplets]–[regular crotchets]. By the time they become crotchets in m. 71 the “machine motif” has effectively mutated into the “apathy motif”, with corresponding melodic material in the first violin line.

The “overloading machines” episode includes a process based rhythmic diminution of material from the second theme in mm. 94–116. This time around the rhythmically repeating pedal-points mutate from regular semiquavers–quintuplets–sextuplets–septuplets–32<sup>nd</sup>-notes, using slurred figures across

## Broader, Aggressive

♩ = c.132

42 (b. 3)

Vln. I

Vln. II

Vla.

Vc.

**f**

**f**

**f**

**f**

stacked A $\flat$ /G $\sharp$  Lydian

C $\sharp$  Locrian

G Ionian

43

**sfz** gliss.

**sfz** gliss.

**sfz** gliss.

**sfz** gliss.

E Dorian

44

Vln. I

Vln. II

Vla.

Vc.

**f**

**f**

**f**

**f**

A Ionian

45

**sfz** gliss.

gliss.

**sfz** gliss.

**sfz** gliss.

G Lydian

sul pont.

**ff**

sul pont.

**ff**

sul pont.

**ff**

sul pont.

**ff**

D Lydian

46

Vln. I

Vln. II

Vla.

Vc.

**fff** gliss.

**fff** gliss.

**fff** gliss.

**fff** gliss.

47–51

nat.

**fff**

nat.

**fff**

nat.

**fff**

nat.

**fff**

stacked E $\flat$ /A $\flat$  Lydian

modal tri/tetrachords before a climax and then a diminuendo using harmonic 2-note tremolos in mm. 108–116.

The “clockwork” episode is intended to symbolise the machine-controlled humans, as well as the machinery itself, ‘running like clockwork’ in a repetitive, mechanical fashion. Hence the lack of any melodic or musically expressive material in this section.

The recapitulation is a “mirror image” of the exposition, for example, the harmonies in the lowest three parts in m. 1 (with the lowest-pitch voices sounding first), are moved to the upper three parts in m. 118 (with the highest-pitched voices sounding first). The soloistic melody is also moved from the first violin (i.e. the highest pitched voice) to the cello (i.e. the lowest pitch voice) and the musical intervals are inverted (resulting in ascending runs from the exposition being transformed into descending runs in the recapitulation and vice versa). It should, however, be noted that the music is not always an exact intervallic inversion of material in the exposition because I decided that it was often more effective (from either a pragmatic musical or scoring perspective) to occasionally deviate from using exact intervallic inversions throughout. A harmonic/polymodal progression model for the recurrence of the first theme in the recapitulation section is shown overleaf (**Fig. 5.1.4**).

*Utopian Mirror* (the second movement of *Urban Wilderness*) is a representation of a utopian ‘virtual reality’ or ‘false reality’ from within the type of dystopian/futuristic world portrayed in films such as *Metropolis* and *The Matrix* as well as in real-life modern urban existence. This is a concept which is very well conveyed in *The Matrix*, where machine-controlled humans in a distant future world are unwittingly put into a semi-conscious state by machines and then sent into a software-based virtual reality, which mimics New York in the year 1999. Due to the harsh reality of the futuristic dystopia in which the exiled underground human communities live, New York in the year 1999 represents a utopia, from a comparative perspective. As a consequence of this, some humans who have escaped the false reality are tempted by the Matrix to re-join it as machine-controlled humans (with added benefits), in exchange for some kind of betrayal of the human community.

I also view this as something of a metaphor for the individuals in real-life modern cities who engage in some form of a harmful pleasurable activity, either to provide an escape from a harsh way of life (such as through drug, alcohol or gambling addiction), or through their own personal greed (such as engaging in some form of corruption for the purposes of some form of personal, financial, corporate or political gain).

*Utopian Mirror* (vol. 1, pp. 72–99) is an experimental étude in the use of harmonics on bowed string instruments, but also juxtaposes the timbral extended techniques employed in this vein with a form of mature chromatic polymodality (having been completed in 2014, it is one of my most recent works), which is predominantly stacked, but also quite often bringing elements of crossed polymodality into play

**Fig. 5.1.4:** harmonic/polymodal model for mm. 118–29 of *Apathetic Machines*

**Atmospheric, Sustained**  
♩ = c.66

**stacked F#/B Dorian upper-structure**

Vln. I  
Vln. II/Vla.  
Vc.

*pp*  
*pp*  
*mp*

*(flautando e molto espress. sempre)*

*3*  
*5*  
*3*

**D Ionian**  
**F# major pentatonic**  
**B♭ Lydian**

**stacked A/D/B♭ Lydian**

**stacked C/F Dorian upper-structure**

Vln. I  
Vln. II/Vla.  
Vc.

*gliss.*  
*gliss.*  
*pp*  
*mf*

*poco rall.*

**E♭ Dorian**  
**D♭ Lydian**

**stacked C/F/E♭ Dorian**

**stacked B/E/A Dorian**

Vln. I  
Vln. II/Vla.  
Vc.

*pp*  
*pp*  
*p*

*release the harmonic*  
*A Tempo*

*cresc.*  
*cresc.*  
*cresc.*

**G#/B Locrian**  
**E♭/B Lydian**  
**A Lydian (♭7)/ B major pentatonic**  
**E♭/B Lydian**  
**E/A/G Lydian**  
**G♭/B/A Lydian**

**stacked D♭/B/E Lydian**

Vln. I  
Vln. II/Vla.  
Vc.

**A/G Ionian**  
**B/A Ionian**  
**E♭/G♭ Lydian**  
**E♭/C#/F# Ionian**  
**D/C/F Lydian**  
**F/E♭/A♭ Lydian**

**stacked D♭/B/E Lydian**

*f*  
*f*  
*ff*

**fifth partial - should sound two octaves and a major third above the fundamental**

**sixth partial - should sound two octaves and a perfect fifth above the fundamental**

in quite an explicit fashion. Normal stopped pitches are often called for on the lowest string of each respective instrument, exploring some of the deepest musical sonorities available for a string quartet configuration. The sections with normal stopped pitch material represent the underlying dystopian reality which exists alongside the virtual/false reality, which is represented by colouristic, calm and ethereal string harmonics.

*Utopian Mirror* has a very long structure; the recordings of this piece (across two separate workshops with Carducci quartet totalled a playing time of the movement at approximately 22 minutes). It is largely through-composed, however, some aspects of it do appear to resemble something which resides in the territory between a long extended ternary form and an extended sonata form. One thing which is certain is that there is a recapitulation of the movement's  $A_1$  section (mm. 1–31) in the  $A_2$  section which occurs in mm. 169–201 (albeit at a slower tempo, with more notes and halved rhythmic time values from that seen in  $A_1$ ). The A section mostly consists of low-pitched stopped pitches, whilst the section which follows it in mm. 32–46 (which I would class as a B section) uses harmonics throughout. I would then class mm. 47–110 as an extended development section (C section) which contains both contrapuntal material on normal stopped pitches and material using harmonics. The section from mm. 111–168 is the movement's climax (D section), marked "Radiant & Very Colouristic", which features repeated glissandi up and down the natural harmonic nodes on each instrument and is symbolically a representation of false utopian bliss. The final section in mm. 202–230 is a coda which features further glissandi along the harmonic nodes, a viola solo and harmonics tremolos as well as double-stopped harmonics in the closing measures. One further point to make is that there are instances in both the C and D sections where fragments of material in the B section recur, consequently, the C and D sections can potentially be divided into smaller subsections for the purposes of analysis.

For further insight into some of the extended techniques employed in *Utopian Mirror* there is also an informative notation guide at the start of the score for *Urban Wilderness* (vol. 1, pp. 38–51). There is also a copy of the extensive hand-written sketch which I put together for *Utopian Mirror* in **Appendix 4** (vol. 2, pp. 109–21).

I previously discussed how much of the string writing in *Apathetic Machines* was influenced by Bartók's string quartets, as well as the film scores of Miklós Rózsa and Bernard Herrmann. *Utopian Mirror* is, however stylistically different to *Apathetic Machines* and has more specific influences, which continue to include Bartók, as well as string writing by Hungarian contemporaries of Bartók, including the string quartets of György Ligeti and György Kurtág. George Crumb's use of string glissando techniques along the harmonic nodes in works such as *Vox Balaenae* (1971) is also an influence which is discussed in the notation guide, as well as some of the string glissando techniques seen in works by Iannis Xenakis, such as *Nomos Alpha* for solo cello (1965). German composers who have specialised in the use

of extended string techniques using harmonics include Helmut Lachenmann and Wolfgang Rihm as well as the Italian composers Salvatore Sciarrino and Franco Donatoni. Sciarrino's *Six Caprices* (1976) for solo violin includes many extended techniques using harmonics and the sound of some the effects in *Utopian Mirror* reminded me of some of the string effects in his chamber opera *Luci mie traditrici* (*Oh My Betraying Eyes*, also known as *The Killing Flower*, 1996–8). Another contemporary solo string piece which inspired me was Kajia Saariaho's *Sept Papillons* (*Seven Butterflies*) for solo cello (2000), which shares some of its natural/organic and bird/insect-related subjectivity with other works in my portfolio including *Iâr fach yr haf* (“*The Butterfly*”), *Y Gwylanod* (“*The Seagulls*”) and *Eternal Owl Call*.

# CHAPTER 6

## Style, Influence & Context

## Style, Influence & Context

*Ynys Afallon* for SSAATTBB is the one piece within the portfolio which can properly be classed as being composed in modal style (i.e. non-polymodal) throughout, albeit with subtle hints at polymodality, as discussed in the previous chapter on modality. Some of the choral writing in this piece does bear a similarity to the modern, modal, a cappella choral writing of popular/crossover composers such as James MacMillan, Eric Whitacre and Paul Maelor; Some aspects of the harmonisation in *Ynys Afallon* bear a similarity to that seen in works such as MacMillan's *Tenebrae Responsories* and *The Gallant Weaver* (both for SSAATTBB a cappella; 2006 and 1997 respectively), whilst some aspects of its madrigal-like polyphony bears a similarity to some of the writing seen in MacMillan's *Màiri* for mixed 16-part choir a cappella (1995). The influence of mainstream contemporary choral music on *Ynys Afallon* can be extended to the work of European minimalist composers, including Arvo Pärt and Henryck Górecki, with some of its atmospheric harmonic layers bearing a resemblance to that seen in some of the string and soprano-voice layering in Górecki's *Symphony No. 3: Symphony of Sorrowful Songs* (1976).

However, the individual harmonic and modal layers in *Ynys Afallon* are, on the whole, richer and denser than that which is present in the above mentioned works and there are more frequent chromatic modal changes which bring some aspects the style closer to that seen in late-period choral works of French impressionists such as Debussy and Ravel as well as incorporating a harmonic language which is closely related to modern modal jazz harmony. In addition, Celtic mythology (rather than religion) forms the subject-matter of the text-setting for this piece.

In truth, the modal compositional style of *Ynys Afallon* for SSAATTBB is not true to my progressive aims as a composer and has, instead, been composed for an incidental purpose and for its suitability to be performed by amateur or semi-professional level singers (as discussed in the previous chapter on modality). From an ideological artistic standpoint, I would have preferred to set such a choral work in a style which is polymodal and with greater chromatic complexity than that which is present in this particular arrangement (aiming towards a choral work which would be suitable for performance by a contemporary professional-level choir) and this could form the basis for a future re-composition of a choral setting to the same text by T. Gwynn Jones (and/or my English-language translation of this text, perhaps set for a mixed 16-part choir).

The settings of works for solo voice to the Welsh-language poetry of Sir John Morris-Jones (namely *Iâr fach yr haf* & *Y Gwylanod* for soprano, flute/piccolo and piano and the version of *Cwyn y Gwynn* for contralto, vibraphone and cello) possess the sort of (poly)modal/impressionistic organicism and nature-related musical symbolism present the works of composers along the axis of Debussy–Messiaen–Takemitsu, right through to living contemporaries such as Kajia Saariaho and Oliver Knussen,



with the harmonic style of the piano writing also bearing a similarity to the syntax which is present in the modal jazz harmonisations of contemporary jazz/improvisatory pianist-composers such as Bill Evans, Keith Jarrett, Chick Corea and Joe Zawinul.

In addition, the Morris-Jones settings look to transcend the sort of conventional tonal or modal song-writing approach which has typically been seen in musical settings of Welsh-language poetry over the last few decades, moving towards a style which possesses touches of abstract modern modal/chromatic impressionism, such as that which is heard towards the end of Pwyll ap Sion's *Merch* (*Lady*, 1998) for baritone and piano and explores contemporary chromatic or serial pitch territory, as is seen in the songs settings of Guto Pryderi Puw, including *Blodeuwedd* and *Dawns y Sêr* (2000 and 2001 respectively) both set for baritone and piano to poetry by Nesta Wyn Jones.

The atmospheric/modal harp writing in the re-composition of *Cwyn y Gwynt* for flute and harp also bears a resemblance to some of the writing in Christopher Painter's *Syniadau'r Serch* (*Thoughts on Love*), set to words from the old verses for baritone, violin and harp (a work which incidentally also won the composer's medal at the Welsh National Eisteddfod in Ebbw Vale in 2010). The flute and harp re-composition of *Cwyn y Gwynt* demonstrates a clear influence of Debussy, Ravel, Messiaen and moves towards the type of post-impressionism evident in some of Toru Takemitsu's chamber works for flute (or other woodwind instruments, including the Japanese shakuhachi) and one accompanying plucked stringed instrument (such as harp, guitar or the Japanese koto or biwa). One Takemitsu work with a similar instrumental configuration which influenced my re-composition of *Cwyn y Gwynt* (*The Wind's Lament*) for flute and harp was Takemitsu's *And then I knew 'twas Wind*, for flute, viola and harp (1992), which shares conceptual and programmatic similarities with my piece, as well as sharing clear similarities in relation to instrumentation and compositional style.

The early works of Takemitsu were also influenced by the work of composers of the Second Viennese School<sup>1</sup> and this type of chromatic or serial expressionism becomes more evident in *Breuddwyd* (the final movement of both versions of *Cwyn y Gwynt*), in particular in the version for contralto, vibraphone and cello, where the pitch material is entirely constructed around semi-modal twelve-tone serial techniques and the type of 'heptachordal rotation' illustrated in **Figures 1.3.1–11** in the first chapter; which brings this movement closer, from a stylistic standpoint, to the post-tonal or modal twelve-tone serial works of Alban Berg, late Stravinsky, Messiaen, Ernest Kerenk and Oliver Knussen.

The piano writing in both *Twilight Impulse* (for clarinet, cello and pianoforte) and the original version of *Ynys Afallon* (for SATB choir and pianoforte) is highly idiomatic and moves towards a greater level of maturity than that which has been seen in the piano accompaniments for *Iâr fach yr haf* and *Y*

---

<sup>1</sup> Timothy Koozin, 'Octatonicism in Recent Solo Piano Works of Toru Takemitsu', *Perspectives of New Music*, Vol. 29, No. 1 (Winter 1991), 124.

*Gwyllanod*. The highly chromatic, post-impressionistic and polymodal style of such piano writing owes itself, in part, to the type of writing seen in the work of French composers in the middle period of the twentieth century, including Messiaen's *Vingt regard sur l'enfant-Jésus* (1944) and *Catalogue d'oiseaux* (1956–8) and the pre-serial works of Henri Dutilleux (such as *Piano Sonata No. 1*, 1947–8). The elements of serialism in some of my piano writing are also closely related to the atmospheric hexachordal rotation-based piano writing of composers such as Witold Lutosławski and Oliver Knussen. Such atmospheric serialism is present in solo works such as Knussen's *Sonya's Lullaby* Op. 16 (1978–9) and a recent cycle of twelve solo piano pieces by British composer Kenneth Hesketh (a protégé of both Dutilleux and Knussen) called *Horae* (2012), inspired by the twelve goddesses in Greek mythology who personified the hours between sunrise and sunset; a work which I recently heard part of premiered by pianist Claire Hammond at Cardiff University on 12 March 2013.

The influence of Messiaen and Takemitsu continues to be evident in *Human Visions: Civilisations* for symphony orchestra, with the etherealism of Messiaen's *Turangalîla-Symphony* for piano solo, ondes Martenot solo and orchestra (1946–8) and the ambient organicism of works such as Takemitsu's *November Steps* for Japanese biwa, shakuhachi and orchestra (1967) being influential on my orchestral writing. My style also bears a resemblance to the orchestral writing of contemporary British serialists who adopt hexachordal rotation techniques, such as Knussen and Julian Anderson, with *Human Visions: Civilisations* incorporating some of the fantasy-like ambience which is heard in works such as Anderson's *Fantasias* (2009).

In fact *Human Visions: Civilisations* also possesses many scoring characteristics which would more often be associated with a work for chamber orchestra or large chamber ensemble, rather than a work for full orchestra. In this respect, some of the atmospheric orchestral textures bear a resemblance to those seen in Lutosławski's *Novelette* for orchestra (1978–9, which is constructed using aleatoric rhythmic techniques) and Kenneth Hesketh's *Ein Lichtspiel* for large chamber ensemble (2006), whilst some of its colouristic and atmospheric use of resonant/metallic percussion instruments is reminiscent of that seen in George Crumb's *Ancient Voices of Children* for soprano, boy soprano, oboe, mandolin, harp, electric piano and percussion (set to texts by Garcia Lorca, 1970). In addition, the use of irregular tuplets and changing time signatures in *Human Visions: Civilisations* adopts a rhythmic syntax which is not dissimilar from that seen in the chamber works of post-serial composers such as Pierre Boulez and Lutosławski.

On reflection, some aspects of the rhythmic scoring in *Human Visions: Civilisations* did actually prove to be somewhat problematic during rehearsal and performance of the piece with the BBC National Orchestra of Wales in the 2012 Welsh Composers' Showcase (in particular things like the 11:8 ♩ tuplets in the crotales and celesta in m. 4 and m. 9; the 7 ♩ tuplets in the wind and string lines in m. 43<sup>2</sup> [♩ beat],

along with the 14:8 ♪ tuplet in the xylophone on the same beat; and the 5:4 ♩ tuplets in the celesta and strings in m. 55 – especially in view of the fact that these tuplets ratios [with mathematically irrational divisions] are often offset against other musical lines, which simultaneously use regular rhythm in other sections of the orchestra).

During rehearsal, it was necessary for conductor Jac Van Steen to ‘think outside of the box’ and solve these rhythmic problems ‘on the fly’ in order for all sections of the orchestra to stay in time with one another correctly; that said, his solutions in this respect did also prove that the rhythms, as I have written then, are achievable with adequate rehearsal time. For future reference it might be wiser for me to construct or dissect similar rhythmic ideas in a more pragmatic fashion when scoring for full orchestra. I feel that this type of rhythmic scoring can work well in pieces for chamber orchestra or large chamber ensemble in sections where it is not absolutely necessary for the musicians to remain one hundred percent in time with one another (introducing an element of rhythmic indeterminacy, so long as individual musicians don’t lose sight of the pulse!), similarly this approach can work well when scoring for independent elements of a full orchestra (i.e. instrumental lines which are not being doubled elsewhere in the orchestra, such as independent wind or percussion lines or string solos) but might be best avoided on string lines which require unison doubling or homophonic chordal passages where different instruments need to remain totally in time with one another.

*Amber on Black* is an experimental work which explores a variety of extended vocal timbres and techniques, including ‘Sprechgesang’ (pitched speech-song) and ‘Sprechstimme’ (relatively pitched speech), as pioneered in expressionist vocal works such as Schoenberg’s *Pierrot Lunaire* (1912).<sup>2</sup> Although chromatic polymodality and intervallic cells (as pertaining to **Fig. 1.4.1** in the first chapter) form a large part of the pitch organisation scheme in this piece, much of the pitch material has also been generated through adopting some form of twelve-tone serial technique (in particular hexachordal or heptachordal rotation techniques). In addition, it is quite clear that much of *Amber on Black* employs unpitched vocal techniques which add a great deal of coloration and timbral variety to the music, often to the point of removing the need for any sort of tonal pitch material altogether. Other works which combine twelve-tone serialism with unpitched timbral sonority within the historical repertoire include

---

<sup>2</sup> Schoenbergian ‘Sprechgesang’ and ‘Sprechstimme’ techniques are also called for in *Cwyn y Gwynt* for contralto, vibraphone and cello and Sprechgesang is called for in the fourth verse of *Ynys Afallon* for SATB choir and pianoforte (mm. 40–57). In both cases the conventional Schoenbergian method of notating this technique with crosses through the stems (on Sprechgesang [pitched]) and crossed noteheads (on Sprechstimme [spoken]) is employed. However, in *Amber on Black* this method of notation has been revised to correspond with that employed in Ligeti’s *Aventure* (1964), where square noteheads are used to indicate pitched Sprechgesang, whilst the Schoenbergian method of notating spoken Sprechstimme is retained but normally notated on a staff with a single staff line – I found the Ligeti method more pragmatic and easier to read and will continue to adopt this system of notation in future vocal scores.

Schoenberg's *Psalm 130: De Profundis*, Op. 50b for chorus (1950) and Luigi Dallapiccola's *Tempus Destruendi & Tempus Aedificandi* for chorus (1971).

*Amber on Black* also quite evidently moves into the realm of exploring phonetic syllables and sonorities which do not necessarily relate to the construction of real words, as is seen in several of Ligeti's vocal works, including, *Aventures & Nouvelles Aventures* for three solo singers and seven instrumentalists (1962–5).<sup>3</sup> Other avant-garde works of a similar genre which influenced my style include Luciano Berio's *Circles* for female voice, harp and two percussionists (1960), *Sequenza III* for woman's voice (1965) and *Cries of London*<sup>4</sup> for eight voices (1974), Ligeti's *Nonsense Madrigals* for six male voices (1988–93), as well as Karlheinz Stockhausen's *Mikrophonie II* for twelve voices, Hammond organ, 4 ring modulators and tape (Nr. 17, 1965) and *Stimmung* for six vocalists (Nr. 24, 1968). In Stockhausen's *Stimmung*, the conventional pitch material employed is relatively straightforward, being based entirely on pitches relating to the harmonic series on B♭, however the quasi-electronic mutations of vocal timbres on phonetic vowel syllables add much greater sonic complexity than first meets the eye and similar timbral mutations on vowels and semi-vowels (such as [m], [ŋ], [n], [ɲ], [ɳ]...) occur in *Amber on Black*.

---

<sup>3</sup> *Amber on Black* adopts a very similar notation scheme to that seen in Ligeti's *Aventure & Nouvelles Aventure*. This is included as a pull-out at the back of the Litolff/Peters study score publication of both works (Edition Peters 4838 [London: 1964] and Edition Peters 5913 [London: 1966] respectively). However, the phonetic pronunciations in *Amber on Black* adopt the 2005 International Phonetic Alphabet (IPA) spellings rather than the less standardised phonetic spellings adopted by Ligeti in *Aventure* and *Nouvelles Aventure*.

<sup>4</sup> The rapidly repeated syllable combinations in mm. 120–58 of *Amber on Black* (also explained in the notation guide on p. 261 of vol. 1) adopt a similar notation scheme to that seen in *VI to Hélène Pousseur* ("money, penny, come to me...") from Berio's *Cries of London* for eight voices, UE 16828 (Milan: Universal Edition, 1976).

# BIBLIOGRAPHY

# **Bibliography**

## **Contemporary Music-Related Literature:**

Adler, Samuel, *The Study of Orchestration: Third Edition* (New York: W. W. Norton & Co, 2002).

Anderson, Julian, 'Harmonic Practices in Oliver Knussen's Music since 1988: Part I', *Tempo*, New Series, No. 221 (July 2002).

Anderson, Julian, 'Harmonic Practices in Oliver Knussen's Music since 1988: Part II', *Tempo*, New Series, No. 223 (Jan 2003).

Artaud, Pierre-Yves, *Present Day Flutes* (Paris: Gerard Billaudot, 1995).

Artaud, Pierre-Yves, *The Multiphonic Flute* (Paris: Gerard Billaudot, 1995).

Babbitt, Milton, *Milton Babbitt: Words About Music* edited by Stephen Dembski and Joseph N. Strauss (Madison: Wisconsin, 1987).

Babbitt, Milton, 'The String Quartets of Bartók', *Musical Quarterly*, vol. 35, no. 3 (July 1949).

Bartók, Bela, *Bela Bartók Essays*: selected and edited by Benjamin Suchoff (Lincoln; London: University of Nebraska Press, 1992).

Biró, Dániel Péter and Krebs, Harald, *The String Quartets of Béla Bartók: Tradition and Legacy in Analytical Perspective* (New York: OUP, 2014).

Blatter, Alfred, *Instrumentation & Orchestration* (Belmont: Schirmer/Thompson Learning: 1997).

Boretz, Benjamin; Cone, Edward T., ed., *Perspectives on Contemporary Music Theory* (New York: W. W. Norton & Co, 1972).

Boretz, Benjamin; Cone, Edward T., ed., *Perspectives on Schoenberg and Stravinsky* (Princeton, N.J.: Princeton University Press, 1968).

Boulez, Pierre, *Conversation with Deliege* (California: Eulenburg Books, 1976).

Boulez, Pierre, *Orientations* (London: Faber and Faber, 1990).

Brindle, Reginald Smith, *Contemporary Percussion* (Oxford: OUP, 1991).

Cage, John, *Silence* (Connecticut: Wesleyan University Press, 1973).

Cage, John, *John Cage: An Anthology* edited by Richard Kostelanetz (New York: Da Capo, 1991).

Cope, David, *Techniques of the Contemporary Composer* (Belmont: Schirmer/Thompson Learning, 1997).

Cross, Jonathan, *Harrison Birtwistle: Man, Mind, Music* (London: Faber and Faber, 2000).

Deleuze & Guattari, *Anti-Oedipus* (London: Athlone, 2000).

Deleuze & Guattari, *A Thousand Plateaus* (London: Continuum, 2002).

Dick, Robert, *The Other Flute: A Performance Manual of Contemporary Techniques*. 2nd Edition (St Louis: Multiple Breath Music Company, 1989).

Dunsby, Jonathan, *Schoenberg: Pierrot Lunaire* (Cambridge: Cambridge University Press, 1992).

Gould, Elaine, *Behind Bars* (London: Faber Music, 2011).

Griffiths, Paul, *Modern Music and After: Directions Since 1945* (Oxford: OUP, 1995).

Harvey, Jonathan, *The Music of Stockhausen* (London: Faber and Faber, 1975).

Hollington, Barnaby, 'A Polychordal Approach to Serial Harmony – Part I', *Academia.edu* (5 May 2014)

<[www.academia.edu/7657483/A\\_Polychordal\\_Approach\\_to\\_Serial\\_Harmony\\_-\\_Part\\_1](http://www.academia.edu/7657483/A_Polychordal_Approach_to_Serial_Harmony_-_Part_1)>

[accessed 30 November 2014].

Kárpáti, János, *Bartók's String Quartets* [translated from the Hungarian by Fred Macnical] (Budapest: Corvina Press, 1975).

Koozin, Timothy, 'Octatonicism in Recent Solo Piano Works of Toru Takemitsu', *Perspectives of New Music*, Vol. 29, No. 1 (Winter 1991).

Kurtz, Michael, *Stockhausen: A Biography* translated by Richard Toop (London: Faber and Faber, 1992).

Lester, Joel, *Analytic Approaches to Twentieth-Century Music* (London: W. W. Norton & Co: 1989).

Lewin, David, *Generalized Musical Intervals and Transformations* (London: Yale University Press, c.1987).

Losseff, Nicky; Doctor, Jenny, ed., *Silence, Music, Silent Music* (Aldershot: Ashgate, 2007).

Maddocks, Fiona, *Harrison Birtwistle: Wild Tracks* (London: Faber and Faber, 2014).

Messiaen, Olivier, *The Technique of My Musical Language* (Paris: Alphonse Leduc, 1956).

Messiaen, Olivier, *Traité de rythme, de couleur, et d'ornithologie [Treatise on rhythm, colour and ornithology]* (Paris: Alphonse Leduc, c.1994–2002).

Messiaen, Olivier, *Vingt Leçons d'Harmonie [Twenty Harmony Lessons]* (Paris: Alphonse Leduc, 1951).

Milstein, Silvina, *Arnold Schoenberg: Notes, Sets, Forms* (Cambridge: University of Cambridge Press, 1992).

Nyman, Michael, *Experimental Music: Cage and Beyond* (Cambridge: University of Cambridge Press, 1999).

Persichetti, Vincent, *Twentieth Century Harmony* (New York: W. W. Norton & Co, 1961).

Rees, Carla, 'Developing a Repertoire of Extended Techniques for the Kingma System Alto and Bass Flute' (unpublished doctoral thesis, Royal College of Music, 2014).

Rees, Carla, 'Kingma System Alto Flute: A Practical Guide for Composers and Performers' < [altoflute.co.uk](http://altoflute.co.uk) > [accessed 27 July 2013].

Rees, Carla, 'Kingma System Bass Flute: A Practical Guide for Composers and Performers' < [bassflute.co.uk](http://bassflute.co.uk) > [accessed 27 July 2013].

Riog Francoli, Miguel A., *Anthology of Post-Tonal Music* (New York: McGraw-Hill, 2007).

Riog Francoli, Miguel A., *Understanding Post-Tonal Music* (New York: McGraw-Hill, 2007).

Rózsa, Miklós, *Double Life: The Autobiography of Miklós Rózsa* (New York: Hippocrene Books, 1998).

Schoenberg, Arnold, *Style and Idea: Selected Writings of Arnold Schoenberg*, edited by Leonard Stein; with translation [from the German] by Leo Black (California: University of California Press, 2010).

Schoenberg, Arnold, *Theory of Harmony*, translated by Roy Carter (California: University of California Press, 1978).

Stilwell, Robyn J.; Powrie, Phil, ed., *Composing Music for the Screen in Germany and the USSR* (Bloomington, IN, USA: Indiana University Press, 2008).

Stockhausen, Karlheniz, *Stockhausen on Music* (London: Marion Boyars, 1991).

Straus, Joseph N., *Introduction to Post-Tonal Theory*, 3rd Edition (Harlow: Pearson, 2014).



Suchoff, Benjamin, *Bartók's Mikrokosmos: Genesis, Pedagogy and Style* (Lanham, Md: Scarecrow Press, 2002).

Tannenbaum, Mya, *Conversations with Stockhausen* (Oxford: Clarendon Press, 1987).

Vagra, Balint, *Converstion with Xenakis* (London: Faber and Faber, 1996).

Vincent, John, *The Diatonic Modes in Modern Music* (Berkeley: University of California Press, 1951).

Warfield, Gerald, 'The Notation of Harmonics for Bowed String Instruments', *Perspectives of New Music*, Vol. 12, No. 1/2 (1973/4).

Whittal, Arnold, 'Bitonality', *The New Grove Dictionary of Music and Musicians*, second edition, edited by Stanley Sadie and John Tyrrell (2001).

Whittal, Arnold, *The Cambridge Companion to Serialism. Cambridge Introductions to Music* (Cambridge: Cambridge University Press, 2008).

Zukofsky, Paul, 'On Violin Harmonics', *Perspectives of New Music*, Vol. 6, No. 2 (1968).

## **Welsh-Language Poetry, Prose & Celtic Mythology:**

Davies, Sioned, *The Mabinogion* translated with an introduction and notes (Oxford: OUP, 2008).

Evans, J Gwenogvryn, *White Book Mabinogion: Welsh tales and romances reproduced from the Peniarth manuscripts* (Pwllheli: Issued to Subscribers Only, 1907).

Guest, Lady Charlotte, *The Mabinogion* translated into English (London: J. M. Dent, 1910).

Hughes, J. Elwyn, ed., *Eisteddfod Genedlaethol Cymru Casnewydd 2004: Cyfansoddiadau a Beirniadaethau [The Welsh National Eisteddfod in the Newport 2004: Compositions and Adjudications]* (Llandysul: Gomer, 2004).

Hughes, J. Elwyn, ed., *Eisteddfod Genedlaethol Cymru Bala 2009: Cyfansoddiadau a Beirniadaethau [The Welsh National Eisteddfod in the Bala 2009: Compositions and Adjudications]* (Llandysul: Gomer, 2009).

Hughes, J. Elwyn, ed., *Eisteddfod Genedlaethol Cymru Blaenau Gwent a Blaenau'r Cymoedd 2010: Cyfansoddiadau a Beirniadaethau [The Welsh National Eisteddfod in Blaenau Gwent and the Fronts of the Valleys 2010: Compositions and Adjudications]* (Llandysul: Gomer, 2010).

Hughes, J. Elwyn, ed., *Eisteddfod Genedlaethol Cymru Wrexham a'r Fro 2011: Cyfansoddiadau a Beirniadaethau [The Welsh National Eisteddfod in Wrexham and the Vale 2011: Compositions and Adjudications]* (Llandysul: Gomer, 2011).

Hughes, J. Elwyn, ed., *Eisteddfod Genedlaethol Cymru Bro Morgannwg 2012: Cyfansoddiadau a Beirniadaethau [The Welsh National Eisteddfod in the Vale of Glamorgan 2012: Compositions and Adjudications]* (Llandysul: Gomer, 2012).

Hughes, J. Elwyn, ed., *Eisteddfod Genedlaethol Cymru Sir Ddinbych a'r Cyffiniau 2013: Cyfansoddiadau a Beirniadaethau [The Welsh National Eisteddfod in Denbighshire and the Borders 2013: Compositions and Adjudications]* (Llandysul: Gomer, 2013).

Hughes, J. Elwyn, ed., *Eisteddfod Genedlaethol Cymru Sir Gâr 2014: Cyfansoddiadau a Beirniadaethau [The Welsh National Eisteddfod in Carmarthenshire 2014: Compositions and Adjudications]* (Llandysul: Gomer, 2014).

Hughes, J. Elwyn, ed., *Eisteddfod Genedlaethol Maldwyn a'r Gororau 2015: Cyfansoddiadau a Beirniadaethau [The Welsh National Eisteddfod in Montgomeryshire 2015: Compositions and Adjudications]* (Llandysul: Gomer, 2015).

Ifans, Dafydd & Rhiannon, *Y Mabinogion Diweddariad [The Revised Mabinogion]* (Llandysul: Gomer, 1995).

Jones, John-Morris, *Beirniadaeth John Morris-Jones [A Critique of John Morris-Jones]* edited by Dafydd Glyn-Jones (Bangor: Dalen Newydd, 2013).

Jones, John-Morris, *Caniadau* (Oxford: Fox Jones & Co., 1907).

Jones, Nesta Wyn, *Dawns y Sêr [The Stars' Dance]* (Llandysul: Gomer, 1999).

Jones, T. Gwynn, *Caniadau* (Cardiff: Hughes a'i Fab, 1992).

Jones, Tegwyn, *Y Mabinogion: Hud yr Hen Chwedlau Celtaidd [Y Mabinogion: The Magic of the Celtic Tales]/ Yn Seiliedig ar Ddiweddariad Dafydd Ifans a Rhiannon Ifans* (Llanrwst: Gwasg Carreg Gwalch, 2000).

Kinney, Phyllis, *Welsh Traditional Music* (Cardiff: Gwasg Prifysgol Cymru, 2011).

Lewis, Saunders, *Dramâu Saunders Lewis: Y Casgliad Cyflawn. Cyfrol 1/2 [Saunders Lewis' Dramas: The Complete Set. Volumes 1/2]* (Cardiff: Gwasg Prifysgol Cymru, 1996).

Parry, Thomas, *John Morris-Jones, 1864-1929* (Cardiff: Gwasg Prifysgol Cymru, 2011).

Tolstoy, Nikolai, *The Oldest British Prose Literature: The Compilation of the Four Branches of the Mabinogi* (Lampeter: Mellen, 2009).

Williams, Ifor, *Pedair Keinc y Mabinogion: Allan o Lyfr Gwyn Rhydderch [Four Branches of the Mabinogi: Out of the White Book of Rhydderch]* (Cardiff: Gwasg Prifysgol Cymru, 1978).

## **Audio Engineering & Computer Programming:**

Christiansen, Tom, *Perl Cookbook*, (California: O'Reilly Media, 2003).

Deitel, H.M. & Deitel, P.J., *C++: How to Program* (New Jersey: Prentice Hall, 1994).

DeVoe, Jiva, *Objective-C* (California: O'Reilly Media, 2011).

Horstmann, Cay, *Big Java: for Java 7 and 8* (New Jersey: John Wiley & Sons, 2010).

Fedora Project Contributors, *Fedora 18 Musicians' Guide: Audio Creation and Music Software in Fedora Linux*

<[http://docs.fedoraproject.org/en-US/Fedora/18/html/Musicians\\_Guide/index.html](http://docs.fedoraproject.org/en-US/Fedora/18/html/Musicians_Guide/index.html)>  
[accessed 27 May 2013].

Koutsomichalis, Marinos, *Mapping and Visualization with SuperCollider* (Birmingham: Packt Publishing, 2013).

Maden, Charles, *Fractals in Music: Introductory Mathematics for Musical Analysis* (Salt Lake City: High Art Press, 2007).

Manning, Peter, *Electronic & Computer Music* (Oxford: Clarendon Press, 1994).

Puckette, Miller, *The Theory and Technique of Electronic Music* (London: World Scientific Publishing Co., c.2007).

Wilson, Scott; Cottle, David; Collins, Nick, *The SuperCollider Book* (Massachusetts: MIT Press, 2011).

Xenakis, Iannis, *Formalized Music: Thought and Mathematics in Composition* (New York: Pendragon Press, 1992).

## **Music Scores**

Bach J. S., *The Art of Fugue* BWV 1080, edited and annotated by Richard Jones, ISBN: 1854728709 (London: ABRSM, 2002).

Bartók, Bela, *4th String Quartet*, Hawkes Pocket Scores no. 77 (London: Boosey & Hawkes, 1939).

Bartók, Bela, *Mikrokosmos: Progressive Piano Pieces* Volumes 1–6 (London: Boosey & Hawkes, 1940).

Berio, Luciano, *Circles* for female voice, harp, 2 percussion players, from E. E. Cummings: Poems 1923–54, UE 13231 (London: Universal Edition, 1961).

Berio, Luciano, *Cries of London* for eight voices, UE 16828 (Milan: Universal Edition, 1976).

Berio, Luciano, *Folk Songs* for mezzo-soprano and seven instruments, UE 34112 (London: Universal Edition, 1968).

Berio, Luciano, *Sequenza III* per voce femminile, UE 13723 (London: Universal Edition, 1968).

Birtwistle, Harrison, *Earth Dances for orchestra* (London: Universal Edition, c.1986).

Birtwistle, Harrison, *Gawain: An Opera in Two Acts* Volumes 1–2 (London: Universal Edition, c.1990).

Crumb, George, *Ancient Voices of Children* a cycle of songs on texts by Garcia Lorca, for soprano, boy soprano, oboe, mandolin, harp, electric piano, percussion, Edition Peters 66303 (London: Peters, c.1970).

Crumb, George, *Vox Balaenae* for three masked players; electric flute, electric cello, electric piano, Edition Peters 66466 (London: Peters, c.1972).

Harvey, Jonathan, *Wagner Dream: An Opera in Nine Scenes*, vocal score (Harlow: Faber Music, 2012).

Knussen, Oliver, *Whitman Settings* for soprano and orchestra, op 25a, 1991–2, ISBN: 0571514103 (London: Faber Music, 1995).

Ligeti, György, *Atmosphères* for symphony orchestra, UE11418 (Vienna: Universal Edition, c.1963).

Ligeti, György, *Aventure* for three solo singers and seven instrumentalists, study score, Edition Peters 4838 (London: Litolf/Peters, 1964).

Ligeti, György, *Nonsense Madrigals* für sechs Männerstimmen, 46 750 (Mainz: Schott, 1994).

Ligeti, György, *Nouvelles Aventure* for three solo singers and seven instrumentalists, study score, Edition Peters 5913 (London: Litolf/Peters, 1966).

Ligeti, György, *String Quartet No.1: Métamorphoses Nocturnes* 1953–4, ED6476 (Mainz: Scott, 1972).

Ligeti, György, *String Quartet No.2* 1968, ED6639 (Mainz: Scott, 1971).

Messiaen, Olivier, *Catalogue d'Oiseaux, pour piano* Volumes 1–7 (Paris: Alphonse Leduc: c.1964).

Messiaen, Olivier, *Vingt Regard Sur l'Enfant-Jesus, pour piano*, D. & F. 13,230 (Paris: Durand, 1947).

Puw, Guto Pryderi, *Conerto ar Gyfer Obo* [*Concerto for Oboe*] (s.l.: the author, c.2006).

Puw, Guto Pryderi, *Portffolio Cyfansoddi* [*Composition Portfolio*] (unpublished doctoral thesis: University of Wales, Bangor, 2002).

Puw, Guto Pryderi, *Reservoirs* ar gyfer cerddorfa lawn [for full orchestra] (s.l.: the author, c.2004).

Takemitsu, Toru, *Air* for flute, SJ 1096 (Tokyo: Schott Japan Company Ltd., 1996).

Takemitsu, Toru, *And the I knew 'twas Wind* for flute, viola and harp, SJ 1071 (Tokyo: Schott Japan Company Ltd., 1992).

Takemitsu, Toru, *Itinerant—In Memory of Isamu Noguchi* for flute, SJ 1055 (Tokyo: Schott Japan Company Ltd., 1989).

Takemitsu, Toru, *Voice* pour flûte solo, MC 551 (Paris: Salabert Editions, 2001).

Thomas, Mansel, *Cwyn y Gwynt* (*The Wind's Lament*), MT137 (York: Banks Music Publications, 1986).

Thomas, Mansel, *Y Gwyllanod* (*The Seagulls*), MT194 (York: Banks Music Publications, 1986).

## **Audio & Video Recordings:**

Anderson, Julian, *Fantasias*, CD, Schott promotional release (2010).

Harrison, Jonty, *Évidence Matérielle*, CD, empreintes DIGITALes IMED 0052 (2000).

Jones, Heather, *Enaid*, CD, Sain SCD 2442 (2006).

Kraftwerk, *Computer World*, CD, EMI 0777 7460402 4 (1981).

Kraftwerk, *Man Machine*, CD, Capitol 0777 7 46039 2 8 (1978).

Kraftwerk, *Trans Europe Express*, rec. 1977, CD, Capitol CDP 7 46473 2 (1987).

Lang, Fritz, *Metropolis*, rec. 1927, DVD, Eureka EKA40321 (2010).

Macmillan, James, *The Sacrifice*, CD, chan10572 (2008).

Puw, Guto Pryderi, *Reservoirs: Orchestral Works by Guto Pryderi Puw*, BBC National Orchestra of Wales; David Cowley, oboe; Jac Van Steen, conductor, CD, Signum Classics SIGCD 378 (2014).

Thomas, Mansel, *Caneuon Mansel Thomas [The Songs of Mansel Thomas]*, Jeremy Huw Williams, baritone; Nigel Foster, piano, CD, Sain SCD 2208 (1999).

Walton, Ireland and Guto Puw, *Great Proms Premieres*, BBC National Orchestra of Wales, CD, BBC Music Magazine, v. 16, no. 12 (2008).

Williams, Jeremy Huw, *Caneuon Jeremy [Songs for Jeremy]*, Jeremy Huw Williams, baritone; Nigel Foster, piano, CD, Sain SCD 2266 (2000).

# APPENDIX 1

Table of Diatonic Modes Referred to in the Academic Commentary

# Table of Diatonic Modes Referred to in the Academic Commentary

## Modes taken from the standard major scale (on C):

Ionian

Dorian

Phrygian

Lydian

Mixolydian

Aeolian

Locrian

## Modes taken from the melodic minor scale (on C):

melodic minor

Phrygian (#6)

Lydian (#5)

Lydian (b7)

Mixolydian (b6)

Aeolian (b5)

altered scale

## Modes taken from the harmonic minor scale (on C):

harmonic minor

Locrian (#6)

Ionian (#5)

Dorian (#4)

Phrygian (#3)

Lydian (#2)

altered scale (b7)

## Modes taken from the major (b6) scale (on C):

Ionian (b6)

Dorian (b5)

Phrygian (b4)

Lydian (b3)

Mixolydian (b2)

Lydian (#2,#5)

Locrian (b7)

## Modes of the pentatonic scale (on C):

C major pentatonic

C minor pentatonic



# APPENDIX 2

## A Chart of Natural Harmonics on the Contrabass

Compiled alongside corresponding charts for the violin, viola and violoncello provided in the notation guide for #2: *“Utopian Mirror”* from *“Urban Wilderness”* for string quartet (vol. 1, pp. 41–43)

# NATURAL HARMONICS ABOVE THE CONTRABASS C-STRING (with low C extension open)

[The series of natural harmonics on this string may also be transposed upwards by fixing the mechanical locks at the bottom of the string to D $\flat$ , D $\natural$ , E $\flat$  and E $\natural$  respectively]

(Practical Natural Harmonics up to the 9<sup>th</sup> partial)

Cb.

(Natural Harmonics Arpeggio/Glissando up to the 12<sup>th</sup> partial)

Resultant pitches shown  
at sounding pitch

## NATURAL HARMONICS ABOVE THE CONTRABASS A-STRING

(Practical Natural Harmonics up to the 9<sup>th</sup> partial)

(Natural Harmonics Arpeggio/Glissando up to the 12<sup>th</sup> partial)

Resultant pitches shown  
at sounding pitch

## NATURAL HARMONICS ABOVE THE CONTRABASS D-STRING

(Practical Natural Harmonics up to the 9<sup>th</sup> partial)

(Natural Harmonics Arpeggio/Glissando up to the 12<sup>th</sup> partial)

Resultant pitches shown  
at sounding pitch

## NATURAL HARMONICS ABOVE THE CONTRABASS G-STRING

(Practical Natural Harmonics up to the 9<sup>th</sup> partial)

(Natural Harmonics Arpeggio/Glissando up to the 12<sup>th</sup> partial)

Resultant pitches shown  
at sounding pitch

# APPENDIX 3

Handwritten sketches for *Human Visions: “Civilisations”* for symphony orchestra



Handwritten musical score for measures 15 through 100. The score is written on three staves (treble, alto, and bass clefs) and includes various musical notations such as notes, rests, and accidentals. The notation is heavily annotated with performance instructions and instrument names in a handwritten style.

Measures 15-92: Includes parts for 8va, Ob/Fl/Picc., Harmonics, Glock, Tpt. Pedals, BCl., and strings. Measure 92 is marked with a large bracket and a key signature change.

Measures 92-96: Includes parts for 8va, CA/Ob/Picc., Harmonics, Glock, Tpt., and BCl. Measure 96 is marked with a large bracket and a key signature change.

Measures 96-100: Includes parts for 8va, Vl. Ia, Vl. Ib, Vl. IIa, Glock, picc./ob./C.A., strings, Hp. tr., BCl. II, Tpt./hn./Tbn., and Gong. Measure 100 is marked with a large bracket and a key signature change.

Handwritten musical score for measures 114 through 118. The score is written on three staves (treble, alto, and bass clefs) and includes various musical notations such as notes, rests, and accidentals. The notation is heavily annotated with performance instructions and instrument names in a handwritten style.

Measures 114-118: Includes parts for 15ma, Harp, Celeste, Glock, Tpt., Vl./Vla, Woodwind, Mar., strings, and Hp. Measure 114 is marked with a large bracket and a key signature change.

116<sup>3</sup>-118

[118<sup>3</sup>-120]

celeste (8) fl/picc.

Harp Gliss.

Handwritten musical score for measures 116<sup>3</sup>-118 and 118<sup>3</sup>-120. The score is written for multiple staves, including Vln. I, Vln. II, Vla. I, Vla. II, Vc./Cb., Tbn., and Tba. The key signature is D major (two sharps). The time signature is 4/4. The score includes various musical notations such as notes, rests, and dynamic markings. The first system (measures 116<sup>3</sup>-118) shows a complex arrangement of instruments, with Vln. I and Vln. II playing a melodic line, Vla. I and Vla. II providing harmonic support, and the lower strings playing a rhythmic pattern. The second system (measures 118<sup>3</sup>-120) continues the melodic and harmonic development, with Vln. I and Vln. II playing a more active role. The score is written in a clear, legible hand, with many annotations and markings.

120

120<sup>4</sup>

Handwritten musical score for measures 120 and 120<sup>4</sup>. The score is written for multiple staves, including Vln. I, Vln. II, Vla. I, Vla. II, Vc./Cb., Tbn., and Tba. The key signature is D major (two sharps). The time signature is 4/4. The score includes various musical notations such as notes, rests, and dynamic markings. The first system (measures 120) shows a complex arrangement of instruments, with Vln. I and Vln. II playing a melodic line, Vla. I and Vla. II providing harmonic support, and the lower strings playing a rhythmic pattern. The second system (measures 120<sup>4</sup>) continues the melodic and harmonic development, with Vln. I and Vln. II playing a more active role. The score is written in a clear, legible hand, with many annotations and markings.

124<sup>4</sup>

Handwritten musical score for measures 124<sup>4</sup>, 125, and 125<sup>3</sup>. The score is written for multiple staves, including Vln. I, Vln. II, Vla. I, Vla. II, Vc./Cb., Tbn., and Tba. The key signature is D major (two sharps). The time signature is 4/4. The score includes various musical notations such as notes, rests, and dynamic markings. The first system (measures 124<sup>4</sup>) shows a complex arrangement of instruments, with Vln. I and Vln. II playing a melodic line, Vla. I and Vla. II providing harmonic support, and the lower strings playing a rhythmic pattern. The second system (measures 125) continues the melodic and harmonic development, with Vln. I and Vln. II playing a more active role. The third system (measures 125<sup>3</sup>) shows a complex arrangement of instruments, with Vln. I and Vln. II playing a melodic line, Vla. I and Vla. II providing harmonic support, and the lower strings playing a rhythmic pattern. The score is written in a clear, legible hand, with many annotations and markings.



126

$\pm \pm \pm$  ] *piece* 126<sup>4</sup>

127

127<sup>2</sup>

Handwritten musical score for measures 126, 127, and 127<sup>2</sup>. The score is written on three staves. The top staff is a treble clef, the middle is a treble clef, and the bottom is a bass clef. The notation includes various notes, rests, and accidentals. Labels include "Vl., Fl.", "Vla., Vc.", and "Cb.". There are also some markings like "b" and "b#" above notes.

129

Handwritten musical score for measures 129 and 130. The score is written on three staves. The notation includes various notes, rests, and accidentals. Labels include "Eb" and "Cb.". There are also some markings like "b" and "b#" above notes.

131

134 15 135  
[ # ] *cel.* [ 15-7 ] *piece*

Handwritten musical score for measures 136, 137, and 138. The score is written on three staves. The notation includes various notes, rests, and accidentals. Labels include "Hp.", "Vla.", "Vc.", "Cel.", "Tpt. 2, 3", "Hn.", "B.C.", "Ob./Cl.", and "Cl. 2". There are also some markings like "b" and "b#" above notes.

Handwritten musical score for measures 131 and 132. The score is written on three staves. The notation includes various notes, rests, and accidentals. Labels include "Tpt. 3" and "strings/brass".

Handwritten musical score for measures 133 and 134. The score is written on three staves. The notation includes various notes, rests, and accidentals. Labels include "Hp.", "Vc.", and "B.C.". There are also some markings like "b" and "b#" above notes.





# APPENDIX 4

Handwritten sketches for #2: “*Utopian Mirror*”  
from “*Urban Wilderness*” for string quartet

## "Inner Utopias" Sketch

low → ham.  
leave: Va. {  
high → Vc. {  
leave: Vln. {  
low → Vln. {  
high → Va. {  
low[end]

## Section 1]

- All harm.
- VLn II  $\rightarrow$  low con sord.
- Va  $\rightarrow$  low con sord.
- VLn I  $\rightarrow$  low con sord.
- Vc.  $\rightarrow$  low con sord.

arm.  $\rightarrow$  low con sord.  
x c. 0:12  
urapose single/similar  
mony in harmonics with a  
inuous harmonic progression  
the low con sord.

Tempi]

- 1] Colouristic, Sustained (Fluid &)
- 2] Subdued but Expressive → poco a poco allargando
- 3] Radiant & Very Colouristic → Tempo Primo

## Use of Harmonics

Use of Harmonics		Section 1	Section 2	Section 3 (Climax)	Section 3 (Coda)
Natural	Vln. Vg. Vc.	upto 5 <sup>th</sup> partial upto 5 <sup>th</sup> /6 <sup>th</sup> partial upto 6 <sup>th</sup> partial	upto 6 <sup>th</sup> partial upto 7 <sup>th</sup> partial upto 8 <sup>th</sup> partial	upto 8 <sup>th</sup> partial upto 9 <sup>th</sup> partial upto 12 <sup>th</sup> partial	as S2 { hints of upper partial from climax
Artificial Harmonics		3 <sup>rd</sup> /4 <sup>th</sup> partial	5 <sup>th</sup> /6 <sup>th</sup> partial	4 <sup>th</sup> -6 <sup>th</sup> partial	4 <sup>th</sup> -6 <sup>th</sup> partial

Section 1 (All harmonics)

Extend by 1 beat  
= 1 bar of  $\frac{2}{4}$

2 4 5

Astring

$V_a$  = change to G string  
 $V_c$  = change to C string

Con sord. section  $\rightarrow$



# "Inner Utopias" Notes (2)

Use of  $\frac{5}{8}$  [?] +  $\frac{4}{4}, \frac{3}{4}, \frac{2}{4}, \frac{5}{4}$

## Section 2

m.1  $\rightarrow \left\{ \begin{array}{l} \text{Vln I \& Vln II} \\ \text{Vln I \& Vc.} \end{array} \right. \rightarrow \left[ \frac{2+5}{4+8} \right] \rightarrow \text{or } \left[ \frac{4+5}{8} \right]$

m.2  $\rightarrow \left\{ \begin{array}{l} \text{Vln I} \\ \text{Vln II} \end{array} \right. \rightarrow \text{semiquaver fragments} \rightarrow \text{m.3} \rightarrow \text{semiquaver fragments}$

Swap!

m.4 Vln II  $\rightarrow$   $\left[ \frac{2+5}{4+8} \right]$  up maj 3rd (root = E $\flat$ )

m.5 Vln I  $\rightarrow$   $\left[ \frac{2+5}{4+8} \right]$  offset on b $\flat$  3

m.6 Vla  $\rightarrow$   $\left[ \frac{2+5}{4+8} \right]$  on b $\flat$  1

m.7 Vc.  $\rightarrow$   $\left[ \frac{2+5}{4+8} \right]$  on b $\flat$  4 = 7:6 f or 8:6 f [?] up maj 3rd (root = A $\flat$ )

m.8 Vln I  $\rightarrow$  double  $\left[ \frac{2+5}{4+8} \right]$  up maj 3rd (root = high C)

m.9 Vln II  $\rightarrow$   $\left[ \frac{2+5}{4+8} \right]$

m.10 Vla  $\rightarrow$   $\left[ \frac{2+5}{4+8} \right]$

m.11 Vc.  $\rightarrow$   $\left[ \frac{2+5}{4+8} \right]$

m.12  $\left[ \frac{2+5}{4+8} \right] \rightarrow$  Vln I (b $\flat$  1) + Vla (b $\flat$  3)

m.13  $\left[ \frac{2+5}{4+8} \right] \rightarrow$  Vln II (b $\flat$  2) + Vc. (b $\flat$  4)

m.14/15  $\left[ \frac{2+5}{4+8} \right] \times 4 + \left[ \frac{2+5}{4+8} \right] \rightarrow$  Vc. rest in  $\frac{5}{8}$

$\left[ \frac{2+5}{4+8} \right] \rightarrow$  all 4 =  $\left[ \frac{2+5}{4+8} \right] \times 2 + \left[ \frac{2+5}{4+8} \right]$  in Vln II + Vc.  $\rightarrow$  2-note trill

$\left[ \frac{2+5}{4+8} \right] \rightarrow$  in Vln II  $\rightarrow$  i.e. 3s vs. 4s etc...

## Section 2 (Transition to Arpeggiando)

m.1  $\rightarrow \left[ \frac{2+5}{4+8} \right]$  in Vc.  $\rightarrow \left[ \frac{2+5}{4+8} \right]$  in Vln/Vla

m.2  $\rightarrow$  12:8 f offset in Vln I

m.3  $\rightarrow$  6 f in Vla (down)

m.4  $\rightarrow$  12:8 f offset in Vln II  $\rightarrow$  12-note Arpeggiando pattern on  $\left[ \frac{2+5}{4+8} \right]$

m.5  $\rightarrow$  Vc. = senza sord. Vla =  $\left[ \frac{2+5}{4+8} \right]$

m.6  $\rightarrow$  Vc. = harm (5th/6th partial + upper natural harmonics) [?]  $\downarrow$  cantabile, dolce.

m.7  $\rightarrow$  Vln I = 4 f  $\rightarrow$  12-note Arpeggiando pattern on  $\left[ \frac{2+5}{4+8} \right]$  mf vs. p

m.8  $\rightarrow$  Vla = senza sord.

m.9  $\rightarrow$  Vla = harm.  $\rightarrow \left[ \frac{2+5}{4+8} \right]$

m.10  $\rightarrow$  Vln II = senza sord

m.11  $\rightarrow$  Vln II = harm. etc... Vla - Vln II - Vc.

m.12  $\rightarrow$  Vln I = senza sord

m.13  $\rightarrow$  Vln I = harm. etc... II - Vc - Vla - I

m.14  $\rightarrow$   $\left[ \frac{2+5}{4+8} \right]$

## Arpeggiando Rhythmic Patterns

Stage 1  $\left[ \frac{2+5}{4+8} \right]$  Vln I, Vln II, Vla, Vc. etc...

Stage 2  $\left[ \frac{2+5}{4+8} \right]$  Vln I, Vln II, Vla etc...

Stage 3  $\left[ \frac{2+5}{4+8} \right]$  Vln I, Vln II etc...

Stage 4  $\left[ \frac{2+5}{4+8} \right]$  Vln I etc...

After  $\left[ \frac{2+5}{4+8} \right]$

m.15 =  $\left[ \frac{2+5}{4+8} \right]$  [d-] ]



originally  $\left[ \frac{4}{2} \right]$  "Inner Utopias" Notes (3)

Section 3  $\left[ 2 \times \frac{2}{2} \right]$  as per previous section but (5<sup>th</sup>/6<sup>th</sup> partial + upper natural harmonics)

mm 1-6 in  $\left[ \frac{2}{2} \right]$   $\begin{cases} m.1 \rightarrow I/II = \text{General (very high!)} , Vc. = G \text{ string} , Vla = C \text{ string} \\ m.2 \rightarrow I = \text{ } , Vc. = C \text{ string} , Vla = G \text{ string} , II = A \text{ string} \\ m.3 \rightarrow II/Vla/Vc. = \text{ } , I = E \text{ string} \end{cases}$

The following bars change to  $\frac{3}{4}$  and combine upper partials with timbral effects and extended techniques...

Section 3  
(Climax)

2  $\rightarrow$   $\text{♩} \text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$   
2 or 0

Harmonic Glissandi String Order:

all [msp  $\rightarrow$  mst]  
gliss/trem  $\rightarrow$  sp.

$\text{♩} \text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$   
on 1<sup>st</sup>, on 1<sup>st</sup>  
2d beat, 2d beat  
 $\uparrow \quad \uparrow$   
I/Vc. II/Vla  
E+D, A+G

Increase  
rate of  
change

Stage	Vln I	Vln II	Vla	Vc.
1	D-string <sup>(se)</sup>	G-string	C	C
2	A	"(se)	"(se)	"
3	"(gliss)	D(se)	G(se)	"
4	E	A	D	"(se)
5	"	"(gliss)	"(se)	G(se)
6	E(trem.)	E(trem.)	A(se)	D(se)
7	E(harm.)	E(trem.)	A(trem.)	A
8	E(harm.)	A(harm.)	D(se)	A(trem.)
9	"	"	G(harm.)	A(se)
10	"	"	"	D(harm.)

Vln I seagull effect on D  
Vln II/Vla seagull effect on G/C  
I-gliss/trem on A, Vln II/Vla seagull effect on D/G  
Vc. seagull effect on C  
II-gliss/trem on A, Vla/Vc. seagull effect on D/G  
Vc. seagull effect on D, Vla seagull effect on A  
Vla seagull effect on D  
Vc. seagull effect on A

Section 3  
(Harmonics)

$\left[ 2d+3d \right]$

$\left[ \frac{5}{2} \right]$  m.1 II - Vc. - Vla - I  
 $\left[ \frac{2}{2} \right]$  m.2 "  
m.3 "

$\text{♩} \text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$   
double time!  
(very high pitches)  
" - No pause [?]

m.4 5x d in  $\text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$  II - Vc. - Vla - I

m.5 5x d in  $\text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$  Vla - I - II/Vc.

$\left[ \frac{4}{2} \right]$  m.6  $\text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$  II/Vc. - I/Vla

m.7  $\text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$  Homophonic

m.8  $\text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$  "

m.9  $\text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$   $\rightarrow \left[ \frac{2}{2} \right]$  Tempo Primo (Melancholy, (colouristic, Sustained)  
J = c. 66

Section 4  
(Low Strings/  
Recap.)

$\text{♩} \text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$  =  $\text{♩} \text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$

$\text{♩} \text{♩} \text{♩} \text{♩} \text{♩} \text{♩}$  = f 10:8f + some nested triplets

pizz. = sul pont. tremolo

Section 4  
(Vla Solo/  
Coda)

Vla solo:  $\rightarrow \text{H} [?]$

5f pizz.

3f pizz.

3f stopped

5f stopped

6f

10:8f x 2 [1 nested 3f + f# trem]

Flautando  
sempre

$\left[ \frac{1}{2} \right] \left[ \frac{3}{2} \right]$

$\left[ \frac{1}{2} \right] \left[ \frac{3}{2} \right]$

$\left[ \frac{1}{2} \right] \left[ \frac{3}{2} \right]$

$\left[ \frac{1}{2} \right] \left[ \frac{3}{2} \right]$

$\left[ \frac{1}{2} \right] \left[ \frac{3}{2} \right]$

$\left[ \frac{1}{2} \right] \left[ \frac{3}{2} \right]$

$\left[ \frac{1}{2} \right] \left[ \frac{3}{2} \right]$

3f stopped trem.  $\rightarrow$  4-note app.  $\downarrow$

2-3-4-note trem.  $\rightarrow$  C-string

2-note harm. trem. [sul pont.]

3d harm. gliss.

3f pizz. gliss.

3f double stops (pizz.)  $\leftarrow$  6<sup>th</sup>/7<sup>th</sup> [?]

harm. double stop [flau.]

1x4 double stops

I+II+Vc.

$\downarrow$

harm. trem.

$\downarrow$

harm.

$\downarrow$

double stops

# "Inner Utopias" Sketch(1)

E $\flat$  Lydian  
C $\sharp$  Dorian  
D $\sharp$  Melodic Minor

D+A+E Lydian [F, C, G]

→ ∞

Handwritten musical notation for the first system, featuring staves for Violoncello (Vc.), Viola (Va.), and Violins I & II (Vln. I, Vln. II). The notation includes various accidentals and dynamic markings such as *9-note*, *12-note*, and *15-note*.

Handwritten musical notation for the second system, featuring staves for Violoncello (Vc.), Viola (Va.), and Violins I & II (Vln. I, Vln. II). The notation includes various accidentals and dynamic markings such as *F $\sharp$  Aedrian + A $\flat$  Lydian*, *Original 9-note*, *cresc.*, and *5*.

Handwritten musical notation for the third system, featuring staves for Violoncello (Vc.), Viola (Va.), and Violins I & II (Vln. I, Vln. II). The notation includes various accidentals and dynamic markings such as *12-note*, *5*, *mf*, *sf*, and *pp*.

Handwritten musical notation for the fourth system, featuring staves for Violoncello (Vc.), Viola (Va.), and Violins I & II (Vln. I, Vln. II). The notation includes various accidentals and dynamic markings such as *mp*, *p*, and *3*.

Handwritten musical notation for the fifth system, featuring staves for Violoncello (Vc.), Viola (Va.), and Violins I & II (Vln. I, Vln. II). The notation includes various accidentals and dynamic markings such as *Artificial*, *Natural*, *Retrograde of mm. 1-3*, and *b $\flat$* .

Handwritten musical notation for the sixth system, featuring staves for Violoncello (Vc.), Viola (Va.), and Violins I & II (Vln. I, Vln. II). The notation includes various accidentals and dynamic markings such as *Lowest practical harmonics*, *All harmonics*, and *5*.



# "Inner Utopias" Sketch (2) [3+2]

[6/2 [3+3] [?]]

(J=1) [3/4 d 1/2]

A $\flat$   
G $\sharp$   
F $\sharp$   
C $\sharp$

Handwritten musical score for the first system of "Inner Utopias" Sketch (2). The system consists of four staves. The top two staves are in treble clef, and the bottom two are in bass clef. The music includes various notes, rests, and accidentals. There are annotations like "dolcis." and "5/2".

Handwritten musical score for the second system of "Inner Utopias" Sketch (2). The system consists of four staves. The top two staves are in treble clef, and the bottom two are in bass clef. The music includes various notes, rests, and accidentals. There are annotations like "New Tempo!", "poco -> poco -> poco", "sp. st. s.p.", and "Vln II -> con sord.".

Handwritten musical score for the third system of "Inner Utopias" Sketch (2). The system consists of four staves. The top two staves are in treble clef, and the bottom two are in bass clef. The music includes various notes, rests, and accidentals. There are annotations like "Vc. -> con sord." and "Vla -> con sord.".

Handwritten musical score for the fourth system of "Inner Utopias" Sketch (2). The system consists of four staves. The top two staves are in treble clef, and the bottom two are in bass clef. The music includes various notes, rests, and accidentals. There are annotations like "Vln I -> con sord." and "Vc. -> con sord.".

Handwritten musical score for the fifth system of "Inner Utopias" Sketch (2). The system consists of four staves. The top two staves are in treble clef, and the bottom two are in bass clef. The music includes various notes, rests, and accidentals. There are annotations like "Vln I basic transpositions", "Vln II", "Vla", and "Vc.".

# "Inner Utopias" Sketch (3)

Handwritten musical notation for the first system, featuring treble and bass staves. Annotations include:  $3 Bb$ ,  $B$ ,  $8vb [?]$ ,  $[2] [?]$ ,  $[*Use: H \& N]$ , and **Swap!**. Fingering numbers (3, 5, 6) and accidentals are present throughout the system.

Handwritten musical notation for the second system. Annotations include:  $D\sharp$ ,  $H$ ,  $5$ ,  $3$ ,  $2$ ,  $5$ ,  $8$ ,  $5$ ,  $3$ ,  $4:3-8:6$ ,  $[2] [?]$ ,  $VmI/II$ , and **swap?**. Fingering numbers and accidentals are present throughout the system.

Handwritten musical notation for the third system. Annotations include:  $9:8f$ ,  $5$ ,  $3$ ,  $2$ ,  $5$ ,  $8$ ,  $5$ ,  $3$ ,  $4:3-8:6$ ,  $8:6f (H) (3) 6 (3)$ ,  $5$ ,  $8:6$ , and **[spellim b?]**. Fingering numbers and accidentals are present throughout the system.

Handwritten musical notation for the fourth system, starting with the instruction **espress-e cantab.** and  $4/4$ . Annotations include:  $Vc.$ ,  $5$ ,  $3$ ,  $2$ ,  $5$ ,  $8$ ,  $5$ ,  $3$ ,  $Vla (ad)$ ,  $1st$ ,  $4/4$ ,  $5$ ,  $3$ ,  $5$ ,  $3$ , and **swap!**. Fingering numbers and accidentals are present throughout the system.

Handwritten musical notation for the fifth system. Annotations include:  $Vc. = [DE\sharp] [?]$ ,  $5$ ,  $3$ ,  $2$ ,  $5$ ,  $8$ ,  $5$ ,  $3$ ,  $8:6f$ ,  $5$ ,  $3$ ,  $5$ ,  $3$ ,  $4:3$ ,  $8:6f$ , and  $\sharp 3 [?]$ . Fingering numbers and accidentals are present throughout the system.

Handwritten musical notation for the sixth system, starting with the instruction **Vla basic Transpositions**. Annotations include:  $VlmI$ ,  $Vla$ ,  $5$ ,  $3$ ,  $2$ ,  $5$ ,  $8$ ,  $5$ ,  $3$ , and  $\sharp 3$ . Fingering numbers and accidentals are present throughout the system.

Handwritten musical notation for the seventh system. Annotations include:  $5f \rightarrow$ ,  $[?]$ ,  $8va$ ,  $5$ ,  $6$ ,  $all$ ,  $mp$ ,  $5$ ,  $3$ ,  $3$ ,  $3$ ,  $1$ ,  $3$ ,  $4:3$ ,  $7:6$ , **(WOODSTOCK)**,  $G\flat [?]$ ,  $D\sharp, C\sharp, A\sharp$ , and  $5$ . Fingering numbers and accidentals are present throughout the system.

[illegible]





# "Inner Utopias" Sketch(6)

Handwritten musical score for "Inner Utopias" Sketch(6). The score is written on multiple staves, including staves for Violin I (Vln I), Violin II (Vln II), Viola (Vla), Violoncello (Vc), and Double Bass (Cb). The notation includes various musical symbols such as notes, rests, accidentals, and dynamic markings.

Key markings and annotations include:

- Measure numbers: [S2], [S3], [S4], [S5], [S6], [S7], [S8], [S9], [S10].
- Instrument parts: Vln I=I, Vln II=II, Vla=I, Vc=IV, Vc=II, Vc=I, Vc=III, Vc=II.
- Dynamic markings:  $f > pp$ ,  $f$ ,  $pp$ .
- Articulation:  $8va$ ,  $15ma$ ,  $22ma$ .
- Performance instructions: "spell as A#-F#?", "Retrograde", "2nd partial in Vc. [+retrograde triplet]", "f# in Vc. | minims!!".
- Chordal structures:  $B, A, G, \#C, \#B$ ,  $G, \#G, C, \#G$ ,  $E, F, \#G, A, \#C$ ,  $E, F, \#G, A, \#C$ ,  $E, F, \#G, A, \#C$ ,  $E, F, \#G, A, \#C$ .
- Other markings:  $[+1?]$ ,  $[+2?]$ ,  $[+3?]$ ,  $[+4?]$ ,  $[+5?]$ ,  $[+6?]$ ,  $[+7?]$ ,  $[+8?]$ ,  $[+9?]$ ,  $[+10?]$ ,  $[+11?]$ ,  $[+12?]$ ,  $[+13?]$ ,  $[+14?]$ ,  $[+15?]$ ,  $[+16?]$ ,  $[+17?]$ ,  $[+18?]$ ,  $[+19?]$ ,  $[+20?]$ ,  $[+21?]$ ,  $[+22?]$ ,  $[+23?]$ ,  $[+24?]$ ,  $[+25?]$ ,  $[+26?]$ ,  $[+27?]$ ,  $[+28?]$ ,  $[+29?]$ ,  $[+30?]$ ,  $[+31?]$ ,  $[+32?]$ ,  $[+33?]$ ,  $[+34?]$ ,  $[+35?]$ ,  $[+36?]$ ,  $[+37?]$ ,  $[+38?]$ ,  $[+39?]$ ,  $[+40?]$ ,  $[+41?]$ ,  $[+42?]$ ,  $[+43?]$ ,  $[+44?]$ ,  $[+45?]$ ,  $[+46?]$ ,  $[+47?]$ ,  $[+48?]$ ,  $[+49?]$ ,  $[+50?]$ ,  $[+51?]$ ,  $[+52?]$ ,  $[+53?]$ ,  $[+54?]$ ,  $[+55?]$ ,  $[+56?]$ ,  $[+57?]$ ,  $[+58?]$ ,  $[+59?]$ ,  $[+60?]$ ,  $[+61?]$ ,  $[+62?]$ ,  $[+63?]$ ,  $[+64?]$ ,  $[+65?]$ ,  $[+66?]$ ,  $[+67?]$ ,  $[+68?]$ ,  $[+69?]$ ,  $[+70?]$ ,  $[+71?]$ ,  $[+72?]$ ,  $[+73?]$ ,  $[+74?]$ ,  $[+75?]$ ,  $[+76?]$ ,  $[+77?]$ ,  $[+78?]$ ,  $[+79?]$ ,  $[+80?]$ ,  $[+81?]$ ,  $[+82?]$ ,  $[+83?]$ ,  $[+84?]$ ,  $[+85?]$ ,  $[+86?]$ ,  $[+87?]$ ,  $[+88?]$ ,  $[+89?]$ ,  $[+90?]$ ,  $[+91?]$ ,  $[+92?]$ ,  $[+93?]$ ,  $[+94?]$ ,  $[+95?]$ ,  $[+96?]$ ,  $[+97?]$ ,  $[+98?]$ ,  $[+99?]$ ,  $[+100?]$ .

"Inner Utopias" Sketch (7)

Handwritten musical score for "Ver Olopias Sketch (7)". The score is written on multiple staves, including staves for Violin I/II, Violoncello/Double Bass, and Violoncello/Violoncello. The notation includes various musical symbols such as notes, rests, accidentals, and dynamic markings. Key annotations include:

- "touch 3rd" and "touch 4th" with arrows pointing to specific notes.
- "8va" indicating an octave shift.
- "just Vln II in this bar!" with an arrow pointing to a specific staff.
- "10:8f" indicating a tempo or performance instruction.
- "Vla = %" and "Vla/Vc = %" indicating performance instructions for the Violoncello/Double Bass.
- "cresc." indicating a crescendo.
- "etc..." indicating a repeat or continuation.
- "sff sff" indicating fortissimo dynamics.
- "Bb" indicating a key signature change.
- "WOODSTOCK" at the bottom center.



# "Inner Utopias" Sketch (8)

all sul pont.

# "Inner Utopias" Sketch(9)

$\left[\frac{2}{2}\right] \times 2 + \left[\frac{3}{4}\right] \text{ Vln} \rightarrow \text{etc...}$   
 $\left[\frac{2}{2}\right] + \left[\frac{5}{4}\right] \rightarrow \text{Vla gliss.}$   
 $\left[\frac{5}{2}\right] \times 2 \rightarrow \text{gliss down to } f\#$   
 $\left[\frac{2}{2}\right] \times 4?$

**Vla solo**

$[tr-] [?]$   
 $b\flat$   $b\flat$

$\begin{bmatrix} b & b & b \\ 3 \end{bmatrix}$

$\begin{bmatrix} b & b & b & b \\ 4 \end{bmatrix}$

$\text{Vla}$   
 $\text{Vln II}$   
 $\text{Vc.}$   
 $\text{Vln I}$

$\text{I+II+Vc.}$   
 $\text{Vla Octachord}$   
 $\text{Vc.}$   
 $\text{Vln II}$   
 $\text{Vln I}$

$\text{pizz.}$   
 $\text{gliss.}$   
 $\text{or } f/G\flat[?]$   
 $\text{Double Harmonics Section}$

$\text{Vc. Vln I II}$   
 $\text{Vc. Vln I II}$   
 $\text{Vc. I II Vc. I II}$   
 $\text{Vla hexachord}$   
 $\text{etc...}$   
 $\text{Pattern \#1 in } [J_s]$   
 $\text{Pattern \#2 in } [J_s]$   
 $\text{Vla solo}$   
 $\text{arco e flautando}$

$\text{I+II+Vc.}$   
 $\text{Vla solo}$



# APPENDIX 5

Handwritten sketches for “*Amber on Black*” for solo SATB singers

# "Amber on Black" Notes (1)

Prelude - #1 - Interlude #1 - #2 - Interlude #2 - #3 - Postlude

Prelude:] → Changing pitches

1.1] [Rhythmic, Articulate] → Same pitches, / speech

1.2] More restrained → Question/Answer [Expressive but with Irony] Sensitive [?]

[Rhythmic but

1.3] Steady, [Gradually Accelerating] → Building in intensity

1.4] [Colourful, Atmospheric] → speed changes → question at the end only [senza misura / without absolute timing]

mf

Prelude:]  $\overset{a}{\text{Amber}} / \overset{a}{\text{Anber}} \quad \overset{a}{\text{mber}} / \overset{a}{\text{nber}} \quad \text{A [mutate vowel]}$

blak bla kah kalb  
lak alb

flitten on a 'r' 'h'

Song → Sprechg. → Sprechs. → Speech

mf/f

1.1] Sprechstimme / Speech → hinting at rhythm (keep syllables from "blak" for rhythm)

f 1.2] Speech → constant rhythm  $(\dot{I}=\dot{J}) \quad \frac{6}{8} + \left[\frac{6}{8} \times 2\right] + \left[\frac{5}{8} \times 2\right] \rightarrow \left[\frac{4}{4} \text{ [layered]} \times 2\right] \rightarrow \left[\frac{3}{4} \text{ [homo phonic]} \times 2\right]$  overlap → " Double-time from 6/8

contrast with f (syllables from 1.2)

p-f 1.3] art tik tion railed  
ait tik ſən reild

Speech → Sprechs. → Sprechg. → pitched with cluster harmonies, gliss., (a [?]) angular/articulate vocal leaps

mf/np

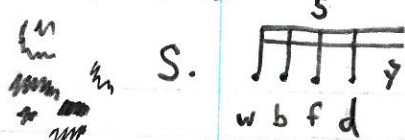
2.1] Song / Sprechg. / gliss → sop. + ten. melodies

[p-ppp]

2.2] Whisper → Murmur → Speech → Murmur → Whisper  
sprechs.

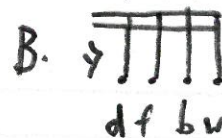


Word buffeted word:



b o d b r t

"Amber on Black" Notes (2)



Articulation: [sən 3ən 2ən sən] → use later on "emotion" & "motion"!

"Derailed": ff gliss. (Whole tone) p → pppp  
de-railed

f(2) 2 syllables!  
Reild

Verse 3

"Track": 1.3  
trak trac trac trac tra[tʃ]  
[s] [short final syllable]

[trace] → treɪə treɪs treɪʃ  
treɪz treɪʒ  
lisp! unpitched pitched  
[long final syllable]

Articulation: 1. full word, pitched, lightly articulated  
2. Gradually mutate individual syllables from the word  
3. full word, unpitched, sharply articulated → mix speech & sprechstimme  
4. → full word, pitched, heavily articulated & angular

"Derailed": 5. As above, w. melisma →  $\frac{3}{4} \times 1$   
6. Flutter on "r" and "R" + "reild"  $[\frac{3}{4} \times 1] + [\frac{4}{4} \times 1]$   
7. Climax with glissandi → fade to p → pppp at end + double barline

1.3.2] a: - tɪk - jɜ - leɪ - sən

S. a: - tɪk - jɜ - laɪ - sən

A. eɪ - tɪk - jɜ - leɪ - sən

T. iɪ - tɪk - jɪ - li - sən

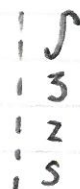
B. uɪ - tɪk - ju - laɪ - sən

Sa: - n

Se: - m

Si: - g

Su: - s



2.1]

Q: Alto (Sprechg.)

A: Tenor (Sprechg.)

Q: Bass (nat. + melisma)

A: Sop. (nat. + melisma)

→ mutated → Q: Tenor (nat.)

A: Alto (nat.)

Q: Sop. (sprechg. + melisma)

A: Bass. (sprechg. + melisma)

hav tu: set I'məʊs(ə)n In 'məʊs(ə)n

T. hav te: set eɪ-məʊ-sən ɛn məʊ-sən

A. hav to sot aɪ-məʊ-ʒən on məʊ-ʒən

S. hav ta: sat aɪ-məʊ-zən ən məʊ-zən

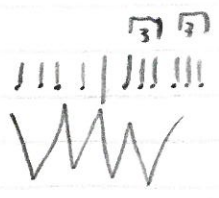
B. hɔɪ ti: sɪt iɪ-məʊ-sɪn In məʊ-sɪn

Am  
Her

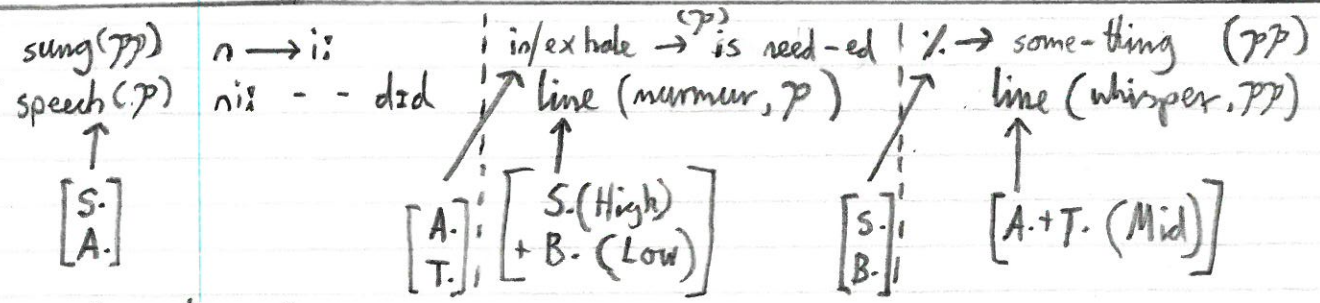
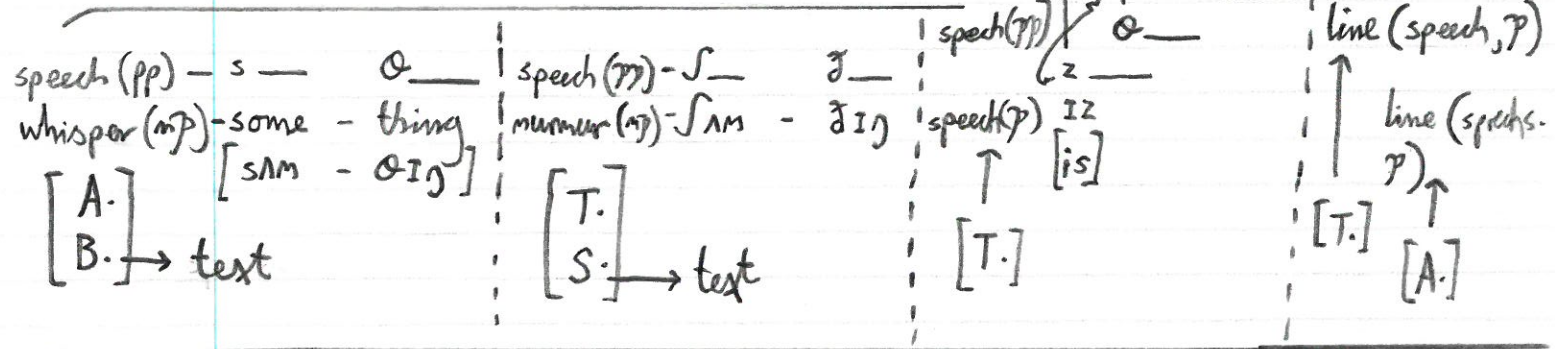
# "Amber on Black" Notes (3)

	Spredng.	Spechs	Speech
S	✓	✓	✓
A	✓	✓	✓
T	✓	✓	✓
B	✓	✓	✓

[T. swaps to  
"s" syllable]

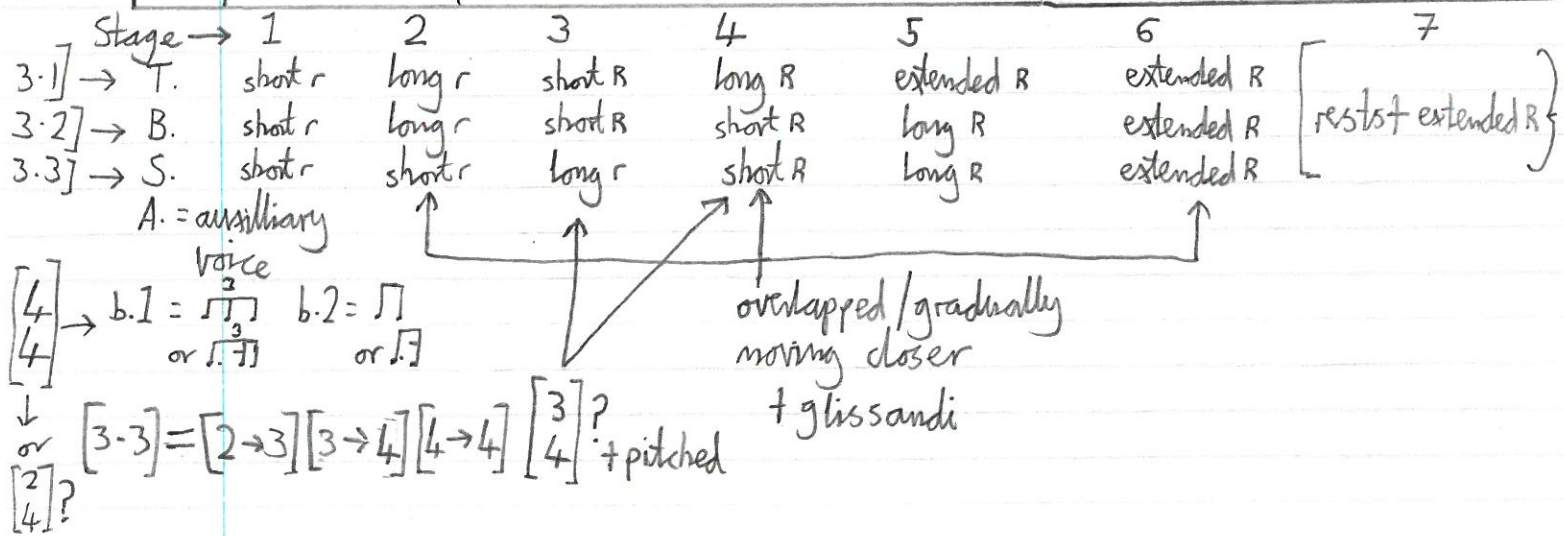


[B. A. S.]



\* pad section out  
w. tongue clicks  
& in/exhale on "h"

from m. 43		Murmur	Whisper	Inhale	Exhale	Inhale & Speak	Exhale & Speak	Tongue Click
	S	✓	✓ [V. 3] + 4.3	✓	✓	✓ [V. 4.3]	✓ [V. 4.3]	✓
(→ 54)	A	✓ [V. 3] + 4.3	✓	✓	✓	✓	✓	✓
	T	✓ [V. 3] + 4.3	✓	✓	✓	✓ ← ✓ [V. 4.3]	✓	✓
	B	✓	✓	✓	✓	✓	✓	✓





# "Amber on Black" Notes (4) "Rhythmic, Mechanical, Machine-Like"

f or ff throughout! → fff [!]

## Verse 3

Building to train-like repetition on "track" in all voices  
T. → A. = words from text (sung?) → or sprechg. [?]: sprechg. (1stave) - sprechg. (5) - sung  
B. → F. = always repetitive pattern  
S. → S. [choo] → tʃu: + melisma on "sparks"  
A. → B. [choo] → tʃu: + melisma on "sparks"

### To Do:

- ✓ S. + A. harmonies → mm. 88-89
- ✓ S. gesture → mm. 90
- ✓ T. + [S./A.] in/exhale + "R"s
- ✓ B. split → m. 91 (-2 [?])
- ✓ S. + B. harmonies → "

rhythm "track" repetition order = B. → A. → S. → T.

priority order = T. solo, S. A. melisma, B. + [S. A. T.] repetitions in [4/4]

at end: [5/4] → Illumination etc...

4.1 → "Illumination" → alternate between 6/4 [1x2] & 5/4 + half time → 6/8 [1x2] & 5/8  
[cell C1] → on natural syllables from the word!

S. → I  
l  
u: ← A. (link = [l] - [k] - [I] - [i:] - [z] - [3] - [ʃ] - [s] - [t] → [l] etc...  
B. → m  
→ i: [originally I]  
→ n  
→ eI [also e:]  
→ ə [differs from ə:]  
→ n [also ɔ, ɒ]

(link vowels using [v] + [o] + [o:])  
initially 10 beats → 2 x [5/8] + [1. 2]  
slower pulse on mutations = [3/2] [?] or [5/4] [?]

4th time → 5/8 [i:] [n] 4/8 [i:] [n] 5/8 [i:] [n] 4/8 [i:] [n]  
3rd time → 5/8 [i:] [n] 4/8 [i:] [n] 5/8 [i:] [n] 4/8 [i:] [n]  
[melisma] [cell A] B2 [A] [C2] [cell A]  
1st time → (nat.) [S. T. A. B.] or 2 [?] (S = T)

Verse 4: Initial Rhythm → 5/8 Il-lu-mi-na-tion 4/8 Il-lu-mi-na-tion

\* change metric structure canon (sprechg.) homophonic  
[9/8] + [2/4] x 2 (S = T) 2nd time → 5 [A + B2] 4 [A + C2] 5 [no 4th bar] or [i:] or [eI]  
then [9/8] + [5/8] (2S + 3S) 8 [or gliss.] 8 etc... 8-na-tion  
[B. A. T. S.] [nat x1] Middle Section (Vowels) → [I] → [U:] → [e:] → [i:]

Verse 3 [part 2]  
[climax] b.1-2-3-4 b.1-2-3-4  
part 1 climax:  
S. 3J 3J<sup>3</sup> 5J<sup>5</sup> 3J<sup>6</sup>  
A. 5J 1<sup>2</sup> 3J<sup>3</sup> 5J<sup>5</sup>  
T. J 5J<sup>2.5</sup> 5J<sup>2.5</sup> 3J<sup>3</sup> 5J<sup>5</sup>  
B. 3d 3d<sup>1.5</sup> 3d<sup>1.5</sup> 3J<sup>3</sup>

1st time → [cell A] [I] + [U:] + [eI] + [ʃ] + [n]  
2nd time → [cell B1] [I] → [U:] → [e:] → [i:] - [ʃn]  
3rd time → [cell C1] [i:] → [U:] - [nIn] → [e:] → [i:] - [ʃan]  
homophonic + (nat.) + vowel mutations  
homophonic + (sprechg.) + gliss.  
or [cell A] in S. + B & [cell B3/C3] tritones in A + T.



Amber on Black Notes (5) "Colourful, Delicate"  
 \* can also mutate [i.e. (əv-a) → (əv-ɪ)]

4.2] "Fast, or slow -" 1. /Fast/- a - (as-ta) % \_\_\_\_\_ } also [p] + [I]  
 2. fest - ε - (εs-tε) % \_\_\_\_\_ }  
 beat rhythm: 6f 3. /Slow/- a - (əv-a) % \_\_\_\_\_ → also [I]  
 4. slav - ε - (av-ε) % \_\_\_\_\_ → also [p, ?]  
 Fast a (as-ta) % \_\_\_\_\_ etc... or 16 x regular f [64<sup>th</sup> - notes] or 20 x quintuplet f [64<sup>th</sup> - notes]

Berio "money, penny..." [half-time] 3f 3f 3f  
 ↳ or 10:8f  
 [4], [3] or [2] [?]

Feathered Beams: Use J=60 for 'senza tempo' sections [i.e. 1 x J beat = 1 sec.]

4 J beats → [2-4-6-8] → easier to feel each new beat!  
 8 J beats → [2-3-4-5-6-7-8-9] → [2-2-4-4-6-6-8-8]

Poem #2] \* [TEMPO]  
 1a] "Damp mist" → nasal/breathy, high pitched + falsetto } [slow/scattered, melancholy]  
 1b] "@ midnight" → nasal/breathy, deep/low-pitched }  
 2a] "procession" [i.e. 'orderly group'] → emerging contrapuntal elements } [slow/contrapuntal]  
 2b] "streetlights" → long note-values, harmonic counterpoint } [colouristic [?]  
 abs/oss to represent light  
 3a] "bowed" } slow/harmonic  
 3b] "haloes" } choir-like  
 → + ord. [!]

4.] "Illumination" → Transition Section ([4] x 2 + [5] x 3) → mf < → pp < → mp < → f <  
 1st time → S. [I]-[ε]-[a]-[ε] + [I]-[y]-[e]-[i] + [e]-[y] > | breathy almost whispered  
 ↳ mf < A. [l]-[λ]-[I] + [i]-[z]-[3] \* [j] + [s] - [t] > | \* = unvoiced syllable  
 + mp T. [u]-[v] + [p]-[p]-[a]-[λ] + [a]-[u] > | 3rd time → [3] + [5]  
 B. [m] + [n]-[n]-[p]-[j] + [p]-[λ] > | Bass Line: [sprechg.]  
 2nd time → S. [I]-[ε]-[a]-[a]-[i]-[y]-[e]-[a] + [a]-[u] > | 3 J 4 J 5 J 8 J  
 A. \* [t]-[l]-[u]-[i] + [I] END! J=BASS ↳ END! > | END = J d or J d  
 T. END! u:=ALTO | a:=SOP > | pppp  
 B. [g]-[p]-[n]-[m] + [n]-[λ]-[3] \* [j] + [z]-[λ] > |



"Amber on Black" Notes (6)  $\rightarrow [1.2] S. = F_1, T. F[\epsilon]_1, A.B. = F_0$  on 2<sup>nd</sup> beat [?]

"Fast, or slow"  $\rightarrow (\text{beat} = 63)$  "Fast" all  $F_1 \rightarrow 3 \times \text{beat} + \dots$  in  $([2/8] \times 2) \rightarrow [\text{sprechg.}]$

Tutti Section (Structure) 1.3] OR<sub>0</sub> sprechg. for 1x beat in all parts  
1.4] SL<sub>1</sub> in S.T.B. [gliss.], So in A. [sprechg.] +  $\dots$  }  $[3] + [5]$   
 }  $[8] + [16]$

short transition 2.1] - 2.4] as above with ORL<sub>1</sub> on "or"  
into the final 3.1]  $\text{beat} \rightarrow 1 \quad 2 \quad 3 \quad 4 \quad 5$   
line at the end  $[3] + [2] \quad 10\text{f} + 6\text{f} + 10\text{f} + 6\text{f} + 10\text{f} \rightarrow [A.+T.] \quad \leftarrow \text{FL}_1 \rightarrow \quad [2] + [5]$   
 $[8] + [8] \quad 10\text{f} + 6\text{f} + 10\text{f} + 6\text{f} + 10\text{f} \rightarrow [S.+B.] \rightarrow \text{"Fast"} [ORL_1] \leftarrow [8] + [16]$

Use  $(\text{beat} = 63)$  or  $\text{semibre}$  4.1]  $\text{beat} \rightarrow 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9$  "slow"  
in final [?] if  $\text{beat} = 63$   $[3] \rightarrow 10\text{f} + 8\text{f} + 6\text{f} + 4\text{f} + 10\text{f} + 8\text{f} + 6\text{f} + 4\text{f} + 10\text{f} \rightarrow [S.]$   
section  $[?] \quad \text{if } \text{beat} = 126 \text{ or } \text{beat} = 42$   $[8] \rightarrow 10\text{f} + 8\text{f} + 6\text{f} + 4\text{f} + 10\text{f} + 8\text{f} + 6\text{f} + 4\text{f} + 10\text{f} \rightarrow [A.]$   
compound time!  $\text{"fast"} \rightarrow [SL_2] \rightarrow T. \quad \text{"slow"} \rightarrow [SL_1] \rightarrow B. \quad \text{"fast"} \rightarrow [SL_2] \rightarrow B. \quad \text{"slow"} \rightarrow [SL_1] \rightarrow B.$

5.1]  $\text{beat} \rightarrow 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9$  "fast"  
"slow"  $[3] \rightarrow 10\text{f} + 10\text{f} + 8\text{f} + 6\text{f} + 4\text{f} + 10\text{f} + 8\text{f} + 6\text{f} + 4\text{f} \rightarrow [T.]$   
"slow"  $[8] \rightarrow 10\text{f} + 10\text{f} + 8\text{f} + 6\text{f} + 4\text{f} + 10\text{f} + 8\text{f} + 6\text{f} + 4\text{f} \rightarrow [B.]$   
"slow"  $\text{"fast"} \rightarrow [ORL_1] \rightarrow [S.] \quad \text{"slow"} \rightarrow [ORL_2] \rightarrow [A.]$

6.1]  $\text{beat} \rightarrow 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9$  "slow"  
"slow"  $([3] \times 2) + [7] \rightarrow 10\text{f} + 4\text{f} + 6\text{f} + 8\text{f} + 10\text{f} + 8\text{f} + 6\text{f} + 4\text{f} + 10\text{f} \rightarrow [S.] @ f$   
"slow"  $[8] \rightarrow 10\text{f} + 4\text{f} + 6\text{f} + 8\text{f} + 10\text{f} + 8\text{f} + 6\text{f} + 4\text{f} + 10\text{f} \rightarrow [A.]$   
"slow"  $\text{"fast"} \rightarrow [FL[\epsilon, d]_2] \rightarrow A. \quad \text{"slow"} \rightarrow [SL[a:2]] \rightarrow T. (\text{no } f!) \quad \text{"slow"} \rightarrow [FL_2] \rightarrow B.$

7.1] S.  $\rightarrow$  Fast a (as-ta)  $\rightarrow (\epsilon s - t\epsilon)$   
A.  $\rightarrow$  slow I (av-I)  $\rightarrow (av - p)$  [swap?...]  
T.  $\rightarrow$  first I (is-ti)  $\rightarrow (ps - tp)$   
B.  $\rightarrow$  slow a (av-a)  $\rightarrow (av - \epsilon)$

7.2]  $[3 \text{ or } 6] \text{f} \quad \text{how was it for you?}$   
 $[8 \text{ or } 16] \text{f} \quad \text{how was it for you?}$

Tutti Section  
Basic Order

- [C] set 1] Fast - or - slow - normal / obvious order
- [CR1] set 2] Fast - or - slow - combination on "Fast", extensions on "or" + "slow"
- [CR2] set 3] all parts accelerate / decelerate with peak on "fast" syllable combination  $\rightarrow 4 \times \text{"Fast" for } x3]$
- [mf] set 4] two parts "fast" + "slow" combination  $[S.+T.]$   
 $\rightarrow 8$  syllable "slow" mutation in the other two parts simultaneously
- END! set 7] As in first set: Fast - or - [slow]  $\rightarrow$  accel / decel in two parts + bring in final line  $[A.+B.]$
- set 5] 5 syllable "or" in two parts + short full speed or accel / decel fragments in other two  $[A.+B.]$
- set 6] one part accels / decels + 8 syllable mutation in other three  $[1 \times \text{"slow"} + 2 \times \text{"fast"}]$
- set 8] no generic tutti section material, but include "how was it fast/slow" + "or" + "fast/slow for you?"

$\rightarrow ([=Acell]) @ ff$ , before sudden change to pp dynamic

continuity between sets  $[4-5-6]$  mutate!  $[3 \times 3/8 [?]] \rightarrow$  semiquaver divisions on glissandi lines!

$[3 \times 3/8 [?]]$  mutate!  $\leftarrow 9\text{f} - 6\text{f} - 9\text{f} \rightarrow 1 \times \text{offset in glissandi lines}$

set 3 = 5f  $\leftarrow$  [accel / decel.]  
length

also mutate in set 7 5f beats (i.e. mutate for  $[2\text{f} + 1\text{f}]$  beats  $\times 2$ )

# "Amber on Black" Sketch (1)

buf - fet - ed [G#, F, G] [A, B, C] [Bb, Ab] [G, F]

Sprechg. 3 5 6

S A T B

Word buf-fet-ed word. Word Ar-tic-u-la-tion

11-note row sprechg.

6 gliss. m. 24<sup>3</sup> sfz sfz 3

de-railed de-railed di r erld

Ar-tic-u-la-tion mf pccofte.

4 4 4 4

gliss. f 6

di R erld erld Ar-tic-u-la-tion de-railed de-railed

flz. f 6 6 6

sprechg.

mm. 65-70 [ALTO 12-note row] mm. 83- S. m. 91 → [Eb, Bb, C]

ppp

m. 91 [T.] [C, Bb, Eb]



"Amber on Black" Sketch (2)

mf mp mf mp

$$\left[ \begin{smallmatrix} 4 \\ 8 \end{smallmatrix} \right] [?] \rightarrow \left[ \begin{smallmatrix} 2 \\ 8 \end{smallmatrix} \right] \times 2 [?]$$

116

S.  $\text{mf} \text{ mp } \text{mf} \text{ mp}$   
i → y → e: → i: → e: → y

A.  $\text{mp} \text{ mf } \text{mp}$   
i: → z → 3 → f → s → l

T.  $\text{mf}$   
o: → ɔ: → a: → ʌ → ɔ: → u:

B.  $\text{mp} \text{ mf } \text{mp}$   
m → n → j → ŋ → j → n

pp pp  
i → ε → a → a:

ppp [unvoiced]  
f, whisper

(natural speaking pitch)

"fast or slow"  
set 8  
(transition)

$$\left[ \begin{smallmatrix} 3 \\ 8 \end{smallmatrix} \right] =$$

S.  $\text{F}_1$  for 2  $\text{f}$  etc...  
for you?

A.  $\text{P}$  ORL<sub>2</sub> for 5  $\text{f}$  etc...  
how... [speech]

T.  $\text{f}$  etc...  
was it for you?  
etc... how...

B.  $\text{S}_6$  for 4  $\text{f}$  etc...  
slow [a] for you?  
how...

$$\left[ \begin{smallmatrix} 6 \\ 16 \end{smallmatrix} \right]$$
  
 $[f = f = c.126, l = c.42]$

119

S.  $\text{pp} \text{ ppp } \text{pp} \text{ ppp } \text{pp}$   
i: → y → ei → ai → ɔ: → u:

A.  $\text{pppp}$   
(murmur)

T.  $\text{SF}_4 \rightarrow \text{F}_1 \rightarrow \text{FS}_3$   
or

B.  $\text{ppp} \text{ pp } \text{ppp} \text{ pp } \text{ppp} \text{ pp}$   
m → n → 3 → f → z → n

ppp → breathy, almost whispered  
→  $\text{F}_2 \rightarrow [\epsilon]$   $\text{SF}_3 \rightarrow \text{F}_1 \rightarrow \text{FS}_3$

ORL<sub>1</sub>  
poco flz.  
[y] or [ɔ:] [r] slow

mp  
ll - lu - mi -

$\text{D}_4 4, \text{C}_5$   
↑  
12-tone pitch -  
(unused classes in m.120)  
↓  
 $\text{F}_3 \#3, \text{G}_3 \#3$



swap figures in S. + A. [?]

122

S.  $p$   $[ε]$   $F_2$   $S_6$  in swap [?]  $\rightarrow$  speech [unvoiced]  $\rightarrow$  (natural speaking pitch)  $\rightarrow$  New Tempo Marking  $\rightarrow$  Colourful, Radiant ( $f = c.63$ )

A.  $pp$   $S_1 \rightarrow S_5$  etc...  $f$  [Energetic] [?]

T.  $unvoiced!$   $[s] \rightarrow [l] \rightarrow [ə] \rightarrow [u:]$  gliss. join [?] etc...  $f$

B.  $na - tion$  etc...  $f$   $[4/8]$  [?]

TEN. END  $\rightarrow$  whisper (unvoiced)

ALTO END  $\rightarrow$  breathy, almost whispered (voiced)

SOP. END  $\rightarrow$  speech (unvoiced)

BASS END  $\rightarrow$  singing voice (ord., voiced) and/or  $[t]$  with  $[d]$

\*  $[f] - [a:] - * [s] \times 2 - * [t]$   $\rightarrow$  (possibly replace  $[a:]$  with  $[ε]$ )  $\rightarrow$  4 syllable expansions  $\uparrow$   $ff$  [!]

\*  $[s] - [l] - [ə] - [u:]$  ("  $[ə]$  with  $[a:]$  )  $\rightarrow$  (or  $[z]$  with  $[j]$ ) always use gliss. [?] or sprechg. [?]

\*  $[f] - [v] - [u:] - [a:] - [i:] - [z] - * [s] \times 2 - * [t]$  "Fast"  $\rightarrow$  8 syllable expansions

\*  $[z] - * [s] - * [t] - [l] - [y] - [ə] - [o:] - [u:]$  "Slow"  $\rightarrow$  8 syllable expansions

Possible Fragments

"Fast"  $\rightarrow [fa:st] = F_0$  single syllables  $\rightarrow$  [pitched with or without glissando]

"or"  $\rightarrow [ɔ:] = OR_0$  @ a natural + @ slowed-down speed  $\rightarrow$  [up or down]

"slow"  $\rightarrow [sləv] = S_0$  speaking speed with syllable expansions/mutations

"Fast" full speed over 2/4 bars  $= F_1/F_2$

"Fast" accelerating over 2/4 bars  $= F_3/F_4$

"Fast" decelerating over 2/4 bars  $= F_5/F_6$

"Slow" full speed "  $= S_1/S_2$

" " accelerating "  $= S_3/S_4$

" " decelerating "  $= S_5/S_6$

"Slow"  $\rightarrow$  "Fast" accelerating "  $= S-F_3/S-F_4$

"Fast"  $\rightarrow$  "Slow" decelerating "  $= F-S_5/F-S_6$

"Fast" + "Slow" [combined] full speed  $= COM_1/COM_2$

" " accelerating  $= COM_3/COM_4$

" " decelerating  $= COM_5/COM_6$

Berio repeated figures with short time-values, mutations etc...

$[y] - [ɔ:] - [r]$  (pitched rolled 'r')  $= [ORL_1]$

["or"] (replace  $[y]$  with  $[i:]$  or  $[r]$  with  $[R]$ )

$([i:] - [ə])$  ["or"]

$[y] - [e:] - [ɔ:] - [v] - [r]$   $= [ORL_2]$

$([dʒ] - [R])$

also  $F[ε]_1 / F[ε]_2$  etc... to represent phonetic mutations etc...

["or"]  $\rightarrow$  3 or 5 syllable phonetic expansions, always constant pitch [?]



# APPENDIX 6

A User Guide for  
*Eternal Owl Call's*

Electronic Performance Interface

# Contents

Page 138	<b>Adding the Z_Library to SuperCollider's Core Class Library</b>
Page 139	<b>Running <i>Eternal Owl Call's</i> Performance Interface</b>
Page 140	<b>A Screenshot of the GUI</b>
Pages 141–53	<b>Using the GUI</b>
Page 141–3	<u>The Left-Hand Columns (Patches &amp; Subpatches)</u>
Page 141–2	The “Next Subpatch” Button
Page 142	The “Start” and “Free” Buttons
Pages 142–3	The “Info...” Buttons
Page 143–5	<u>The Central Columns (Top Area)</u>
Page 143	The “Free All Synths” Button
...	The “Exit GUI” Button
...	The “Reboot Program” Button
Page 144	The Mode of Operation
...	Testing Mode
...	Performance Mode
Page 145	Record Mode
...	The Last Recorded Session
...	<u>The Audio Sample Auditioning Panel</u>
Page 146	<u>The Doppler Effect Auditioning Panel</u>
Page 146–7	<u>The Microphone Bus Routing Panel</u>
Pages 147–8	<u>The Modulation Routine Settings Panel</u>
Page 147	The “Further Info...” Button
Page 148	The “Set” and “Free” Buttons
...	<u>The Top-Right Corner</u>
...	The Full Screen Toggle Button
...	The Help Button
Pages 148–50	<u>The Post Window</u>
Page 149	The “Clear Post Window” Button
...	The “Patch # 2 Debug Data” Button
Page 150	The “Memory Size” Button
...	The “Number of Audio Bus Channels” Button

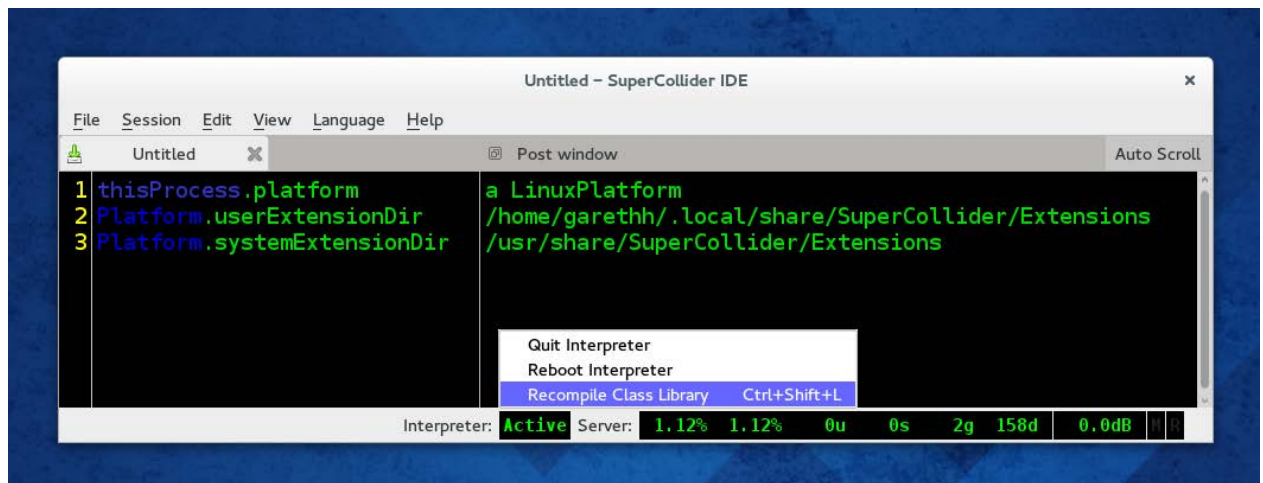
<i>Pages 150–1</i>	<u>The Master, Scope &amp; Analyzer Panels</u>
<i>Page 151</i>	The Master Panel
...	The Oscilloscope Panel
...	The Frequency Analyzer Panel
...	<u>The Server Levels Panel</u>
<i>Page 152</i>	<u>Closing and Destroying the GUI Window</u>
...	<u>Killing the Program Using CmdPeriod</u>
<i>Pages 152–3</i>	<u>The User Directory</u>
<i>Page 154</i>	<b>Shortcut Keys...</b>
<i>Pages 155–8</i>	<b>Example Run Script</b>

# Adding the Z Library to SuperCollider's Core Class Library

The files required to run the performance interface for *Eternal Owl Call* are included in the **Z\_Library** – a library of class definitions that I have coded and added as extension's to SuperCollider's core class library. To compile the **Z\_Library**, you must copy the 'Z\_Library.d' directory (and all of its subdirectories) from the data CD, included with this PhD thesis, to one of the two SuperCollider class extension directories; returned by evaluating the two following lines of code in SuperCollider IDE:

```
Platform.userExtensionDir    // user extension directory
Platform.systemExtensionDir // system extension directory
```

Once the **Z\_Library** has been successfully copied to one of these directories, recompile SuperCollider's class library by selecting the relevant option from the interpreter's pop-up menu at the bottom of SuperCollider IDE as shown below (shortcut key: **Ctrl+Shift+L** [on Linux or Windows] or **Cmd+Shift+L** [on Mac OSX]):



## Running *Eternal Owl Call's* Performance Interface

To run *Eternal Owl Call's* performance interface and graphical user interface (GUI), the following line of code should be evaluated in SuperCollider IDE:

```
z = CZ_EternalOwlCall() // this boots the program and opens the GUI

/*
  if you close and destroy the GUI window while audio synths are still
  running you can reopen the GUI window with one of the two following
  commands...
*/
z.reopenGUI
CZ_EternalOwlCall.class_zpi_pointer.reopenGUI
```

To evaluate a line of code in SuperCollider IDE, simply move the cursor to the relevant line and press either **Ctrl+Enter** or **Shift+Enter** (on Linux or Windows), or either **Cmd+Enter** or **Shift+Enter** (on Mac OSX).

Don't worry too much if you don't fully understand some of the computer programming jargon that occurs in these lines of code; once you have run the program the GUI is actually very user friendly and easy to use!

# Eternal Owl Call : Real-Time Performance Interface

Next Subpatch (SPACEBAR)

Next Subpatch = 1.01

Patch # 1 ...

Info?...

1.01	Start	Free
1.02	Start	Free
1.03	Start	Free
1.04	Start	Free
1.05	Start	Free

Patch # 2 ...

Info?...

2.01	Start	Free
2.02	Start	Free
2.03	Start	Free
2.04	Start	Free
2.05	Start	Free
2.06	Start	Free
2.07	Start	Free
2.08	Start	Free
2.09	Start	Free
2.10	Start	Free
2.11	Start	Free

Patch # 3 ...

Info?...

3.01	Start	Free
3.02	Start	Free
3.03	Start	Free
3.04	Start	Free
3.05	Start	Free
3.06	Start	Free

Free All Synths

Exit GUI

Reboot Program

Mode of Operation =

TESTING

Record Mode =

On

Last Recorded Session:

Play

Stop

Audio Sample Auditioning ...

Tawny Owl Sample:

Play

Stop

Pigmy Owl Sample:

Play

Stop

Otter Sample:

Play

Stop

Frog (Amazon) Sample:

Play

Stop

Frog (Peru) Sample:

Play

Stop

Mosquito Sample:

Play

Stop

Doppler Effect Auditioning ...

Otter Sample Doppler:

Play

Stop

Mosquito Sample Doppler:

Play

Stop

Otter Formant Doppler:

Play

Stop

Mosquito Formant Doppler:

Play

Stop

Microphone Bus Routing ...

Direct Microphone Signal Only:

Off

Microphone to Reverb Channel:

On

Mic to Reverb + 7 Tap Delay:

Off

Mic to Reverb + 12 Tap Delay:

Off

Reverb + 12 Tap Delay + Comb Filter:

Off

Modulation Routine Settings ...

Further Info?...

Frequency Modulation:

Set

Free

Pan Modulation:

Set

Free

EXIT Full Screen (F11)

Help (F1)

Clear Post Window

Post Window & Debugging ...

Server Sample Rate = 48000Hz  
Server Process ID = 2869

Directory path for reading/writing =  
/home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/Classes.d/Compositions.d/Blodyn-Tylluan.d/Nr3\_Eternal\_Owl\_Call.d/CZ\_EternalOwlCall.d

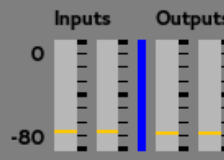
Record mode enabled!...  
Initializing effects processors...  
Program initialized!  
Post Window data will be displayed here...

Patch # 2 Debug Data

Memory Size

Number of Audio Bus Channels

MASTER SETTINGS



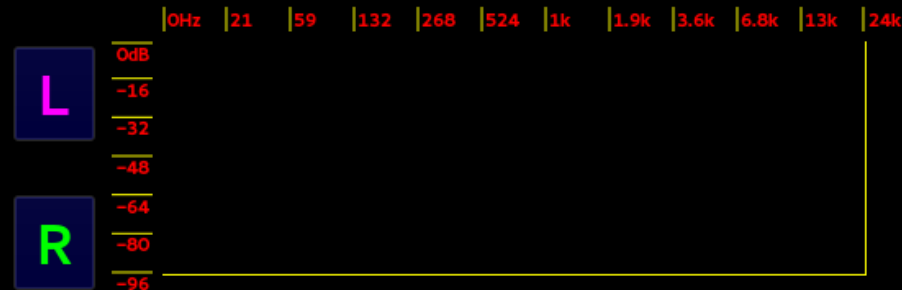
Oscilloscope ...

SCOPE SETTINGS

L  
R

Frequency Analyzer ...

ANALYZER SETTINGS



17:09:26

Server :

3.12%

3.10%

95u

12s

5g

160d

Interface Copyright (c) : Mr Gareth Olubunmi Hughes (1 July 2014 - Present)



## Using the GUI

The figure on the previous page shows *Eternal Owl Call*'s main GUI window in full screen mode (the default loading view for this window). There are three discernable control areas on the GUI's main window as follows:

1. The left-hand columns, which have buttons allocated for starting and freeing each subpatch required for the performance of *Eternal Owl Call*.
2. The central columns, which have buttons allocated for performing general system operations, as well as testing some of the audio samples and effects processors used by the program.
3. The right-hand columns, which have analysis panels in place for displaying information messages posted by the program, musical amplitude/frequency levels and the power consumption levels of SuperCollider's server.

This easy-to-use GUI is designed for the purposes of performing and rehearsing *Eternal Owl Call*'s electronics section (as indicated in the score, vol. 1, pp. 294–7), as well as testing some of the sounds and effects that have been used with the piece.

*Eternal Owl Call*'s electronics section is made up of three separate patches, which are each divided into a number of smaller subpatches (as outlined in vol. 1, pp. 298–300). The rising numerical values on the GUI's leftmost column (i.e. 1.01, 1.02, 1.03 etc...) correspond to the equivalent numerical values which appear on the electronics part in the score. Whenever such an indication occurs in the score, the corresponding subpatch should be started on the GUI.

### The Left-Hand Columns (Patches & Subpatches)

The left-hand columns have buttons allocated for starting and freeing each subpatch, as shown in the figure below, which shows the top-left area of the main window (descending to the end of **Patch # 1**):



### **The “Next Subpatch” Button**

The “Next Subpatch” button (shortcut key: **Spacebar**) starts each subpatch in sequential order. In theory, during a live performance of *Eternal Owl Call*, the electronics performer should only need to press this button, or the spacebar, to perform the entire piece from start to finish.

The display panel underneath this button indicates which subpatch is next in the sequence. This value is determined by referencing the last subpatch to have been started or freed by the program. When the final subpatch is reached the panel will display **‘Next Subpatch = END’**.

In addition, a useful shortcut key to free the last subpatch performed is **Ctrl+Backspace**.

## The “Start” and “Free” Buttons

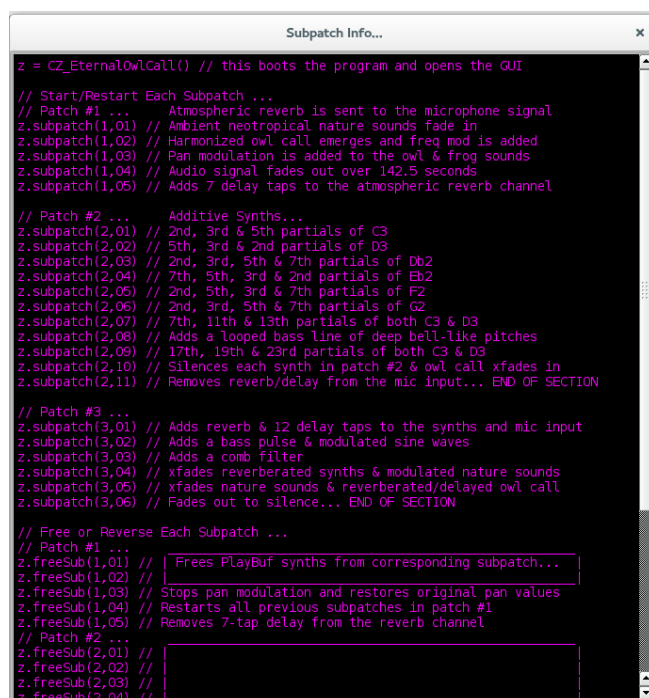
These buttons start or free each subpatch in the program. The “Start” button will create audio synths used by that by the corresponding subpatch or change values on existing synths. Once a subpatch has been started, the button’s background colour turns black and its label changes to “Restart”. If you press it again (i.e. ‘restart’ the subpatch) any synths already created by that subpatch will be silenced, freed and overwritten to avoid any loss of computer memory.

The “Free” buttons will silence and free any synths created by the corresponding subpatch and restore any changed synth values to their previous settings. The “Free” button will also restore any other settings which have been changed by the corresponding subpatch (e.g. if the subpatch triggers a crossfade the source audio signal is restored; or if the microphone bus effects setting is changed, the previous setting is restored).

In theory, you should not need to “Restart” or “Free” any subpatches during a live performance; however, these buttons are very useful during rehearsals and might be required should anything go wrong during a live performance.

## The “Info?...” Buttons

The “Info?...” buttons open a separate window (the “Subpatch Info...” window; shortcut key: **Alt+I**), as shown below:



```

Subpatch Info...
z = CZ_EternalOwlCall() // this boots the program and opens the GUI

// Start/Restart Each Subpatch ...
// Patch #1 ... Atmospheric reverb is sent to the microphone signal
z.subpatch(1,01) // Ambient neotropical nature sounds fade in
z.subpatch(1,02) // Harmonized owl call emerges and freq mod is added
z.subpatch(1,03) // Pan modulation is added to the owl & frog sounds
z.subpatch(1,04) // Audio signal fades out over 142.5 seconds
z.subpatch(1,05) // Adds 7 delay taps to the atmospheric reverb channel

// Patch #2 ... Additive Synths...
z.subpatch(2,01) // 2nd, 3rd & 5th partials of C3
z.subpatch(2,02) // 5th, 3rd & 2nd partials of D3
z.subpatch(2,03) // 2nd, 3rd, 5th & 7th partials of Db2
z.subpatch(2,04) // 7th, 5th, 3rd & 2nd partials of Eb2
z.subpatch(2,05) // 2nd, 5th, 3rd & 7th partials of F2
z.subpatch(2,06) // 2nd, 3rd, 5th & 7th partials of G2
z.subpatch(2,07) // 7th, 11th & 13th partials of both C3 & D3
z.subpatch(2,08) // Adds a looped bass line of deep bell-like pitches
z.subpatch(2,09) // 17th, 19th & 23rd partials of both C3 & D3
z.subpatch(2,10) // Silences each synth in patch #2 & owl call xfades in
z.subpatch(2,11) // Removes reverb/delay from the mic input... END OF SECTION

// Patch #3 ...
z.subpatch(3,01) // Adds reverb & 12 delay taps to the synths and mic input
z.subpatch(3,02) // Adds a bass pulse & modulated sine waves
z.subpatch(3,03) // Adds a comb filter
z.subpatch(3,04) // xfades reverberated synths & modulated nature sounds
z.subpatch(3,05) // xfades nature sounds & reverberated/delayed owl call
z.subpatch(3,06) // Fades out to silence... END OF SECTION

// Free or Reverse Each Subpatch ...
// Patch #1 ...
z.freeSub(1,01) // Frees PlayBuf synths from corresponding subpatch...
z.freeSub(1,02) //
z.freeSub(1,03) // Stops pan modulation and restores original pan values
z.freeSub(1,04) // Restarts all previous subpatches in patch #1
z.freeSub(1,05) // Removes 7-tap delay from the reverb channel
// Patch #2 ...
z.freeSub(2,01) //
z.freeSub(2,02) //
z.freeSub(2,03) //
z.freeSub(2,04) //

```

This window provides succinct information that explains which actions are performed by the “Start” and “Free” button for each subpatch. This is a useful guide for the electronics performer and should be consulted as a reference where necessary.

When this window is in focus, pressing **Return** or **Esc** will close it.

## The Central Columns (Top Area)

The top area of the columns in the centre of the GUI contains buttons allocated for performing general system and recording operations as follows:



### The “Free All Synths” Button

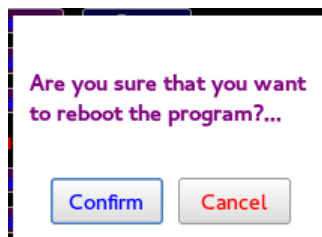
The “Free All Synths” button (shortcut key: **Ctrl+Esc**) will silence and free all active audio synths created by the program, including anything scheduled on the **SystemClock**, **TempoClock** or **AppClock**. It will also reset the next subpatch value to the first subpatch (i.e. ‘**Next Subpatch = 1.01**’).

### The “Exit GUI” Button

The “Exit GUI” button (shortcut key: **Esc**) will minimise the GUI window in order for the user to be able to easily exit the GUI screen and access other programs and windows that are active on the operating system.

### The “Reboot Program” Button

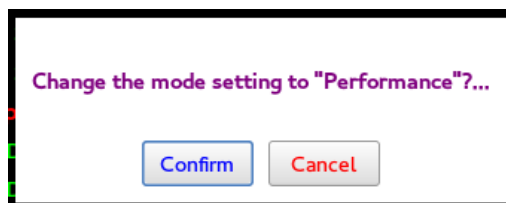
If the “Reboot Program” button is pressed (shortcut key: **Ctrl+B**), you will first be presented with a confirmation dialog window as follows:



If you confirm, all windows created by the program will first be closed and destroyed, SuperCollider’s server will then be rebooted and a fresh GUI window will reopen with all values reset.

## The Mode of Operation

There are two modes of operation: “Testing” and “Performance” (an explanation of each is provided below). If you press on the button to change this setting, you will first be presented with a confirmation dialog window as follows:



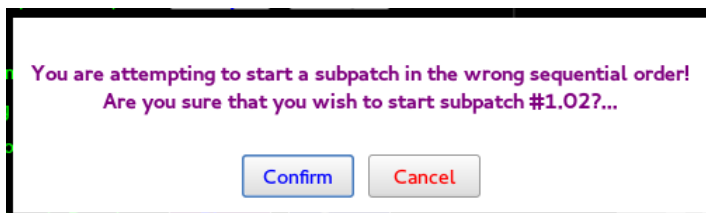
If you confirm, the mode of operation will toggle between both of these settings.

## Testing Mode

Testing mode is the default mode of operation. This mode allows most buttons in program to perform an action immediately after only being pressed once. Testing mode is ideal for testing the program and for use during rehearsals. Many users will also find it to be practical during a live performance; however, performance mode is a slightly safer mode to use during a live performance (as explained below) and the electronics performer should consider which of the two modes of operation they would prefer to use in such circumstances.

## Performance Mode

Some users of the GUI may prefer to set the mode of operation to performance mode during a live performance of *Eternal Owl Call*. Performance mode will allow the user to start each subpatch *in the correct order only* without issuing a warning. However, if the user attempts to press any other button which generates or frees a synth, or changes the effects processor settings (e.g. starting a subpatch in the wrong order, freeing a subpatch, freeing all subpatches, auditioning a sample or effect, changing the effects that the microphone channel is sent to, etc...) the user will first be presented with a confirmation dialog window. As an example of this, here is the message which is displayed if the user attempts to start a subpatch in the wrong order:



## Record Mode

Record mode will allow the user to save the current session to disk as a 48 kHz, 16-bit AIFF file. This will record all audio synths generated on SuperCollider's **RootNode**, including the microphone channel, to a single file. The file will be saved in the "recordings.dir" directory, which is located in the program's **User Directory** (described later on in this user guide).

Record mode can be switched on or off with the blue toggle button in top area of the central columns. Record mode is switched on by default and a new file is automatically saved to disk when record mode is switched off. In addition, a file will automatically be saved to disk if the node for recording the session to disk is freed (e.g. following a call to **CmdPeriod**) or if the SuperCollider server is quit.

## The Last Recorded Session

If a session has been recorded at some point since the program was first run, the "Last Recorded Session" buttons will allow the user to audition the last recorded session from within the program.

The "Play" and "Stop" buttons are self-explanatory; when "Play" is pressed, the button's background colour turns black and its label changes to "Replay". If you press it again (i.e. 'replay' the audio file) the synth already created to play the file will be silenced, freed and overwritten to avoid any loss of computer memory.

## The Audio Sample Auditioning Panel

The "Audio Sample Auditioning" panel, in the columns at the centre of the GUI, contains buttons allocated for auditioning six audio samples as follows:



All six of these samples are pre-recorded sounds of natural wildlife, which are used as source material for some of the program's subpatches.

The "Play" and "Stop" buttons are self-explanatory; when "Play" is pressed, the button's background colour turns black and its label changes to "Replay". If you press it again (i.e. 'replay' the audio file) the synth already created to play the file will be silenced, freed and overwritten to avoid any loss of computer memory.

## The Doppler Effect Auditioning Panel

The “Doppler Effect Auditioning” panel, in the columns at the centre of the GUI, contains buttons allocated for auditioning four examples of the doppler-based effects processing synths, which are used with some of the program’s subpatches as follows:



These doppler effects simulate the sensation of an audio signal (with the corresponding audio sample as its source) circulating around the listener, falling in pitch as it moves behind the listener and rising in pitch as it moves in front of the listener.

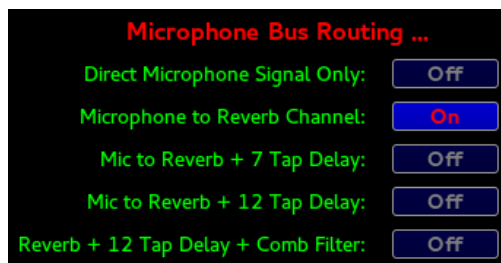
The “Sample Doppler” modifies the sample’s pitch (or ‘frequency’) by changing its sample rate (i.e. changing its playback ‘speed’), as a result, the lower frequencies have a longer playback length than the higher frequencies.

The “Formant Doppler” modifies the sample’s frequency by applying a grain-based formant pitch-shift, as a result, a time-correction is applied to the pitch-shift and all frequencies within a specific range will be of the same length as the original sample.

The “Play” and “Stop” buttons are self-explanatory; when “Play” is pressed, the button’s background colour turns black and its label changes to “Replay”. If you press it again (i.e. ‘replay’ the effect) the synth already created to play the effect will be silenced, freed and overwritten to avoid any loss of computer memory.

## The Microphone Bus Routing Panel

The “Microphone Bus Routing” panel, towards the bottom of the columns at the centre of the GUI, contains buttons allocated for changing the effects processors that the microphone channel is sent to as follows:



This panel will allow the musicians to audition each individual microphone effects setting during a test or a rehearsal.

The panel is set to “Microphone to Reverb Channel” by default, because this is the effects setting that is called for at the start of the score for *Eternal Owl Call*. If this setting is changed during a rehearsal,

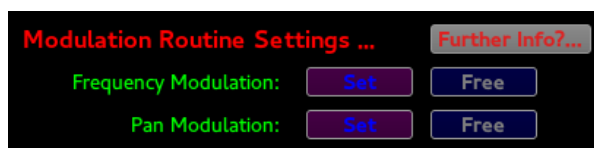
the electronics performer must remember to change it back prior to beginning a performance of the piece, *unless* the musicians have decided in advance that they would prefer a different effects setting at the beginning of the piece (e.g. certain electronics performers might prefer to have a ‘dry’ flute signal at the start of the piece by changing the initial setting to “Direct Microphone Signal Only”).

Several subpatches in the program will automatically change the microphone bus routing setting (to correspond with *Eternal Owl Call*’s score) as follows:

Subpatch #	Automatic Microphone Bus Routing Modification:
1.05	Changes to “Mic to Reverb + 7 Tap Delay”
2.11	Changes to “Direct Microphone Signal Only”
3.01	Changes to “Reverb + 12 Tap Delay + Comb Filter”, with the comb filter’s amplitude value set to zero (i.e. ‘silent’)
3.03	Triggers the comb filter
3.04	Fades the comb filter to silence over 60 seconds

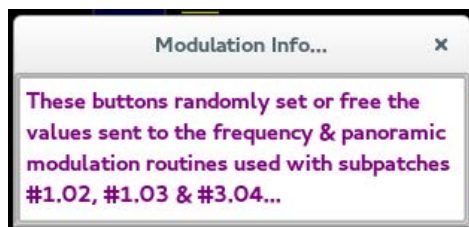
## The Modulation Routine Settings Panel

The “Modulation Routine Settings” panel, at the bottom of the columns in the centre of the GUI, contains buttons allocated for changing the modulation routine settings used with subpatches **#1.02**, **#1.03** & **#3.04** as follows:



### The “Further Info?...” Button

The “Further Info?...” button opens a separate window (the “Modulation Info...” window), which provides further information about these buttons as shown below:



When this window is in focus, pressing **Return** or **Esc** will close it.

## The “Set” and “Free” Buttons

These buttons set or free the modulation values that are sent to the corresponding subpatches. The “Set” button will randomly generate values sent to the frequency or panoramic modulation routines used.

When “Set” is pressed, the button’s background colour turns black and its label changes to “Reset”. If you press it again (i.e. ‘reset’ the modulation values) new modulation values are randomly generated and resent.

The “Free” buttons will set the frequency or panoramic modulation values to zero – effectively removing any modulation.

## The Top-Right Corner

The top-right corner of the GUI contains the following two utility buttons:



## The Full Screen Toggle Button

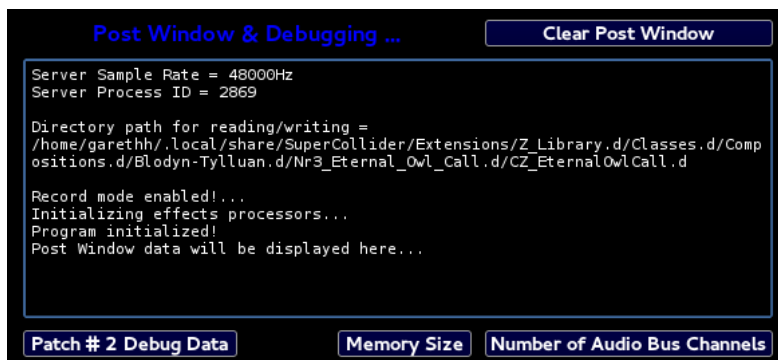
This exits/enters full screen mode (shortcut key: **F11**).

## The Help Button

This opens the help file (containing this user guide in PDF format; shortcut key: **F1**) in a separate window.

## The Post Window

The post window displays user information and debug messages which have been sent by the program (e.g. when a button is pressed and an action performed etc...). The initial boot messages sent to the post window are shown below:





The post window is modelled on the post window in SuperCollider IDE and the messages sent to the GUI's post window will also show up on SuperCollider IDE's own post window. The user should however bear in mind that general system-based exception, error or warning messages will not show up on the GUI's post window and if a problem of this nature is being tested for then the user should monitor SuperCollider's own IDE in the usual way.

If the data shown in the post window exceeds the boundaries of its visual area a scrollbar will occur on the right-hand side, allowing the user to navigate vertically.

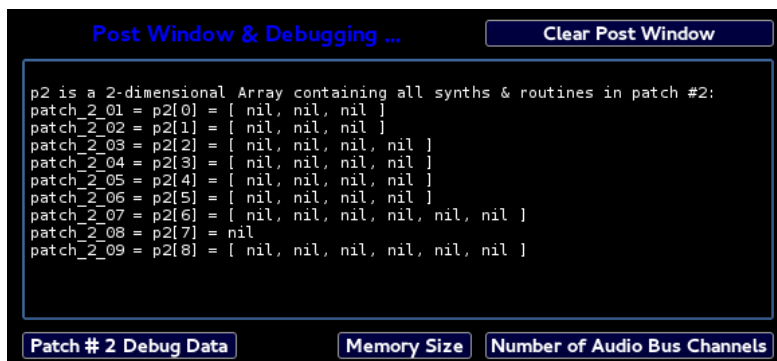
## The “Clear Post Window” Button

The “Clear Post Window” button allows the user to clear any data which is visible in the post window (shortcut key: **Ctrl+Shift+P**).

## The “Patch # 2 Debug Data” Button

The “Patch # 2 Debug Data” button will display data for helping with debugging the second patch, which creates arrays of bell-like additive synths based on frequencies of the natural harmonic series, as well as other bell-modelling synths.

The default debug data displayed by this button (i.e. prior to the creation of any synths) is displayed below:



Subpatches **#2.01–07** and **#2.09** each contain either 3, 4 or 6 arrays of additive synths. The single value attributed to subpatch **#2.08** refers to a **Routine**, which loops a bass line of deep bell-modelled synthesizer frequencies, created using the **Klank** class.

This debug button was useful during the software development process and will also be useful should the second patch be further developed further.

## The “Memory Size” Button

The “Memory Size” button displays the number of kilobytes of real-time memory allocated to the server. This memory is used to allocate synths and any memory that unit generators themselves allocate (for instance in the case of delay ugens which do not use buffers, such as **CombN**), and is separate from the memory used for buffers. Setting this too low is a common cause of ‘exception in real time: alloc failed’ errors (SuperCollider’s default value is 8,192 kB).<sup>1</sup>

The default value can be modified from within SuperCollider IDE. As an example, evaluating the follow line of code will double the default value:

```
s.options.memSize = 16384 // where s = the default server
```

## The “Number of Audio Bus Channels” Button

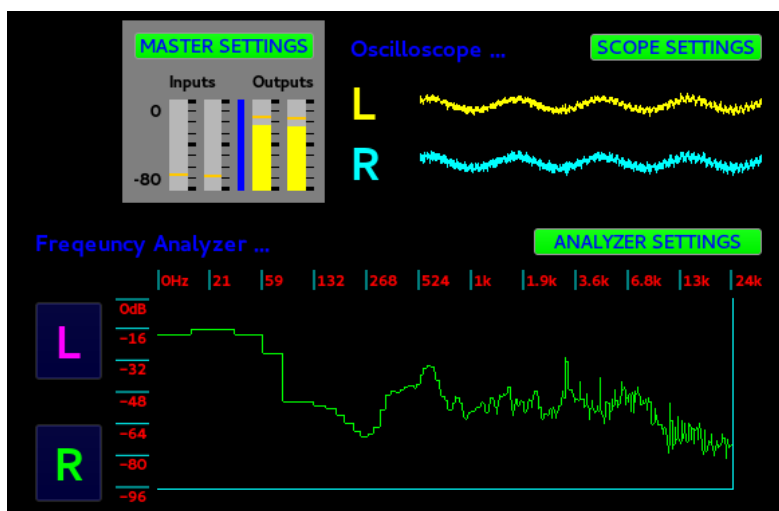
The “Number of Audio Bus Channels” button displays the number of audio rate busses, which includes input and output busses (SuperCollider’s default value is 128).<sup>2</sup>

The default value can be modified from within SuperCollider IDE. As an example, evaluating the follow line of code will double the default value:

```
s.options.numAudioBusChannels = 256 // where s = the default server
```

## The Master, Scope & Analyzer Panels

Directly underneath the post window there are a number of panels for monitoring the musical amplitude and frequency levels that are sent to the input and output channels as follows:



<sup>1</sup> As defined in the documentation for the ‘memSize’ instance method in SuperCollider 3.7alpha’s **ServerOptions** class.

<sup>2</sup> Ibid., for the ‘numAudioBusChannels’ instance method.

## The Master Panel

The master panel displays SuperCollider's input/output meter levels. The "MASTER SETTINGS" button has not yet been implemented. Once implemented, it will open an additional window that will allow the user to modify dynamics processing levels on the master channels.

## The Oscilloscope Panel

The oscilloscope panel displays an amplitude scope for both the left and right output channels. The "SCOPE SETTINGS" button has not yet been implemented. Once implemented, it will open an additional window that will allow the user to modify visual analysis parameters on the oscilloscope.

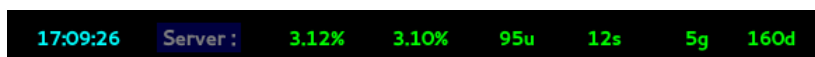
## The Frequency Analyzer

The frequency analyzer panel displays an amplitude against frequency scope, ranging from -96 to 0 dB and 0–24,000 Hz respectively. The "L" and "R" buttons will switch between the analysis data on the left and right-hand sides separately.

The "ANALYZER SETTINGS" button has not yet been implemented. Once implemented, it will open an additional window that will allow the user to modify visual analysis parameters on the frequency analyzer.

## The Server Levels Panel

The server levels panel at the bottom right-hand side of the GUI displays data relating to SuperCollider's server levels as follows:



The leftmost value displays the current time in hour:minutes:seconds format. The server values display the following data (respectively from left to right):

- Peak CPU usage
- Average CPU usage
- Number of running **UGens**
- Number of running **Synths**
- Number of **Groups**
- Number of loaded **SynthDefinitions**

## Closing and Destroying the GUI Window

The shortcut key **Ctrl+Q** will open a confirmation dialog window that allows the user to permanently close and destroy the main GUI window. Confirming this action will also close and destroy any other GUI windows that have been created by the program.

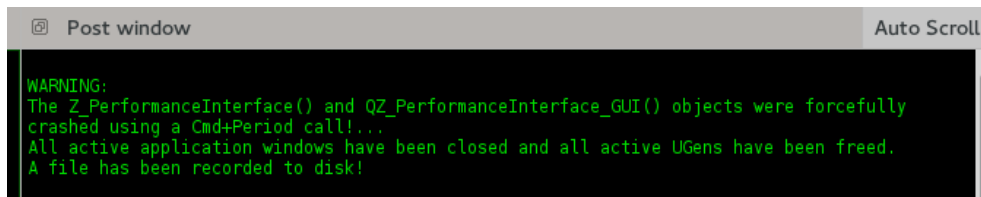
The GUI window can also be closed in the same way by exiting full screen mode and closing the window by clicking on the cross in the top right-hand corner of the window (shortcut key: **Alt+F4**).

The user should bear in mind that when the GUI window is closed in this way, the synths created by the program will still be running in the background. In such circumstances, if the user needs to re-access GUI functionality then the GUI should be reopened, as described in the **Running *Eternal Owl* Call's Performance Interface** section at the start of this user guide.

## Killing the Program Using CmdPeriod

The shortcut key **Ctrl+.** (on Linux or Windows) or **Cmd+.** (on Mac OSX), which calls SuperCollider's commonly used **CmdPeriod** function(s), will close all GUI windows and kill all synths created by the program. This is the best way to fully quit the program.

When **CmdPeriod** is called in this way, the following warning message will show in SuperCollider IDE's post window:

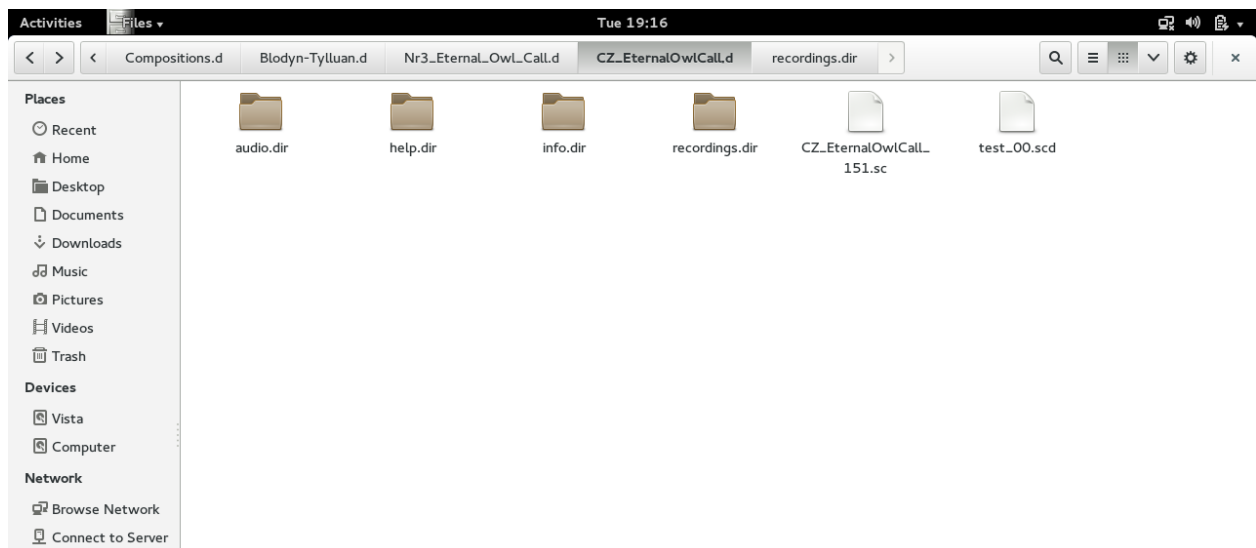


## The User Directory

The user directory is located at the path shown in the post window when the program is first booted as follows:

```
Directory path for reading/writing =
/home/garethh/.local/share/SuperCollider/Extensions/Z_Library.d/Classes.d/Comp
ositions.d/Blodyn-Tylluan.d/Nr3_Eternal_Owl_Call.d/CZ_EternalOwlCall.d
```

An example of this directory opened in a graphical OS environment is shown below:



- The “audio.dir” directory is used for storing the audio files that are used by the program
- The “help.dir” directory is for storing the user documentation files
- The “info.dir” directory is for storing information files (e.g. the file that is opened when the “Subpatch Info...” window is opened)
- The “recordings.dir” directory is the location where the program’s recording sessions are saved (as described in the previous section on **Record Mode**)
- The .sc file contains the SuperCollider library class required for running this program
- The .scd file contains a SuperCollider document that was used for testing the library class

## Shortcut Keys for Real-Time Performance Interface GUI

### Main Window

- Spacebar**            - *Plays the next subpatch*
- Ctrl+Backspace** - *Frees the last subpatch*
- Esc**                - *Exits the GUI screen (i.e. minimises the window)*
- Ctrl+Esc**        - *Frees all active audio synthesizers, including anything registered on the **SystemClock**, **TempoClock** or **AppClock***
- Ctrl+B**            - *Reboots the program*
- Ctrl+Shift+P**    - *Clears the post window*
- F1**                - *Opens the help PDF file*
- F11**              - *Exits/enters full screen mode*
- Ctrl+Q**           - *Closes and destroys the GUI window(s)*
- Ctrl+.**            - *Kills all active SuperCollider synths and routines and closes/destroys the GUI window(s)*

### Additional Windows

- Alt+I**            - *Opens the Subpatch Information Window*  
[internal: **Return** or **Esc** to close]
- (no shortcut key)            - *Modulation Information Window*  
[internal: **Return** or **Esc** to close]
- (activated by other commands) - *Confirmation Dialog Window*  
[internal: **Return**/**Esc** to confirm/cancel]

### Generic Qt-Language Commands Inherited By Each Window [in Linux Fedora Version 20]

- Alt+F4**            - *Closes and destroys the GUI window(s)*
- Alt+Tab**          - *Changes the window via a toggle screen*
- Alt+Esc**          - *Returns to the last active window*
- Alt+Spacebar**   - *Opens the window menu list*
- Super+H**          - *Minimizes the window*
- Super+Up**        - *Maximizes the window*
- Super+Down**     - *Unmaximizes the window*
- Alt+F7**           - *Moves the window*
- Alt+F8**           - *Resizes the window*

## Example Run Script

The following SuperCollider code is an example of a script that can be used for running *Eternal Owl Call*'s performance interface and GUI. The commented-out text that follows the initial script provides technical information and other background information relating to the development of the program and the composition of the piece:

```
z = CZ_EternalOwlCall() // this boots the program and opens the GUI

/*
  if you close and destroy the GUI window while audio synths are still
  running you can reopen the GUI window with one of the two following
  commands...
*/
z.reopenGUI
CZ_EternalOwlCall.class_zpi_pointer.reopenGUI

/**
```

To run this program and load the GUI move the cursor to the first line of this script and press either Ctrl+Enter or Shift+Enter (Linux or Windows) or either Cmd+Enter or Shift+Enter (Mac OSX). A written summary of the program is provided in the comments below...

Author of this Application: Mr Gareth Olubunmi Hughes (1 July 2014 - Present)  
 Copyright (c) : Mr Gareth Olubunmi Hughes (1 July 2014 - Present)  
 Date of Completion: 11 January 2016

This application has been created with the SuperCollider programming language for use with a musical composition for bass flute with live electronic processing, written for performance/recording by avant-garde flautist Carla Rees and the Rarescale contemporary music ensemble. The composition is called "Eternal Owl Call" and depicts the Celtic mythological fable of *Blodeuwedd*, a beautiful maiden who is conjured from flowers and oak to marry a prince but flees and is eventually transformed into an owl for all eternity as punishment for her sin.

The application contains three large patches (which are each divided into respective sub-patches), which generate electronic sounds/effects, read audio files through buffers and process/modify the live flute sound via a microphone signal. Each respective sub-patch should be triggered at a given point in time as indicated in the score.

Students are welcome to perform this piece and to experiment with the source code below; however, if this material is used for the purposes of either a professional performance/recording or the source code is used or modified in any professional capacity then please consult Gareth Hughes (currently a PhD candidate @ The School of Music, Cardiff University, Wales, UK), who is both the composer of the corresponding piece and the author of this computer code.

The code was originally generated and tested using a computer with the following specification...

SuperCollider version 3.7alpha0

OS: Linux Fedora version 20 (with Planet CCRMA + real-time kernel modification)

Hardware: Gateway MA7 laptop with 15.4" screen, Intel 1.73GHz dual-core processor, 2GB RAM, 150GB (5,400 rpm) hard-drive, Focusrite Scarlett 2i2 USB audio interface, Audio Technica AT4040 condenser microphone.

The application has also been tested on an iMac running Mac OSX and SuperCollider version 3.66

However, a workstation or laptop with a higher specification than above is recommended by the author for the purposes of a professional performance or recording as follows...

Recommended System Requirements:

Supercollider version 3.6 or higher.

OS: Linux Fedora/Ubuntu (or any other suitable Linux or UNIX distribution which is supported by SuperCollider), any version of Mac OSX which is supported by SuperCollider. The application has not been tested on MS Windows at the time of writing.

CPU: Intel or AMD dual-core or quad-core processor (including the Intel i3, i5 & i7 series) with a base clock-speed of at least 2.0 GHz.

Memory: 4GB RAM or higher.

Hard Drive: 7,200 rpm or SSD hard-drives are recommended (even on a laptop if possible). Hard drives with 5,400 rpm transfer speed are reasonable but not ideal.

Audio Interface: Any professional-grade soundcard which supports 24-bit audio.

Microphone: Any professional-grade condenser microphone that is suitable for recording a woodwind instrument.

Rarescale recommend the Neumann KM184 (or similar 180 series cardioid mic) fixed on a stand and/or a Sony ECM77B microphone attached to the headjoint of the flute (as recommended by Michael Oliva and specified on Carla Rees' [www.bassflute.co.uk](http://www.bassflute.co.uk) website).

Generally, a normal recording position attached to or near the bass flute is fine; no special or unorthodox microphone placement is required for this piece.

In addition to the electronically generated sounds and signal processors created by this application, the script also requires six specifically named audio files to run correctly. These files must be saved in a directory named "audio.dir", which must be located in the same directory as the application's class script.

The audio files are sound recordings of wildlife, which have been obtained from the Macaulay Library at the Cornell Laboratory of Ornithology (CLO), Ithaca, New York, USA (via the recommendation of Dr Arlene Sierra, who also recently utilised birdsong recordings from this exhaustive audio/video file archive in her composition "Urban Birds" for three pianos and live electronics). A copyright agreement has been reached between Gareth Hughes and CLO for the use of these audio files in the corresponding piece.

All six sound recordings are 16-bit, 48,000 Hz AIFF files. Caution must be taken to ensure that SuperCollider is running at the correct sample rate when this application runs, otherwise the playback rate on the files will not be at the correct speed.



Each respective audio file has the following attributes...

Filename: TawnyOwl\_Germany\_04.aiff  
 Author(s) of the Original Sound Recording: Claus König  
 Location of the Original Sound Recording: Baden-Württemberg, Germany  
 Time & Date of the Original Sound Recording: No information  
 Length of the Original Sound Recording: 01:21  
 Common Name(s) of Species Recorded: Tawny Owl  
 Latin/Scientific Name(s) of Species: *Strix Aluco*  
 Macaulay Library Catalog #: 4539  
 Sound Recording Edited By: Gareth Olubunmi Hughes  
 Length of the Edited Sound Recording: 01:18

Filename: Otter\_Peru\_PigmyOwl\_03.aiff  
 Author(s) of the Original Sound Recording: NPR/NGS Radio Expeditions; Charles Munn;  
 Walter Mancilla Huaman  
 Location of the Original Sound Recording: Madre de Dios, Peru  
 Time & Date of the Original Sound Recording: 14:20, 23 October 1999  
 Length of the Original Sound Recording: 01:43:51  
 Common Name(s) of Species Recorded: Undulated Tinamou; Giant Otter; Pigmy Owl  
 Latin/Scientific Name(s) of Species: *Crypturellus undulatus*; *Pteronura Brasiliensis*; *Glaucidium*  
 Macaulay Library Catalog #: 141271  
 Sound Recording Edited By: Gareth Olubunmi Hughes  
 Length of the Edited Sound Recording: 03:35

Filename: Otter\_CostaRica\_Streams&Birdsong\_01.aiff  
 Author(s) of the Original Sound Recording: David L. Ross, Jr.  
 Location of the Original Sound Recording: Heredia, Costa Rica  
 Time & Date of the Original Sound Recording: 00:00, 3 March 1991  
 Length of the Original Sound Recording: 12:10  
 Common Name(s) Species Recorded: Long-Tailed Otter  
 Latin/Scientific Name(s) of Species: *Lontra Longicaudis*  
 Macaulay Library Catalog #: 72667  
 Sound Recording Edited By: Gareth Olubunmi Hughes  
 Length of the Edited Sound Recording: 06:05

Filename: NeotropicalFrog\_Amazon\_Brasil\_01.aiff  
 Author(s) of the Original Sound Recording: Richard Bierregaard  
 Location of the Original Sound Recording: Amazonas, Brazil  
 Time & Date of the Original Sound Recording: 19:45, 18 February 1989  
 Length of the Original Sound Recording: 02:39  
 Common Name(s) of Species Recorded: Neotropical Frog  
 Latin/Scientific Name(s) of Species: *Leptodactylus Rhodomystax*  
 Macaulay Library Catalog #: 43749  
 Sound Recording Edited By: Gareth Olubunmi Hughes  
 Length of the Edited Sound Recording: 01:56

Filename: NeotropicalFrog\_Peru2\_01.aiff  
Author(s) of the Original Sound Recording: Madre de Dios, Peru  
Location of the Original Sound Recording: Curtis Marantz  
Time & Date of the Original Sound Recording: 16:47, 9 August 1994  
Length of the Original Sound Recording: 02:48  
Common Name(s) of Species Recorded: Neotropical Frog  
Latin/Scientific Name(s) of Species: Leptodactylus Pentadactylus  
Macaulay Library Catalog #: 76075  
Sound Recording Edited By: Gareth Olubunmi Hughes  
Length of the Edited Sound Recording: 02:18

Filename: Mosquito\_CostaRica\_01.aiff  
Author(s) of the Original Sound Recording: David L. Ross, Jr.  
Location of the Original Sound Recording: Heredia, Costa Rica  
Time & Date of the Original Sound Recording: 10:30, 19 May 1996  
Length of the Original Sound Recording: 02:39  
Common Name(s) of Species Recorded: Mosquitos  
Latin/Scientific Name(s) of Species: Culicidae  
Macaulay Library Catalog #: 184738  
Sound Recording Edited By: Gareth Olubunmi Hughes  
Length of the Edited Sound Recording: 01:13

\*/

# APPENDIX 7

SuperCollider Documentation & Source Code for the [Z\\_Library](#):

A class extension library added and used for the creation of  
*Eternal Owl Call's* electronic performance interface



# Contents

<p>Pages 163–219</p> <p>Pages 163–5</p> <p>Pages 166–77</p> <p>Pages 178–82</p> <p>Pages 183–5</p> <p>Pages 186–93</p> <p>Pages 194–7</p> <p>Pages 198–9</p> <p>Pages 200–4</p> <p>Pages 206–7</p> <p>Pages 208–12</p> <p>Pages 214–5</p> <p>Pages 216–9</p>	<p><b>SuperCollider Class Extension Library Documentation:</b></p> <p>Introduction to the <a href="#">Z_Library</a></p> <p><a href="#">CZ_EternalOwlCall</a> A performance interface for use with Hughes's “<i>Eternal Owl Call</i>” composition</p> <p><a href="#">QCZ_EternalOwlCall_GUI</a> A GUI front-end for use with Hughes's “<i>Eternal Owl Call</i>” composition</p> <p><a href="#">QZ_Analyzer</a> A 2-channel frequency analysis window</p> <p><a href="#">QZ_AnalyzerView</a> A 2-channel frequency analysis view</p> <p><a href="#">QZ_ConfirmDialog</a> A customizable dialog window utility with user options</p> <p><a href="#">QZ_Meter</a> A stereo i/o amplitude meter window</p> <p><a href="#">QZ_MeterView</a> A stereo i/o amplitude meter view</p> <p><a href="#">QZ_Scope</a> A stereo oscilloscope window</p> <p><a href="#">QZ_ScopeView</a> A stereo buffer plotting view</p> <p><a href="#">QZ_ServerLevelsPanel</a> A window indicating the current time and the CPU usage levels on a SuperCollider server</p> <p><a href="#">QZ_ServerLevelsPanelView</a> A view indicating the current time and the CPU usage levels on a SuperCollider server</p>
<p>Pages 220–308</p> <p>Pages 220–62</p> <p>Pages 264–87</p> <p>Pages 288–94</p> <p>Page 294</p> <p>Pages 296–7</p> <p>Pages 298–301</p> <p>Pages 301–2</p> <p>Pages 303–5</p> <p>Page 305</p> <p>Pages 306–7</p> <p>Pages 307–8</p>	<p><b>SuperCollider Class Extension Library Source Code:</b></p> <p><a href="#">CZ_EternalOwlCall</a></p> <p><a href="#">QCZ_EternalOwlCall_GUI</a></p> <p><a href="#">QZ_AnalyzerView</a></p> <p><a href="#">QZ_Analyzer</a> [same file as <a href="#">QZ_AnalyzerView</a>]</p> <p><a href="#">QZ_ConfirmDialog</a></p> <p><a href="#">QZ_MeterView</a></p> <p><a href="#">QZ_Meter</a> [same file as <a href="#">QZ_MeterView</a>]</p> <p><a href="#">QZ_ScopeView</a></p> <p><a href="#">QZ_Scope</a> [same file as <a href="#">QZ_ScopeView</a>]</p> <p><a href="#">QZ_ServerLevelsPanelView</a></p> <p><a href="#">QZ_ServerLevelsPanel</a> [same file as <a href="#">QZ_ServerLevelsPanelView</a>]</p>



# Introduction to the Z\_Library

An Introduction to the Z\_Library Class Extension Library

---

See also: [Using Extensions](#), [Using Quarks](#), [Writing Classes](#)

The Z\_Library is a SuperCollider class extension library, which has been added by composer and software developer Gareth Olubunmi Hughes, primarily to support the performance of live electroacoustic music. The extension library includes additional class definitions that can be used for data handling, sound generation or analysis, building GUIs and interfaces that are designed to be used for the creation and performance of a particular musical composition or related item of creative work.

## Class Name Semantics

---

The respective class names of generic Z\_Library classes have the "Z\_" prefix (i.e. "Zed Underscore" [British English] or "Zee Underscore" [American English]). Additional letters are also often added to the start of this prefix to represent further specifics relating to that particular class as follows:

- "Z" by itself represents a generic Z\_Library class (typically used for data handling, algorithms, controlling values and sound generation or analysis),
- "Q" represents a GUI-based class that has been constructed by interfacing the C++ Qt-language API with SuperCollider (i.e. 'QtCollider'),
- "J" represents a GUI-based class that has been constructed by interfacing the Java Swing Toolkit API with SuperCollider (i.e. 'JCollider'),
- "C" represents a class relating to a particular item of compositional work.

At the time of writing, the following class extensions have been added to the Z\_Library:

<a href="#">CZ_EternalOwlCall</a>	A performance interface for use with Hughes's <i>"Eternal Owl Call"</i> composition
<a href="#">QCZ_EternalOwlCall_GUI</a>	A GUI front-end for use with Hughes's <i>"Eternal Owl Call"</i> composition
<a href="#">QZ_Analyzer</a>	A 2-channel frequency analysis window
<a href="#">QZ_AnalyzerView</a>	A 2-channel frequency analysis view
<a href="#">QZ_ConfirmDialog</a>	A customizable dialog window utility with user options
<a href="#">QZ_Meter</a>	A stereo i/o amplitude meter window
<a href="#">QZ_MeterView</a>	A stereo i/o amplitude meter view
<a href="#">QZ_Scope</a>	A stereo oscilloscope window
<a href="#">QZ_ScopeView</a>	A stereo buffer plotting view
<a href="#">QZ_ServerLevelsPanel</a>	A window indicating the current time and the CPU usage levels on a SuperCollider server
<a href="#">QZ_ServerLevelsPanelView</a>	A view indicating the current time and the CPU usage levels on a SuperCollider server

## Syntax Semantics on Variables and Methods

---

Class definitions which are added to the Z\_Library tend to follow fairly rigorous naming conventions, especially on variables and methods which have more than one word in them. As is explained in the above [section](#), class names typically start with a prefixed letter combination (in uppercase), followed by an underscore, followed by a word combination written in **UpperCamelCase**.

Class **Methods** or **Functions** (i.e. anything within the class which can receive arguments or parameters) are normally written in **lowerCamelCase**. As examples of this, within the [CZ\\_EternalOwlCall](#) class, [bellPitchSynth](#) is a method which can receive up to three arguments, whilst [playSample](#) is a getter pointing to a function which can receive up to five arguments.

Class values which are written in `words_separated_by_underscore` format represent either conventional variables (i.e. [Integers](#), [Floats](#), [Booleans](#), [Chars](#), [Strings](#) etc...), collections (i.e. [Arrays](#), [Lists](#) etc...) or objects (i.e. instances of any other more specific class, which is not a conventional variable or collection). As examples of this, within the [QZ\\_ServerLevelsPanelView](#) class, [hour\\_stamp](#), is a getter which points to a string (i.e. a conventional variable), [time\\_label](#) and [server\\_label](#) are both getter/setters which point to instances of the [QStaticText](#) class (i.e. they are both objects) whilst [level\\_label](#) is a getter/setter which points to an [Array](#) containing six separate instances of the [QStaticText](#) class (i.e. a collection).

These naming conventions are applied predominantly; however, there will be some exceptions, in particular where it is sensible to retain historical SuperCollider conventions on certain variable names.

## Directory Extension Semantics

---

Many of the directories within the Z\_Library have extensions at the end of their respective names which represent different meanings (i.e. `".d"` or `".dir"`). These unique directory extensions can be especially useful when searching the library from within a command-line operating system terminal or writing batch scripts to process files/directories within the library.

The `".d"` extension represents a directory that has been added by a software developer, which generally contains class, help or plug-in definition(s) or further subdirectories within it which also contain such definition(s). As examples of this, `"Z_Library.d"` is the library's root directory, containing all of the library's definitions whilst `"Z_Library.d/Classes/GUI.d"` is one of its subdirectories, containing all of the library's GUI-related class definitions.

The `".dir"` extension represents a directory that is used by a class or a program for the purposes of reading/writing data and may also represent a directory which has been automatically created from within the class or from within a program running on that class. As example of this, if you go to the default user directory for the [CZ\\_EternalOwlCall](#) class, returned by evaluating the following line of code:

```
| CZ_EternalOwlCall.filenameSymbol.asString.dirname
```

you will notice that the directory includes a number of subdirectories with a `".dir"` extension. The `"audio.dir"` directory contains a number of audio files which are read and processed by the class whilst the `"recordings.dir"` directory is reserved by the class as an area for writing audio files to disk. In addition, if a `"recordings.dir"` directory does not already exist within the class's directory then a new directory with that name is created by calling the [File: \\*mkdir](#) method.

These naming conventions are applied predominantly; however, there will be some exceptions, in particular where it is sensible to retain historical SuperCollider conventions on certain directory names. As an example, the Z\_Library extension's root directory contains three immediate subdirectories named `"Classes"`, `"HelpSource"` and `"Plugins"` (i.e. without a directory extension) as this adheres to convention within the root directory of any SuperCollider class library or class extension library (see the [Using Extensions: How Extensions Folders Should be Organised](#) guide section for further information on this).



---

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Guides/Z\_Library-  
Introduction.schelp  
link::Guides/Z\_Library-Introduction::  
sc version: 3.7alpha0

# CZ\_EternalOwlCall

A performance interface for use with Hughes's "Eternal Owl Call" composition.

Source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/Classes/Compositions.d/Blodyn-Tylluan.d/Nr3\_Eternal\_Owl\_Call.d/CZ\_EternalOwlCall.d/[CZ\\_EternalOwlCall\\_155.sc](#)  
Inherits from: [Object](#)

See also: [QCZ\\_EternalOwlCall\\_GUI](#)

## Description

CZ\_EternalOwlCall is a performance interface for use with Gareth Olubunmi Hughes's mixed medium composition "*Eternal Owl Call*", for bass flute & live electronics. Its commands and methods should be controlled by an instance of the [QCZ\\_EternalOwlCall\\_GUI](#) GUI class, which is automatically created when a new instance is called.

Example:

```
z = CZ_EternalOwlCall() // this boots the program and opens the GUI

/*
  if you close and destroy the GUI window while audio synths are still running
  you can reopen the GUI window with one of the two following commands...
*/
z.reopenGUI
CZ_EternalOwlCall.class_zpi_pointer.reopenGUI
```

## Class Methods

\* **new**

Creates a new CZ\_EternalOwlCall instance. The method does not require any arguments.

\* **initSynthDefs**

Initiates several [SynthDefs](#) used by this class at [Startup](#) as follows:

```
// SynthDef to record the session and save to disk:
\recordSession

// SynthDefs & buses for effects processors:
\stereoReverbChannel
\combChannel
12.do{ arg i; "delayTap"++i } // 12 separate delay taps
\atmosphericReverbChannel

// SynthDefs for sample playback:
\stereoSamplePlayer
\monoToStereoSamplePlayer
\monoSamplePlayer
```

```

// SynthDefs for sample doppler effects:
\monoCircularSampleDoppler
\stereoTriangularSampleDoppler
\monoCircularFormantDoppler
\stereoTriangularFormantDoppler

// SynthDefs for sample pitch shifts & frequency modulation:
\monoSampleShift
\monoToStereoSampleShift
\monoToStereoFormantShift

// SynthDef for the bell modelling synth:
\grandfatherClockPitch

// SynthDefs for the modular synths:
\midModPulse
\lowModPulse
\bassPulse

```

\* **valid\_instance\_count**

\* **valid\_instance\_count = value**

The count of valid instances to be created by this class. By default this cannot exceed 1. This is to prevent the potentially problematic creation of multiple instances of this powerful, memory-consuming interface. If the user attempts to create a second valid instance of this class a warning is sent to the post window and the subsequent object is rendered invalid, meaning that it cannot be used in any meaningful way.

Returns:

an [Integer](#)

\* **invalid\_instance\_count**

\* **invalid\_instance\_count = value**

The count of invalid instances to be created by this class. See above for further clarification.

Returns:

an [Integer](#)

\* **allow\_multiple\_instances**

\* **allow\_multiple\_instances = value**

**WARNING: General SuperCollider users should avoid modifying this value!**

A [Boolean](#) indicating whether multiple valid instances of this class can be created or not (as explained above). The default value is **false** and this should only ever need to be modified for advanced software development-related reasons.

Returns:

a [Boolean](#)

\* **class\_zpi\_pointer**

\* **class\_zpi\_pointer** = value

A pointer to the last valid instance to be created by this class.

**Returns:**

an object pointer to an instance of CZ\_EternalOwlCall (i.e. this class!)

\* **server**

The server which the performance interface is running on.

**Returns:**

an instance of [Server](#)

## Inherited class methods

9 methods from [Object](#) ► [show](#)

## Instance Methods

### Data Handling

- **valid\_instance**

- **valid\_instance** = value

A [Boolean](#) indicating whether the instance is valid or not. This is determined by evaluating the *\*valid\_instance\_count* and *\*allow\_multiple\_instances* class variables. If the instance returns **false** then it cannot be used in any meaningful way.

**Returns:**

a [Boolean](#)

- **next\_patch**

- **next\_patch** = value

The pch value for the next sequential subpatch (e.g. for subpatch # 1.05, pch = 1).

**Returns:**

an [Integer](#)

- **next\_subpatch**

- **next\_subpatch** = value

The sub value for the next sequential subpatch (e.g. for subpatch # 1.05, sub = 5).

---

**Returns:**

an [Integer](#)

---

- **last\_patch**


---

- **last\_patch** = value

The pch value for the last sequential subpatch (e.g. for subpatch # 1.05, pch = 1).

---

**Returns:**

an [Integer](#)

---

- **last\_subpatch**


---

- **last\_subpatch** = value

The sub value for the last sequential subpatch (e.g. for subpatch # 1.05, sub = 5).

---

**Returns:**

an [Integer](#)

---

- **dpath**

The root directory path which the program reads from and writes to. This value is set when the [-programStart](#) method is called.

This normally returns `CZ_EternalOwlCall.filenameSymbol.asString.dirname`

---

**Returns:**

a [String](#)

## Booting & Rebooting

---

- **programStart** (directory\_path)

Boots or reboots the program and loads the GUI.

---

**Arguments:**

**directory\_path** The root directory path which the program reads from and writes to. This will also be automatically written to the [-dpath](#) value.

---

**Discussion:**

Example:

```
z = CZ_EternalOwlCall() // this boots the program and opens the GUI
z.programStart(CZ_EternalOwlCall.filenameSymbol.asString.dirname) // reboots
```

## Subpatches

### - **subpatch** (pch: 1, sub: 1)

Starts or restarts a subpatch.

#### Arguments:

**pch** an [Integer](#) representing the patch value

**sub** an [Integer](#) representing the subpatch value

#### Discussion:

Example:

```
z = CZ_EternalOwlCall() // this boots the program and opens the GUI

// Start/Restart Each Subpatch ...
// Patch #1 ...      Atmospheric reverb is sent to the microphone signal
z.subpatch(1,01) // Ambient neotropical nature sounds fade in
z.subpatch(1,02) // Harmonized owl call emerges and freq mod is added
z.subpatch(1,03) // Pan modulation is added to the owl & frog sounds
z.subpatch(1,04) // Audio signal fades out over 142.5 seconds
z.subpatch(1,05) // Adds 7 delay taps to the atmospheric reverb channel

// Patch #2 ...      Additive Synths...
z.subpatch(2,01) // 2nd, 3rd & 5th partials of C3
z.subpatch(2,02) // 5th, 3rd & 2nd partials of D3
z.subpatch(2,03) // 2nd, 3rd, 5th & 7th partials of Db2
z.subpatch(2,04) // 7th, 5th, 3rd & 2nd partials of Eb2
z.subpatch(2,05) // 2nd, 5th, 3rd & 7th partials of F2
z.subpatch(2,06) // 2nd, 3rd, 5th & 7th partials of G2
z.subpatch(2,07) // 7th, 11th & 13th partials of both C3 & D3
z.subpatch(2,08) // Adds a looped bass line of deep bell-like pitches
z.subpatch(2,09) // 17th, 19th & 23rd partials of both C3 & D3
z.subpatch(2,10) // Silences each synth in patch #2 & owl call xfades in
z.subpatch(2,11) // Removes reverb/delay from the mic input... END OF SECTION

// Patch #3 ...
z.subpatch(3,01) // Adds reverb & 12 delay taps to the synths and mic input
z.subpatch(3,02) // Adds a bass pulse & modulated sine waves
z.subpatch(3,03) // Adds a comb filter
z.subpatch(3,04) // xfades reverberated synths & modulated nature sounds
z.subpatch(3,05) // xfades nature sounds & reverberated/delayed owl call
z.subpatch(3,06) // Fades out to silence... END OF SECTION
```

### - **freeSub** (pch: 1, sub: 1)

Frees or reverses a subpatch.

#### Arguments:

**pch** an [Integer](#) representing the patch value

**sub** an [Integer](#) representing the subpatch value

#### Discussion:

Example:

```
z = CZ_EternalOwlCall() // this boots the program and opens the GUI
```

```

// Free or Reverse Each Subpatch ...
// Patch #1 ...
z.freeSub(1,01) // | Frees PlayBuf synths from corresponding subpatch... |
z.freeSub(1,02) // |
z.freeSub(1,03) // Stops pan modulation and restores original pan values
z.freeSub(1,04) // Restarts all previous subpatches in patch #1
z.freeSub(1,05) // Removes 7-tap delay from the reverb channel
// Patch #2 ...
z.freeSub(2,01) // |
z.freeSub(2,02) // |
z.freeSub(2,03) // |
z.freeSub(2,04) // |
z.freeSub(2,05) // | Frees additive synths from corresponding subpatch... |
z.freeSub(2,06) // |
z.freeSub(2,07) // |
z.freeSub(2,08) // |
z.freeSub(2,09) // |
z.freeSub(2,10) // Stops owl call and clears routine to free additive synths
z.freeSub(2,11) // Adds 7-tap delay and reverb back onto the mic signal
// Patch #3 ...
z.freeSub(3,01) // Restores 7-tap delay/reverb with only a direct mic signal
z.freeSub(3,02) // Frees modular synths
z.freeSub(3,03) // Silences the comb filter
z.freeSub(3,04) // Restores ReverbGroupChannel & restarts modular synths...
z.freeSub(3,05) // Restarts the modulated nature sounds fade-in
z.freeSub(3,06) // Restarts all previous subpatches in patch #3

```

## - freeAllSynths

Frees all active audio synths, including anything scheduled on the [SystemClock](#), [TempoClock](#) or [AppClock](#). This also resets the microphone bus routing setting to "Microphone to Reverb Channel".

## Modulation Routines

### - setFreqModulation

Returns a [Function](#) which sets or resets the randomly generated frequency modulation values sent to subpatch #1.02, #1.03 & #3.04.

### - setPanModulation

Returns a [Function](#) which sets or resets the randomly generated panoramic modulation values sent to subpatch #1.02, #1.03 & #3.04.

### - freeFreqModulation

Returns a [Function](#) which frees the frequency modulation values sent to subpatch #1.02, #1.03 & #3.04.

### - freePanModulation

Returns a [Function](#) which frees the panoramic modulation values sent to subpatch #1.02, #1.03 & #3.04.

## Microphone Effects Bus Routing

### - `directMicrophoneSignal`

Returns a [Function](#) which sets the microphone bus routing setting to "Direct Microphone Signal Only".

### - `sendMicToBus_Reverb`

Returns a [Function](#) which sets the microphone bus routing setting to "Microphone to Reverb Channel".

### - `sendMicToBus_Reverb__7TapDelay`

Returns a [Function](#) which sets the microphone bus routing setting to "Mic to Reverb + 7 Tap Delay".

### - `sendMicToBus_Reverb_12TapDelay`

Returns a [Function](#) which sets the microphone bus routing setting to "Mic to Reverb + 12 Tap Delay".

### - `sendMicToBus_Reverb_12TapDelay_Comb`

Returns a [Function](#) which sets the microphone bus routing setting to "Reverb + 12 Tap Delay + Comb Filter".

## Dynamics Processing

### - `masterChannel`

Sets the values of a low-cut filter and a peak limiter on the [RootNode](#).

## Recording & Playback

### - `startRecording`

Returns a [Function](#) which starts recording the [RootNode](#)'s audio signal to a 16-bit, 48kHz AIFF file. The file is saved in the "recordings.dir" directory, which is a subdirectory of the path returned by the [-dpath](#) method.

### - `stopRecording`

Returns a [Function](#) which stops recording the [RootNode](#)'s audio signal and saves the session to a 16-bit, 48kHz AIFF file.

### - `playLastRecordedSession`

Returns a [Function](#) which plays the last recorded session, providing that at least one session has been recorded since the program was first started.



## Audio Sample & Effects Auditioning

### - playSample

Returns a [Function](#) which allows the user to play and audition any one of the six audio samples, which are used as source material for some of the program's subpatches.

#### Arguments:

<b>(symbol)</b>	The symbolic name of the audio file to be played. This directs the function to the buffer index of the corresponding audio file.
<b>(channels)</b>	Number of audio channels that the buffer will be (i.e. mono = 1, stereo = 2). This must be a fixed integer. The architecture of the SynthDef cannot change after it is compiled.
<b>(rate)</b>	The ratio of the sample playback speed: 1.0 is the server's sample rate, 2.0 is one octave up, 0.5 is one octave down -1.0 is backwards normal rate... etc. Interpolation is cubic.
<b>(start_pos_secs)</b>	The start position of the audio file in seconds.
<b>(loop)</b>	An <a href="#">Integer</a> indicating whether the audio sample will loop or not (off = 0, on = 1). This is modulateable.

#### Discussion:

Example:

```
z = CZ_EternalOwlCall() // this boots the program and opens the GUI

// Audio file auditioning commands ...
z.playSample.(symbol:"TawnyOwl" ,channels:1, rate:1, start_pos_secs:0, loop:1)
z.playSample.(symbol:"PigmyOwl" ,channels:2, rate:1, start_pos_secs:0, loop:1)
// start_pos_secs:285, to get 'colourful splashes' at end of otter file...
z.playSample.(symbol:"Otter" ,channels:1, rate:1, start_pos_secs:0, loop:1)
z.playSample.(symbol:"FrogAmazon",channels:1, rate:1, start_pos_secs:0, loop:1)
z.playSample.(symbol:"FrogPeru2" ,channels:1, rate:1, start_pos_secs:0, loop:1)
z.playSample.(symbol:"Mosquito" ,channels:2, rate:1, start_pos_secs:0, loop:1)

// free audition playback...
z.freeAudition.()
```

### - playDoppler

Returns a [Function](#) which allows the user to play and audition four examples of the doppler-based processing synths, which are used with some of the program's subpatches.

#### Arguments:

<b>(symbol)</b>	The symbolic name of the audio file which the effect is applied to. This directs the function to the buffer index of the corresponding audio file and also indicates the type of doppler effect required (i.e. sample or formant-based).
-----------------	--

#### Discussion:

Example:

```
z = CZ_EternalOwlCall() // this boots the program and opens the GUI

// Doppler auditioning commands ...
z.playDoppler.(symbol:"OtterSampleDoppler" )
z.playDoppler.(symbol:"MosquitoSampleDoppler" )
```

```

z.playDoppler.(symbol:"OtterFormantDoppler"    )
z.playDoppler.(symbol:"MosquitoFormantDoppler")

// free audition playback...
z.freeAudition.()

```

## - freeAudition

Returns a [Function](#) which frees any audio sample that is being auditioned in the program.

## Bell Modelling (Patch # 2)

### - bellPitchSynth (semitone\_offset: 0, pan: 0, amp: 0.0875)

Plays a bell synth pitch, generated using the [Klank](#) class and the [\grandfatherClockPitchSynthDef](#) (a definition added by this class at [Startup](#) via the [\\*initSynthDefs](#) class method). The synth created will automatically be freed following its release using a [doneAction](#) value of 2 on its envelope.

#### Arguments:

<b>semitone_offset</b>	The offset, in semitones, from the default pitch of C3 (130.81Hz or <a href="#">48.midicps</a> ).
<b>pan</b>	The panoramic value.
<b>amp</b>	The amplitude ratio.

#### Returns:

an instance of [Synth](#)

#### Discussion:

Example:

```

z = CZ_EternalOwlCall() // this boots the program and opens the GUI

48.midicps // returns the bell's default frequency

z.bellPitchSynth()           // plays C3
z.bellPitchSynth(semitone_offset: 2) // plays D3
z.bellPitchSynth(semitone_offset: -2) // plays Bb2
z.bellPitchSynth(semitone_offset: 0.5) // plays C3 + a quartertone

```

### - additiveSynth (midi\_value: 48, input\_partial: 2, input\_gain: 1, pan: 0, isynth: 0, jsynth: 0, bell: true, fade\_in: 0.02, gate: 1, release\_time: 8)

Plays an array of bell-like additive synth pitches based on partials of the natural harmonic series. The array contains a maximum of 11 elements, however any partial frequencies above 20kHz will be rejected and not added to the array. Each additive synth element plays a [SinOsc](#) (i.e. sine wave) synth modulated by a [Pulse](#) (i.e. square wave) amplitude filter.

#### Arguments:

<b>midi_value</b>	The midi value of the lowest pitch which is heard when the <code>input_partials</code> arg = 2 (i.e. its default value). The fundamental pitch of the natural harmonics heard is actually an octave lower than this (i.e. <a href="#">36.midicps</a> on the default).
-------------------	---

<b>input_partial</b>	The input partial of the harmonic series, which acts as the base (i.e. lowest pitched) frequency value heard on the array of additive synths.
<b>input_gain</b>	The input amplitude ratio.
<b>pan</b>	The panoramic value.
<b>isynth</b>	The p2[i] index of the 2-dimensional array where the synth is stored in memory.
<b>jsynth</b>	The p2[j] index of the 2-dimensional array where the synth is stored in memory.
<b>bell</b>	A <a href="#">Boolean</a> indicating whether or not a <a href="#">\grandfatherClockPitch</a> synth (called via the <a href="#">-bellPitchSynth</a> method) is heard on input frequency when the additive synth is first started.
<b>fade_in</b>	Not yet implemented.
<b>gate</b>	The release gate (not yet implemented).
<b>release_time</b>	Not yet implemented, however, the individual additive synth array elements can be released by calling the <a href="#">Node: -release</a> method and supplying a <code>releaseTime</code> arg.

#### Returns:

an [Array](#) of additive synths

#### Discussion:

Example:

```
z = CZ_EternalOwlCall() // this boots the program and opens the GUI

a = z.additiveSynth(input_gain: 6) // plays partials of C3
b = z.additiveSynth(input_gain: 6, midi_value: 50) // plays partials of D3

a.free // frees synth 'a'
b.size.do{ arg i; b[i].release(10) } // releases synth 'b' over 10 seconds
```

#### - p2\_DebugData

Dumps debug data for the bell-modelling arrays in patch # 2.

## GUI

#### - gui

#### - gui = value

A pointer to the performance interface's corresponding GUI window.

#### Returns:

an object pointer to an instance of [QCZ\\_EternalOwlCall\\_GUI](#)

#### - reopenGUI

This method allows the user to reopen the GUI window if it has already been closed/destroyed while the program's synths are still running.

## Inherited instance methods

314 methods from Object ► [show](#)

## Undocumented instance methods

- **addTaps** (taps: 7, new\_at\_start: true)
- **bus\_Reverb** (new\_at\_start: true, delay\_bus\_exists: false, comb\_bus\_exists: false)
- **bus\_Reverb\_12TapDelay**
- **bus\_Reverb\_12TapDelay\_Comb** (start\_comb: true)
- **bus\_Reverb\_\_7TapDelay**
- **directMicSignal**
- **freeAdditiveSub** (i)
- **freeEffects**
- **freeFreqMod**
- **freeNeotropicalAmbience**
- **freeOwlHarmonizer**
- **freePanMod**
- **freeSynths\_p2** (i\_min, i\_max, j\_min, j\_max, sum, interval)
- **freqModLoop** (wait\_before: 30, wait\_after: 0)
- **initBuses**
- **initProgram** (directory\_path)
- **initSubpatchFlags**
- **init\_p2**
- **invalidSubpatch** (pch, sub)
- **neotropicalAmbience** (fade\_in: 15, target)
- **owlHarmonizer** (mod\_at\_start: true, wait\_before: 30, wait\_after: 0, target)
- **panModLoop**
- **setLastSubpatchValue** (pch, sub)
- **setNextSubpatchValue** (pch, sub)
- **validSubpatch** (pch, sub, symbol: "started")

- **validateSubpatch** (flag, pch, sub)

---

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Classes  
/CZ\_EternalOwlCall.schelp  
link::Classes/CZ\_EternalOwlCall::  
sc version: 3.7alpha0

# QCZ\_EternalOwlCall\_GUI

A GUI front-end for use with Hughes's "Eternal Owl Call" composition.

Source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/Classes/Compositions.d/Blodyn-Tylluan.d/Nr3\_Eternal\_Owl\_Call.d/QCZ\_EternalOwlCall\_GUI.d/QCZ\_EternalOwlCall\_GUI\_92.sc  
Inherits from: QWindow : Object

See also: [CZ\\_EternalOwlCall](#)

## Description

QCZ\_EternalOwlCall\_GUI is a GUI front-end for use with Gareth Olubunmi Hughes's mixed medium composition "*Eternal Owl Call*", for bass flute & live electronics. Its functionality controls an instance of the [CZ\\_EternalOwlCall](#) performance interface class.

A new instance of QCZ\_EternalOwlCall\_GUI is automatically created when a new instance [CZ\\_EternalOwlCall](#) is called - this is the most logical way to instantiate this class and users should refer to the coding example at the start of the documentation for the [CZ\\_EternalOwlCall](#) class for guidance on how this should be done.

It is also possible to link the two classes by instantiating the QCZ\_EternalOwlCall\_GUI class and supplying a corresponding [CZ\\_EternalOwlCall](#) instance as an argument. However, this method of running the program is not recommended because it will cause the SuperCollider server to be booted twice.

Example:

```
/*
  WARNING: This is not the recommended method to run the program because
  it will cause the SuperCollider server to be booted twice...
*/
g = QCZ_EternalOwlCall_GUI( CZ_EternalOwlCall.new ) // boots and opens the GUI

/*
  if you close and destroy the GUI window while audio synths are still running
  you can reopen the GUI window with one of the two following commands...
*/
g.zpi.reopenGUI
CZ_EternalOwlCall.class_zpi_pointer.reopenGUI
```

## Class Methods

```
* new (zpi_pointer,
name: "Eternal Owl Call : Real-Time Performance Interface", bounds,
resizable: true, border: true, server, scroll: true)
```

Creates a new QCZ\_EternalOwlCall\_GUI instance.

### Arguments:

- zpi\_pointer** A pointer to the GUI's corresponding instance of [CZ\\_EternalOwlCall](#) (the performance interface which handles data, as well as the allocation and freeing of audio-rate and control-rate ugens and synths).
- name** A String for the text that will be displayed in the title bar. The default is "Eternal Owl Call : Real-Time Performance Interface".

<b>bounds</b>	A <a href="#">Rect</a> specifying position and size of the window. The size does not include the border and title bar. Position is measured from the bottom-left corner of the screen (this is different than <a href="#">View: -bounds</a> ).
<b>resizable</b>	A <a href="#">Boolean</a> indicating whether this window is resizable by the user. The default is <b>true</b> .
<b>border</b>	A <a href="#">Boolean</a> indicating whether this window has a border. Borderless windows have no title bar and thus can only be closed in code. The default is <b>true</b> .
<b>server</b>	This is a dummy argument which is here to provide compatibility with SwingOSC and has no effect.
<b>scroll</b>	A <a href="#">Boolean</a> indicating whether this window will add scrollbars if its contents exceed its bounds. If this is set to <b>true</b> , then <a href="#">View: -resize</a> settings will be ignored for contained views. The default is <b>true</b> .

\* **class\_gui\_pointer**

\* **class\_gui\_pointer** = value

A pointer to the last valid instance to be created by this class.

#### Returns:

an object pointer to an instance of `QCZ_EternalOwlCall_GUI` (i.e. this class!)

\* **server**

The server which the performance interface is running on.

#### Returns:

an instance of [Server](#)

## Inherited class methods

9 methods from [QWindow](#) ► [show](#)

9 methods from [Object](#) ► [show](#)

## Instance Methods

- **post\_window**

- **post\_window** = value

The text view for the post window, which is contained within the main GUI window.

#### Returns:

an instance of [TextView](#)

- **z\_meter\_view**

- **z\_meter\_view** = value

The amplitude meter view which is contained within the main GUI window.

**Returns:**

---

an instance of [QZ\\_MeterView](#)

- **z\_scope\_view**

- **z\_scope\_view** = value

The oscilloscope view which is contained within the main GUI window.

**Returns:**

---

an instance of [QZ\\_ScopeView](#)

- **z\_analyzer\_view**

- **z\_analyzer\_view** = value

The frequency analyzer view which is contained within the main GUI window.

**Returns:**

---

an instance of [QZ\\_AnalyzerView](#)

- **z\_server\_levels\_panel\_view**

- **z\_server\_levels\_panel\_view** = value

The server levels panel view which is contained within the main GUI window.

**Returns:**

---

an instance of [QZ\\_ServerLevelsPanelView](#)

- **zpi**

- **zpi** = value

A pointer to the GUI's corresponding performance interface (which handles data, as well as the allocation and freeing of audio-rate and control-rate ugens and synths).

**Returns:**

---

an object pointer to an instance of [CZ\\_EternalOwlCall](#)



## - **postMsg** (value, line\_feed: true)

Posts a value as a string on the post window.

### Arguments:

- value** The value to be posted below any text which already exists on the post window. If the value supplied is not a string, then the method will convert it into one.
- line\_feed** A [Boolean](#) to indicate whether or not a `"\n"` character (i.e. a 'line feed') is automatically added to the end of the string. The default value is **true**

### Returns:

the [String](#) which is returned by the post window's [TextView](#)

## - **setNextLabel**

Sends the correct information to the "Next Subpatch" label.

## - **setMode** (value)

Sets the program's mode of operation.

### Arguments:

- value** An [Integer](#), where 0 = Testing Mode and 1 = Performance Mode

## - **setRecordModeButton** (mode: 'on')

Sets the value on the record mode button. This will not perform the value action, it will simply change the button's value.

### Arguments:

- mode** This must be either `\on` or `\off`

## - **freeRecPlayButton**

Resets the default play value on the "Last Recorded Session" button. This will not perform the value action, it will simply change the button's value.

## - **freePlayButtons**

Resets the default play values on the "Last Recorded Session", "Audio Sample Auditioning" and "Doppler Effect Auditioning" buttons. This will not perform any value actions, it will simply change the buttons' values.

## - **setMicBusButton** (i)

Sets a value on the "Microphone Bus Routing" button and resets all the other buttons to their respective default values (as only one setting can exist at any given time). This will not perform any value actions, it will simply change the buttons' values.

### Arguments:

i The index of the "Microphone Bus Routing" panel's button array, where:

- [0] = "Direct Microphone Signal Only"
- [1] = "Microphone to Reverb Channel"
- [2] = "Mic to Reverb + 7 Tap Delay"
- [3] = "Mic to Reverb + 12 Tap Delay"
- [4] = "Reverb + 12 Tap Delay + Comb Filter"

## - **closeOtherWindows**

Closes all windows in the GUI, except for the main window (namely the confirmation dialog window, the subpatch information window and the modulation information window).

## Inherited instance methods

37 methods from **QWindow** ► [show](#)

315 methods from **Object** ► [show](#)

## Undocumented instance methods

- **openDialog** (bounds: 'default', message: 'default',  
button\_strings: 'default', completion\_function)

---

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Classes  
/QCZ\_EternalOwlCall\_GUI.schelp  
link::Classes/QCZ\_EternalOwlCall\_GUI::  
sc version: 3.7alpha0

## SuperCollider CLASSES (extension)

[Z\\_Library>GUI>Interfaces](#), [GUI>Interfaces](#)

# QZ\_Analyzer

A 2-channel frequency analysis window.

Source: [/home/garethh/.local/share/SuperCollider/Extensions/Z\\_Library.d/Classes/GUIs.d/QtCollider.d/Signal\\_Analysis.d/QZ\\_Analyzer.d/QZ\\_Analyzer\\_028.sc](#)  
Inherits from: [QWindow](#) : [Object](#)

See also: [QZ\\_AnalyzerView](#), [FreqScopeView](#), [FreqScope](#)

## Description

QZ\_Analyzer displays a [QWindow](#) containing a [QZ\\_AnalyzerView](#), which is a 2-channel (stereo) frequency analyzer showing the frequency spectrum of two separate audio busses.

Example:

```
s.boot           // boot the server first!
w = QZ_Analyzer() // opens the frequency analysis window
w.close          // closes the window
```

## Class Methods

\* **new** (name: "Z\_Analyzer", bounds, resizable: false, border: true, server, scroll: false)

Creates a new QZ\_Analyzer instance.

### Arguments:

- name** A String for the text that will be displayed in the title bar. The default is "Z\_Analyzer".
- bounds** A [Rect](#) specifying position and size of the window. The size does not include the border and title bar. Position is measured from the bottom-left corner of the screen (this is different than [View: -bounds](#)).
- resizable** A [Boolean](#) indicating whether this window is resizable by the user. The default is **false**.
- border** A [Boolean](#) indicating whether this window has a border. Borderless windows have no title bar and thus can only be closed in code. The default is **true**.
- server** This is a dummy argument which is here to provide compatibility with SwingOSC and has no effect.
- scroll** A [Boolean](#) indicating whether this window will add scrollbars if its contents exceed its bounds. If this is set to **true**, then [View: -resize](#) settings will be ignored for contained views. The default is **false**.

## Inherited class methods

9 methods from [QWindow](#) ► [show](#)

9 methods from [Object](#) ► [show](#)

## Instance Methods

- **z\_analyzer\_view**

- **z\_analyzer\_view** = value

The frequency analysis view which is contained within the [QWindow](#)

Returns:

an instance of [QZ\\_AnalyzerView](#)

## Inherited instance methods

37 methods from [QWindow](#) ► [show](#)

315 methods from [Object](#) ► [show](#)

## Examples

```
s.boot // boot the server first!

w = QZ_Analyzer() // opens the frequency analysis window

z = w.z_analyzer_view // creates a pointer to the window's QZ_AnalyzerView() instance

// play sounds...
(
a = { SinOsc.ar(freq:330, mul:0.35) }.play;
b = { Pan2.ar( SinOsc.ar(freq:220), pos:1, level:0.35 ) }.play;
)

// change the button colors...
z.freqscope_settings_button.states = [["ANALYZER SETTINGS", Color.white, Color.blue]]
// change the title label color...
z.freqscope_label.stringColor = Color.white
// change the "L" button properties...
z.left_button.states = [["0", Color.cyan, Color.magenta.alpha_(0.25)]]
// change the "R" button properties...
z.right_button.states = [["1", Color.yellow, Color.magenta.alpha_(0.25)]]
// change the scope display colors...
(
z.wave_color_left = [ Color.cyan ];
z.wave_color_right = [ Color.yellow ];
)
// change the color of the grid readings...
z.readings_color = Color.white
// change the gridline colors...
(
z.grid_color_left = Color.green;
z.grid_color_right = Color.red;
)
// resize the window and view bounds and increase the number of readings...
(
w.bounds = Rect(200,500,835,492);
z.freq_readings_width = 700;
z.db_readings_height = 400;
z.number_of_freq_readings = 18;
z.number_of_db_readings = 17;
)
```

```
// free sounds...  
(  
a.free; a = nil;  
b.free; b = nil;  
)  
  
w.close // close the window
```

---

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Classes/QZ\_Analyzer.schelp  
link::Classes/QZ\_Analyzer::  
sc version: 3.7alpha0

# QZ\_AnalyzerView

A 2-channel frequency analysis view.

Source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/Classes/GUIs.d/QtCollider.d/Signal\_Analysis.d/QZ\_Analyzer.d/QZ\_Analyzer\_029.sc  
Inherits from: [QView](#) : [QObject](#) : [Object](#)

See also: [QZ\\_Analyzer](#), [FreqScopeView](#), [FreqScope](#)

## Description

QZ\_AnalyzerView is a 2-channel (stereo) frequency analyzer which shows the frequency spectrum of two separate audio busses. The class extends [QView](#) and can be added as a cell into a [QWindow](#) or one of the window's child views. QZ\_AnalyzerView also supports the implementation of [QZ\\_Analyzer](#).

## Class Methods

### \* **new** (parent, bounds)

Creates a new instance of QZ\_AnalyzerView that can be passed as a child view to a [QView](#) or a [QWindow](#), effectively placing it within the parent's visual space. If there is a [decorator](#) installed on the parent, it will manage the position of the new QZ\_AnalyzerView.

#### NOTE: Qt GUI:

The 'parent' argument may be omitted, in which case the view will be displayed as a window on its own, when [shown](#).

The 'bounds' argument may be omitted, in which case the view will be created with its preferred size at position (0,0).

If a parent is given and there is a layout installed on it, the layout will manage the position and size of this view and the 'bounds' argument will have no effect.

#### Arguments:

**parent** The instance of [QView](#) or [QWindow](#) that the new view will become a child of.

**bounds** A [Rect](#) or a [Point](#) describing the size and position of the new QZ\_AnalyzerView. If a Point is given, its coordinates will denote the view's size, while the view's position will be (0,0). Position is measured relative to the parent's top-left corner.

#### Returns:

a new instance of QZ\_AnalyzerView

### \* **server**

The server from which the analyzer is reading.

#### Returns:

an instance of [Server](#)

## \* `sample_rate`

The sample rate which the server is running at.

### Returns:

a sample rate value

## Inherited class methods

15 methods from `QView` ► [show](#)

16 methods from `QObject` ► [show](#)

9 methods from `Object` ► [show](#)

## Instance Methods

### - `freqscope_label`

### - `freqscope_label = value`

The title label.

### Returns:

an instance of [QStaticText](#)

### - `freqscope_settings_button`

### - `freqscope_settings_button = value`

The "ANALYZER SETTINGS" button.

### Returns:

an instance of [QPushButton](#)

### - `left_button`

### - `left_button = value`

The "L" button.

### Returns:

an instance of [QPushButton](#)

### - `right_button`

### - `right_button = value`

The "R" button.

### Returns:

an instance of [QPushButton](#)

## - `readings_color`

### - `readings_color = color`

The string color of the numerical readings on the grid.

#### Returns:

---

an instance of [Color](#)

## - `grid_color_left`

### - `grid_color_left = color`

The color of the gridlines and the border for the left channel display.

#### Returns:

---

an instance of [Color](#)

## - `grid_color_right`

### - `grid_color_right = color`

The color of the gridlines and the border for the right channel display.

#### Returns:

---

an instance of [Color](#)

## - `wave_color_left`

### - `wave_color_left = color`

The color of the waveform on the left channel display. This must be wrapped inside an array to work correctly (i.e. `[color]` ).

#### Returns:

---

an [Array](#) containing a single instance of [Color](#)

## - `wave_color_right`

### - `wave_color_right = color`

The color of the waveform on the right channel display. This must be wrapped inside an array to work correctly (i.e. `[color]` ).

#### Returns:

---

an [Array](#) containing a single instance of [Color](#)

## - `number_of_freq_readings`

### - `number_of_freq_readings = n`



The number of individual frequency readings displayed.

Returns:

---

an [Integer](#)

- **freq\_readings\_width**

- **freq\_readings\_width** = width

The width of the frequency readings area.

Returns:

---

an [Integer](#)

- **freq\_readings\_height**

- **freq\_readings\_height** = height

The height of the frequency readings area.

Returns:

---

an [Integer](#)

- **number\_of\_db\_readings**

- **number\_of\_db\_readings** = n

The number of individual decibel readings displayed.

Returns:

---

an [Integer](#)

- **db\_readings\_width**

- **db\_readings\_width** = width

The width of the decibel readings area.

Returns:

---

an [Integer](#)

- **db\_readings\_height**

- **db\_readings\_height** = height

The height of the decibel readings area.

Returns:

---

an [Integer](#)

- **freqscope\_left**

- **freqscope\_left = value**

The frequency analyzer for the left channel.

**Returns:**

---

an instance of [FreqScopeView](#)

- **freqscope\_right**

- **freqscope\_right = value**

The frequency analyzer for the right channel.

**Returns:**

---

an instance of [FreqScopeView](#)

- **freqscope\_left\_view**

- **freqscope\_left\_view = value**

A panel to which the frequency analyzer for the left channel is added. This also provides the color for the border behind the view.

**Returns:**

---

an instance of [QView](#)

- **freqscope\_right\_view**

- **freqscope\_right\_view = value**

A panel to which the frequency analyzer for the right channel is added. This also provides the color for the border behind the view.

**Returns:**

---

an instance of [QView](#)

```
- drawReadings (view, width, height, number_of_readings: 5, unit_suffix: "",
last_string: true, min_reading: 0, max_reading: 100,
direction: 'horizontal', shape: 'lin', reading_position: 'tail', line_color)
```

Adds a horizontal or vertical grid readings panel.

#### Arguments:

---

<b>view</b>	an instance of <a href="#">QUserView</a> .
<b>width</b>	The view's width.
<b>height</b>	The view's height.
<b>number_of_readings</b>	The number of numerical readings.
<b>unit_suffix</b>	The unit suffix (i.e. "Hz" or "dB").
<b>last_string</b>	A boolean indicating whether the last reading needs to be wrapped inside the visual area or not.
<b>min_reading</b>	The minimum reading.
<b>max_reading</b>	The maximum reading.
<b>direction</b>	The readings direction (i.e. <code>\horizontal</code> or <code>\vertical</code> ).
<b>shape</b>	The shape of the increment values (i.e. <code>\lin</code> , <code>\log</code> or <code>\exp</code> ).
<b>reading_position</b>	The position where the readings values occur (i.e. <code>\head</code> or <code>\tail</code> ).
<b>line_color</b>	The <a href="#">Color</a> of the gridlines.

#### Returns:

---

an instance of [QUserView](#)

#### Discussion:

---

Example:

```
s.boot // boot first!

// create a new UserView()
(
  z = QZ_AnalyzerView(); // allocate memory
  u = QUserView().alwaysOnTop_(true).background_(Color.black);
  u.bounds = Rect(200,200,200,40);
  u.front;
)

// add readings to the user view...
(
  z.drawReadings(
    view:u,
    width:150,
    height:25,
    last_string:false,
    line_color:Color.white
  ).refresh
)

u.close // close the user view
z.close // free memory
```

## - `formatFreqValue` (float, unit\_suffix: "", suffix: false)

Formats a musical frequency value.

### Arguments:

<b>float</b>	A float value receiver
<b>unit_suffix</b>	The unit suffix (e.g. "Hz" for frequency).
<b>suffix</b>	A boolean indicating whether the suffix is added to the string or not.

### Returns:

A formatted [String](#)

### Discussion:

Example:

```
s.boot // boot first!
z = QZ_AnalyzerView() // allocate memory
z.formatFreqValue(float:6821.359174, unit_suffix:"Hz", suffix:true) // returns "6.8kHz"
z.close // free memory
```

## - `refreshReadings` (index: 0)

Refreshes the analysis grid, implementing any new colors or dimension values that have been modified.

### Arguments:

**index** The index of the channel to be displayed following a refresh (i.e. [0] = left, [1] = right).

## Inherited instance methods

138 methods from [QView](#) ► [show](#)

25 methods from [QObject](#) ► [show](#)

315 methods from [Object](#) ► [show](#)

## Examples

```
s.boot // boot the server first!

// configure the window...
(
w = QWindow(
  "My Analyzer", Rect(400,200,550,235)
).alwaysOnTop_(true).background_(Color.black);

z = QZ_AnalyzerView(w.asView);

w.onClose = { z.close }; // frees the synth memory allocated
w.front;
)
```

```

// play sounds...
(
a = { SinOsc.ar(freq:330, mul:0.35) }.play;
b = { Pan2.ar( SinOsc.ar(freq:220), pos:1, level:0.35 ) }.play;
)

// change the button colors...
z.freqscope_settings_button.states = [["ANALYZER SETTINGS", Color.white, Color.blue]]
// change the title label color...
z.freqscope_label.stringColor = Color.white
// change the "L" button properties...
z.left_button.states = [["0", Color.cyan, Color.magenta.alpha_(0.25)]]
// change the "R" button properties...
z.right_button.states = [["1", Color.yellow, Color.magenta.alpha_(0.25)]]
// change the scope display colors...
(
z.wave_color_left = [ Color.cyan ];
z.wave_color_right = [ Color.yellow ];
)
// change the color of the grid readings...
z.readings_color = Color.white
// change the gridline colors...
(
z.grid_color_left = Color.green;
z.grid_color_right = Color.red;
)
// resize the window and view bounds and increase the number of readings...
(
w.bounds = Rect(200,500,835,492);
z.freq_readings_width = 700;
z.db_readings_height = 400;
z.number_of_freq_readings = 18;
z.number_of_db_readings = 17;
)

// free sounds...
(
a.free; a = nil;
b.free; b = nil;
)

w.close // close the window

```

---

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Classes  
 /QZ\_AnalyzerView.schelp  
 link::Classes/QZ\_AnalyzerView::  
 sc version: 3.7alpha0

## SuperCollider CLASSES (extension)

[Z\\_Library>GUI>Accessories](#), [Libraries>crucial>GUI](#), [GUI>Accessories](#)

# QZ\_ConfirmDialog

A customizable dialog window utility with user options.

Source: [/home/garethh/.local/share/SuperCollider/Extensions/Z\\_Library.d/Classes/GUIs.d/QtCollider.d/Dialogs.d/QZ\\_ConfirmDialog.d/QZ\\_ConfirmDialog\\_13.sc](#)  
 Inherits from: [QWindow](#) : [Object](#)

See also: [ModalDialog](#), [Sheet](#)

## Description

QZ\_ConfirmDialog displays a borderless [QWindow](#) which acts as a customizable dialog window utility. A variable number of buttons can be added to the dialog, along with corresponding action functions.

Example:

```
d = QZ_ConfirmDialog() // opens the dialog window
d.close                // closes the dialog window
```

## Class Methods

\* **new** (button\_strings, button\_string\_colors, name: "Confirm Dialog...", bounds, resizable: true, border: false, server, scroll: false)

Creates a new QZ\_ConfirmDialog instance.

### Arguments:

<b>button_strings</b>	An array of strings (i.e. one for each respective option button). The number of buttons created on the dialog window is determined by the size of this array.
<b>button_string_colors</b>	An array of string colors (i.e. one for each respective option button).
<b>name</b>	A String for the text that will be displayed in the title bar. The default is "Confirm Dialog...", however because the window has no border, the title bar is invisible in any case.
<b>bounds</b>	A <a href="#">Rect</a> specifying the position and size of the window. The size does not include the border and title bar. Position is measured from the bottom-left corner of the screen (this is different than <a href="#">View: -bounds</a> ).
<b>resizable</b>	A <a href="#">Boolean</a> indicating whether this window is resizable by the user. The default is <b>true</b> .
<b>border</b>	A <a href="#">Boolean</a> indicating whether this window has a border. Borderless windows have no title bar and thus can only be closed in code. The default is <b>false</b> .
<b>server</b>	This is a dummy argument which is here to provide compatibility with SwingOSC and has no effect.
<b>scroll</b>	A <a href="#">Boolean</a> indicating whether this window will add scrollbars if its contents exceed its bounds. If this is set to <b>true</b> , then <a href="#">View: -resize</a> settings will be ignored for contained views. The default is <b>false</b> .

## \* **server**

The server which the GUI is running on.

### Returns:

an instance of [Server](#)

## Inherited class methods

9 methods from [QWindow](#) ► [show](#)

9 methods from [Object](#) ► [show](#)

## Instance Methods

### - **message**

### - **message = value**

A panel containing the option message to the user.

### Returns:

an instance of [QStaticText](#)

### - **purple**

The default string color of the option message.

### Returns:

an instance of [Color](#)

### - **button\_string\_array**

### - **button\_string\_array = value**

An array of strings (i.e. one for each respective option button). Setting this value will not modify the button strings displayed on the dialog window; however, this can be done by setting the button string properties on the array of [QButtons](#) returned by the [button](#) method.

### Returns:

an [Array](#) containing a variable number of instances of the [String](#) class

### - **string\_color\_array**

### - **string\_color\_array = value**

An array of string colors (i.e. one for each respective option button). Setting this value will not modify the button string colors displayed on the dialog window; however, this can be done by setting the button color properties on the array of [QButtons](#) returned by the [button](#) method.

### Returns:

an [Array](#) containing a variable number of instances of the [Color](#) class

## - **button**

### - **button** = value

The array of option buttons on the dialog window.

#### Returns:

an [Array](#) containing a variable number of instances of the [QButton](#) class

## - **completion\_function**

### - **completion\_function** = value

An array of completion functions (i.e. one for each respective option button). If no action is required on a particular button then the corresponding function value should be set to **nil**.

The default completion function for each respective option button sends a message to the post window explaining how to configure the function and its args.

#### Returns:

an [Array](#) containing a variable number of instances of the [Function](#) class

## - **args**

### - **args** = value

An array of argument sets (i.e. one for each respective completion function). If no arguments are required on a particular completion function then the corresponding args value should be set to `[]` (i.e. an empty array).

#### Returns:

a 2-dimensional [Array](#) containing the sets of arguments that are passed to the completion function array specified above (i.e. [arg1, arg2, arg3, etc..])

## Inherited instance methods

37 methods from [QWindow](#) ► [show](#)

315 methods from [Object](#) ► [show](#)

## Examples

```
// default...
QZ_ConfirmDialog()

// example with 1 button...
(
d = QZ_ConfirmDialog( ["Done"], bounds:Rect(500,300,250,125) );
d.message.string = "The action has been performed!...";
"~"
)

// example with 3 buttons...
QZ_ConfirmDialog( ["Yes", "No", "Cancel"], bounds:Rect(500,300,280,150) )
```



```

// example with 5 buttons...
(
d = QZ_ConfirmDialog(
    button_strings:["Yes", "No", "Cancel", "Reboot", "Quit"],
    button_string_colors:[Color.blue, Color.magenta, Color.grey,
        Color.red, Color.green]);
d.message.align = \center;
"~"
)

// example with a completion function array...
(
d = QZ_ConfirmDialog( ["Post", "Cancel"], bounds:Rect(500,300,250,125));
d.message.string = "Send a post window message?...";
d.completion_function = [ { "A message has been sent!".postln }, nil ];
"~"
)

// example of a completion function array with args...
(

d = QZ_ConfirmDialog( ["Fruit", "Veg", "Salad"], bounds:Rect(500,300,280,125) );
d.message.string_("Choose an item type...").align_(\center);

d.args = [
    /* Fruit args... */ ["Apple","Orange","Banana"],
    /* Veg   args... */ ["Carrot","Parsnip"],
    /* Salad args... */ ["Lettuce","Tomato","Cucumber","Celery"]
];

3.do{ arg i;
    d.completion_function[i] = { |w,x,y,z|

        (" " ++ d.args[i].size ++ " \" " ++ d.button_string_array[i]
        ++ "\" args were supplied... ").postln;
        d.args[i].size.do{ arg j;
            d.args[i][j].postln
        }
    }
};
"~"
)

```

---

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Classes  
 /QZ\_ConfirmDialog.schelp  
 link::Classes/QZ\_ConfirmDialog::  
 sc version: 3.7alpha0

## SuperCollider CLASSES (extension)

[Z\\_Library>GUI>Interfaces](#), [GUI>Interfaces](#)

# QZ\_Meter

A stereo i/o amplitude meter window.

Source: [/home/garethh/.local/share/SuperCollider/Extensions/Z\\_Library.d/Classes/GUIs.d/QtCollider.d/Signal\\_Analysis.d/QZ\\_Meter.d/QZ\\_Meter\\_28.sc](#)  
 Inherits from: [QWindow](#) : [Object](#)

See also: [QZ\\_MeterView](#), [ServerMeterView](#), [ServerMeter](#)

## Description

QZ\_Meter displays a [QWindow](#) containing a [QZ\\_MeterView](#) stereo input/output amplitude meter.

Example:

```
s.boot           // boot the server first!
w = QZ_Meter()  // opens the meter window
w.close         // closes the window
```

## Class Methods

\* **new** (name: "Z\_Meter", bounds, resizable: false, border: true, server, scroll: true)

Creates a new QZ\_Meter instance.

### Arguments:

<b>name</b>	A String for the text that will be displayed in the title bar. The default is "Z_Meter".
<b>bounds</b>	A <a href="#">Rect</a> specifying position and size of the window. The size does not include the border and title bar. Position is measured from the bottom-left corner of the screen (this is different than <a href="#">View: -bounds</a> ).
<b>resizable</b>	A <a href="#">Boolean</a> indicating whether this window is resizable by the user. The default is <b>false</b> .
<b>border</b>	A <a href="#">Boolean</a> indicating whether this window has a border. Borderless windows have no title bar and thus can only be closed in code. The default is <b>true</b> .
<b>server</b>	This is a dummy argument which is here to provide compatibility with SwingOSC and has no effect.
<b>scroll</b>	A <a href="#">Boolean</a> indicating whether this window will add scrollbars if its contents exceed its bounds. If this is set to <b>true</b> , then <a href="#">View: -resize</a> settings will be ignored for contained views. The default is <b>true</b> .

## Inherited class methods

9 methods from [QWindow](#) ► [show](#)

9 methods from [Object](#) ► [show](#)

## Instance Methods

- [z\\_meter\\_view](#)

- `z_meter_view = value`

The i/o amplitude meter view which is contained within the [QWindow](#)

Returns:

an instance of [QZ\\_MeterView](#)

## Inherited instance methods

37 methods from [QWindow](#) ► [show](#)

315 methods from [Object](#) ► [show](#)

## Examples

```
s.boot // boot the server first!

w = QZ_Meter() // opens the meter window
w.bounds = Rect(600,250,150,150)
z = w.z_meter_view // creates a pointer to the window's QZ_MeterView() instance

// change the background color
z.background = Color.white;
// change the button properties...
z.master_settings_button.states = [{"MASTER SETTINGS", Color.yellow, Color.black}]
// change the input label properties...
z.in_label.string_("IN").stringColor_(Color.red).background_(Color.white);
// change the output label properties...
z.out_label.string_("OUT").stringColor_(Color.red).background_(Color.white);
// change the max label colors...
z.max_label.stringColor_(Color.blue).background_(Color.white);
// change the min label colors...
z.min_label.stringColor_(Color.blue).background_(Color.white);
// change the separator line color...
z.separator_line.background = Color.magenta

// play sounds...
(
a = { SinOsc.ar(freq:330, mul:0.35) }.play;
b = { Pan2.ar( SinOsc.ar(freq:220), pos:1, level:0.35 ) }.play;
)
// free sounds...
(
a.free; a = nil;
b.free; b = nil;
)

w.close // close the window
```

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Classes/QZ\_Meter.schelp  
link::Classes/QZ\_Meter::  
sc version: 3.7alpha0

# QZ\_MeterView

A stereo i/o amplitude meter view.

Source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/Classes/GUIs.d/QtCollider.d/Signal\_Analysis.d/QZ\_Meter.d/QZ\_Meter\_28.sc

Inherits from: [QView](#) : [QObject](#) : [Object](#)

See also: [QZ\\_Meter](#), [ServerMeterView](#), [ServerMeter](#)

## Description

QZ\_MeterView is a stereo input/output amplitude meter which extends [QView](#) and can be added as a cell into a [QWindow](#) or one of the window's child views. QZ\_MeterView also supports the implementation of [QZ\\_Meter](#).

## Class Methods

### \* **new** (parent, bounds)

Creates a new instance of QZ\_MeterView that can be passed as a child view to a [QView](#) or a [QWindow](#), effectively placing it within the parent's visual space. If there is a [decorator](#) installed on the parent, it will manage the position of the new QZ\_MeterView.

#### NOTE: Qt GUI:

The 'parent' argument may be omitted, in which case the view will be displayed as a window on its own, when [shown](#).

The 'bounds' argument may be omitted, in which case the view will be created with its preferred size at position (0,0).

If a parent is given and there is a layout installed on it, the layout will manage the position and size of this view and the 'bounds' argument will have no effect.

#### Arguments:

**parent** The instance of [QView](#) or [QWindow](#) that the new view will become a child of.

**bounds** A [Rect](#) or a [Point](#) describing the size and position of the new QZ\_MeterView. If a Point is given, its coordinates will denote the view's size, while the view's position will be (0,0). Position is measured relative to the parent's top-left corner.

#### Returns:

a new instance of QZ\_MeterView

### \* **server**

The server from which the meter synths are reading.

#### Returns:

an instance of [Server](#)

### \* **initSynthDefs**

Initiates a [SynthDef](#) used by this class at [Startup](#) as follows:

```

/*
    outputs a control-rate ugen from an input signal
    to a bus so that the signal's value can be read...
*/
SynthDef(\monoOutput, { |bus, channel|
    Out.kr( bus, Amplitude.kr( In.ar(channel) ) )
}).add;

```

## Inherited class methods

15 methods from QView ► [show](#)

16 methods from QObject ► [show](#)

9 methods from Object ► [show](#)

## Instance Methods

### - **input\_bus**

### - **input\_bus = value**

The busses which read the data values of the input channels.

#### Returns:

an [Array](#) containing two separate instances of [Bus](#) (i.e. [0] = left, [1] = right)

### - **output\_bus**

### - **output\_bus = value**

The busses which read the data values of the output channels.

#### Returns:

an [Array](#) containing two separate instances of [Bus](#) (i.e. [0] = left, [1] = right)

### - **input\_signal**

### - **input\_signal = value**

The synths which represent the data values of the input channels. These are control-rate synths which represent the audio-rate signal being sent to the [SoundIn](#) class.

#### Returns:

an [Array](#) containing two control-rate synths (i.e. [0] = left, [1] = right)

### - **output\_signal**

### - **output\_signal = value**

The synths which represent the data values of the output channels. The synths are generated via the [\monoOutput SynthDef](#) (a definition added by this class at [Startup](#) via the [initSynthDefs](#) class method).

#### Returns:

an [Array](#) containing two instances of [Synth](#) (i.e. [0] = left, [1] = right)

- **input\_level**

- **input\_level** = value

The GUI views showing the input signals.

**Returns:**

---

an [Array](#) containing two separate instances of [QLevelIndicator](#) (i.e. [0] = left, [1] = right)

- **output\_level**

- **output\_level** = value

The GUI views showing the output signals.

**Returns:**

---

an [Array](#) containing two separate instances of [QLevelIndicator](#) (i.e. [0] = left, [1] = right)

- **input\_level\_routine**

- **input\_level\_routine** = value

The routines for updating the input signal levels displayed by the GUI.

**Returns:**

---

an [Array](#) containing two separate instances of [Routine](#) (i.e. [0] = left, [1] = right)

- **output\_level\_routine**

- **output\_level\_routine** = value

The routines for updating the output signal levels displayed by the GUI.

**Returns:**

---

an [Array](#) containing two separate instances of [Routine](#) (i.e. [0] = left, [1] = right)

- **master\_settings\_button**

- **master\_settings\_button** = value

The "MASTER SETTINGS" button.

**Returns:**

---

an instance of [QButton](#)

- **in\_label**

- **in\_label** = value

The "Input" label.

**Returns:**

---

an instance of [QStaticText](#)

### - **out\_label**

#### - **out\_label** = value

The "Output" label.

#### Returns:

---

an instance of [QStaticText](#)

### - **max\_label**

#### - **max\_label** = value

The label for the maximum amplitude level.

#### Returns:

---

an instance of [QStaticText](#)

### - **min\_label**

#### - **min\_label** = value

The label for the minimum amplitude level.

#### Returns:

---

an instance of [QStaticText](#)

### - **separator\_line**

#### - **separator\_line** = value

The vertical bar which separates the input and output meters.

#### Returns:

---

an instance of [QView](#)

### - **freeUGens**

Frees the level indicator routines and ugens that have been created by the program. The signal synths will only be freed if they are playing at the point when this method is called (verified via a call to the [Node: - isPlaying](#) method).

### - **freeSynths**

Frees the signal synths, regardless of whether they are playing or not.

## Inherited instance methods

138 methods from [QView](#) ► [show](#)

25 methods from [QObject](#) ► [show](#)

315 methods from [Object](#) ► [show](#)

## Examples

---

```
s.boot // boot the server first!

// configure the window...
(
w = QWindow(
  "My Meter", Rect(600,250,150,350)
).alwaysOnTop_(true).background_(Color.grey);

z = QZ_MeterView(w.asView);

w.onClose = { // free the synth memory allocated...
  z.close;
  z.freeSynths;
};
w.front;
)

// change the background color
w.background = Color.white;
// change the button properties...
z.master_settings_button.states = [{"MASTER SETTINGS", Color.yellow, Color.black}]
// change the input label properties...
z.in_label.string_("INPUT").stringColor_(Color.red).background_(Color.white);
// change the output label properties...
z.out_label.string_("OUTPUT").stringColor_(Color.red).background_(Color.white);
// change the max label colors...
z.max_label.stringColor_(Color.blue).background_(Color.white);
// change the min label colors...
z.min_label.stringColor_(Color.blue).background_(Color.white);
// change the separator line color...
z.separator_line.background = Color.magenta

// play sounds...
(
a = { SinOsc.ar(freq:330, mul:0.35) }.play;
b = { Pan2.ar( SinOsc.ar(freq:220), pos:1, level:0.35 ) }.play;
)
// free sounds...
(
a.free; a = nil;
b.free; b = nil;
)

w.close // close the window
```

---

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Classes/QZ\_MeterView.schelp  
link::Classes/QZ\_MeterView::  
sc version: 3.7alpha0





## SuperCollider CLASSES (extension)

[Z\\_Library>GUI>Interfaces](#), [GUI>Interfaces](#)

# QZ\_Scope

A stereo oscilloscope window.

Source: [/home/garethh/.local/share/SuperCollider/Extensions/Z\\_Library.d/Classes/GUIs.d/QtCollider.d/Signal\\_Analysis.d/QZ\\_Scope.d/QZ\\_Scope\\_015.sc](#)  
 Inherits from: [QWindow](#) : [Object](#)

See also: [QZ\\_ScopeView](#), [ScopeView](#), [QScope2](#), [Stethoscope](#), [QStethoscope2](#)

## Description

QZ\_Scope displays a [QWindow](#) containing a [QZ\\_ScopeView](#) stereo oscilloscope.

Example:

```
s.boot           // boot the server first!
w = QZ_Scope()  // opens the oscilloscope window
w.close         // closes the window
```

## Class Methods

\* **new** (name: "Z\_Scope", bounds, resizable: true, border: true, server, scroll: false)

Creates a new QZ\_Scope instance.

### Arguments:

<b>name</b>	A String for the text that will be displayed in the title bar. The default is "Z_Scope".
<b>bounds</b>	A <a href="#">Rect</a> specifying position and size of the window. The size does not include the border and title bar. Position is measured from the bottom-left corner of the screen (this is different than <a href="#">View: -bounds</a> ).
<b>resizable</b>	A <a href="#">Boolean</a> indicating whether this window is resizable by the user. The default is <b>true</b> .
<b>border</b>	A <a href="#">Boolean</a> indicating whether this window has a border. Borderless windows have no title bar and thus can only be closed in code. The default is <b>true</b> .
<b>server</b>	This is a dummy argument which is here to provide compatibility with SwingOSC and has no effect.
<b>scroll</b>	A <a href="#">Boolean</a> indicating whether this window will add scrollbars if its contents exceed its bounds. If this is set to <b>true</b> , then <a href="#">View: -resize</a> settings will be ignored for contained views. The default is <b>false</b> .

## Inherited class methods

9 methods from [QWindow](#) ► [show](#)

9 methods from [Object](#) ► [show](#)

## Instance Methods

- [z\\_scope\\_view](#)

- **z\_scope\_view** = value

The oscilloscope view which is contained within the [QWindow](#)

Returns:

an instance of [QZ\\_ScopeView](#)

## Inherited instance methods

37 methods from [QWindow](#) ► [show](#)

315 methods from [Object](#) ► [show](#)

## Examples

```
s.boot // boot the server first!

w = QZ_Scope() // opens the oscilloscope window

z = w.z_scope_view // creates a pointer to the window's QZ_ScopeView() instance

// change the button colors...
z.scope_settings_button.states = [ ["SCOPE SETTINGS", Color.white, Color.blue]]
// change the title label color...
z.scope_label.stringColor = Color.white
// change the "L" label color...
z.left_label.stringColor = Color.magenta
// change the "R" label color...
z.right_label.stringColor = Color.green
// change the scope display colors...
z.scope_display.waveColors = [ Color.magenta, Color.green ]

// play sounds...
(
a = { SinOsc.ar(freq:330, mul:0.35) }.play;
b = { Pan2.ar( SinOsc.ar(freq:220), pos:1, level:0.35 ) }.play;
)
// free sounds...
(
a.free; a = nil;
b.free; b = nil;
)

w.close // close the window
```

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Classes/QZ\_Scope.schelp  
link::Classes/QZ\_Scope::  
sc version: 3.7alpha0

## SuperCollider CLASSES (extension)

Z\_Library>GUI>Views, GUI>Views

# QZ\_ScopeView

A stereo buffer plotting view.

Source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/Classes/GUIs.d/QtCollider.d/Signal\_Analysis.d/QZ\_Scope.d/QZ\_Scope\_015.sc

Inherits from: [QView](#) : [QObject](#) : [Object](#)

See also: [QZ\\_Scope](#), [Stethoscope](#), [QStethoscope2](#), [ScopeView](#), [QScope2](#)

## Description

QZ\_ScopeView is a stereo oscilloscope which extends [QView](#) and can be added as a cell into a [QWindow](#) or one of the window's child views. QZ\_ScopeView also supports the implementation of [QZ\\_Scope](#).

## Class Methods

\* **new** (parent, bounds)

Creates a new instance of QZ\_ScopeView that can be passed as a child view to a [QView](#) or a [QWindow](#), effectively placing it within the parent's visual space. If there is a [decorator](#) installed on the parent, it will manage the position of the new QZ\_ScopeView.

### NOTE: Qt GUI:

The 'parent' argument may be omitted, in which case the view will be displayed as a window on its own, when [shown](#).

The 'bounds' argument may be omitted, in which case the view will be created with its preferred size at position (0,0).

If a parent is given and there is a layout installed on it, the layout will manage the position and size of this view and the 'bounds' argument will have no effect.

### Arguments:

**parent** The instance of [QView](#) or [QWindow](#) that the new view will become a child of.

**bounds** A [Rect](#) or a [Point](#) describing the size and position of the new QZ\_ScopeView. If a Point is given, its coordinates will denote the view's size, while the view's position will be (0,0). Position is measured relative to the parent's top-left corner.

### Returns:

a new instance of QZ\_ScopeView

\* **server**

The server from which the scope synths are reading.

### Returns:

an instance of [Server](#)

## \* `initSynthDefs`

Initiates a `SynthDef` used by this class at `Startup` as follows:

```
/*
  reads values from an audio bus, using ScopeOut2
  to write it to a ScopeBuffer...
*/
SynthDef(\monoscope, { arg bus = 0, bufnum, zoom = 1.0;
  var input;
  input = In.ar(bus, 2);
  // ScopeOut2 writes the audio to the buffer
  ScopeOut2.ar( input, bufnum, 4096, 1024.0/zoom )
}).add
```

## Inherited class methods

15 methods from `QView` ► [show](#)

16 methods from `QObject` ► [show](#)

9 methods from `Object` ► [show](#)

## Instance Methods

### - `scope_synth`

### - `scope_synth = value`

The scope synth which reads the data values. The synth is generated via the `\monoscope SynthDef` (a definition added by this class at `Startup` via the `initSynthDefs` class method)

Returns:

an instance of `Synth`

### - `scope_buffer`

### - `scope_buffer = value`

The scope buffer used by the `\monoscope` synth.

Returns:

an instance of `ScopeBuffer`

### - `scope_settings_button`

### - `scope_settings_button = value`

The "SCOPE SETTINGS" button

Returns:

an instance of `QPushButton`

### - `scope_label`

- **scope\_label** = value

The title label

**Returns:**

---

an instance of [QStaticText](#)

- **left\_label**

- **left\_label** = value

The "L" label

**Returns:**

---

an instance of [QStaticText](#)

- **right\_label**

- **right\_label** = value

The "R" label

**Returns:**

---

an instance of [QStaticText](#)

- **scope\_display**

- **scope\_display** = value

A 2-channel Qt-language oscilloscope display. Its wave colors can be modified by evaluating ...

scope\_display.waveColors = [ color1, color2 ]

**Returns:**

---

an instance of [QScope2](#)

- **xZoom**

- **xZoom** = float: 1

Gets or sets the scaling factor on the horizontal axis.

**Returns:**

---

a [Float](#)

- **yZoom**

- **yZoom** = float: 1

Gets or sets the scaling factor on the vertical axis.

**Returns:**a [Float](#)**freeScopeSynth**

Frees the scope synth, regardless of whether it is playing or not.

## Inherited instance methods

138 methods from [QView](#) ► [show](#)25 methods from [QObject](#) ► [show](#)315 methods from [Object](#) ► [show](#)

## Examples

```
s.boot // boot the server first!

// configure the window...
(
  w = QWindow(
    "My Scope", Rect(600,250,300,300)
  ).alwaysOnTop_(true).background_(Color.black);

  z = QZ_ScopeView(w.asView);

  w.onClose = { // free the synth memory allocated...
    z.close;
    z.freeScopeSynth;
  };
  w.front;
)

// change the button colors...
z.scope_settings_button.states = [{"SCOPE SETTINGS", Color.white, Color.blue}]
// change the title label color...
z.scope_label.stringColor = Color.white
// change the "L" label color...
z.left_label.stringColor = Color.magenta
// change the "R" label color...
z.right_label.stringColor = Color.green
// change the scope display colors...
z.scope_display.waveColors = [ Color.magenta, Color.green ]

// play sounds...
(
  a = { SinOsc.ar(freq:330, mul:0.35) }.play;
  b = { Pan2.ar( SinOsc.ar(freq:220), pos:1, level:0.35 ) }.play;
)
// free sounds...
(
  a.free; a = nil;
  b.free; b = nil;
)

w.close // close the window
```

---

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Classes/QZ\_ScopeView.schelp  
link::Classes/QZ\_ScopeView::  
sc version: 3.7alpha0





## SuperCollider CLASSES (extension)

[Z\\_Library>GUI>Interfaces](#), [GUI>Interfaces](#), [Libraries>crucial>GUI](#)

# QZ\_ServerLevelsPanel

A window indicating the current time and the CPU usage levels on a SuperCollider server.

Source: [/home/garethh/.local/share/SuperCollider/Extensions/Z\\_Library.d/Classes/GUIs.d/QtCollider.d/Server.d/QZ\\_ServerLevelsPanel.d/QZ\\_ServerLevelsPanel\\_16.sc](#)  
Inherits from: [QWindow](#) : [Object](#)

See also: [QZ\\_ServerLevelsPanelView](#), [Server](#)

## Description

QZ\_ServerLevelsPanel displays a [QWindow](#) containing a [QZ\\_ServerLevelsPanelView](#), which indicates the current time and the CPU usage levels on a SuperCollider Server.

Example:

```
s.boot // boot the server first!
w = QZ_ServerLevelsPanel() // opens the server levels window
w.close // closes the window
```

## Class Methods

\* **new** (name: "Z\_ServerLevelsPanel", bounds, resizable: false, border: true, server, scroll: false)

Creates a new QZ\_ServerLevelsPanel instance.

### Arguments:

<b>name</b>	A String for the text that will be displayed in the title bar. The default is "Z_ServerLevelsPanel".
<b>bounds</b>	A <a href="#">Rect</a> specifying position and size of the window. The size does not include the border and title bar. Position is measured from the bottom-left corner of the screen (this is different than <a href="#">View: -bounds</a> ).
<b>resizable</b>	A <a href="#">Boolean</a> indicating whether this window is resizable by the user. The default is <b>false</b> .
<b>border</b>	A <a href="#">Boolean</a> indicating whether this window has a border. Borderless windows have no title bar and thus can only be closed in code. The default is <b>true</b> .
<b>server</b>	This is a dummy argument which is here to provide compatibility with SwingOSC and has no effect.
<b>scroll</b>	A <a href="#">Boolean</a> indicating whether this window will add scrollbars if its contents exceed its bounds. If this is set to <b>true</b> , then <a href="#">View: -resize</a> settings will be ignored for contained views. The default is <b>false</b> .

## Inherited class methods

9 methods from [QWindow](#) ► [show](#)

9 methods from [Object](#) ► [show](#)

## Instance Methods

- `z_server_levels_panel_view`
- `z_server_levels_panel_view = value`

The server levels panel view which is contained within the [QWindow](#)

#### Returns:

an instance of [QZ\\_ServerLevelsPanelView](#)

## Inherited instance methods

37 methods from [QWindow](#) ► [show](#)

315 methods from [Object](#) ► [show](#)

## Examples

```
s.boot // boot the server first!

w = QZ_ServerLevelsPanel() // opens the server levels window

// create a pointer to the window's QZ_ServerLevelsPanelView() instance...
z = w.z_server_levels_panel_view

// change the background color...
w.background = Color.green
// change the time stamp color...
z.time_label.stringColor = Color.red
// change the "server" label colors...
z.server_label.background_(Color.blue).stringColor_(Color.white)
// change the server levels string color...
(
  l = z.level_label;
  l.size.do { arg i; l[i].stringColor = Color.magenta }
)

w.close // close the window
```

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Classes/  
 /QZ\_ServerLevelsPanel.schelp  
 link::Classes/QZ\_ServerLevelsPanel::  
 sc version: 3.7alpha0

# QZ\_ServerLevelsPanelView

A view indicating the current time and the CPU usage levels on a SuperCollider server.

Source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/Classes/GUIs.d/QtCollider.d/Server.d/QZ\_ServerLevelsPanel.d/QZ\_ServerLevelsPanel\_16.sc

Inherits from: [QView](#) : [QObject](#) : [Object](#)

See also: [QZ\\_ServerLevelsPanel](#), [Server](#)

## Description

[QZ\\_ServerLevelsPanelView](#) is a view indicating the current time and the CPU usage levels on a SuperCollider [Server](#). The class extends [QView](#) and can be added as a cell into a [QWindow](#) or one of the window's child views. [QZ\\_ServerLevelsPanelView](#) also supports the implementation of [QZ\\_ServerLevelsPanel](#).

## Class Methods

### \* [new](#) (parent, bounds)

Creates a new instance of [QZ\\_ServerLevelsPanelView](#) that can be passed as a child view to a [QView](#) or a [QWindow](#), effectively placing it within the parent's visual space. If there is a [decorator](#) installed on the parent, it will manage the position of the new [QZ\\_ServerLevelsPanelView](#).

#### NOTE: Qt GUI:

The 'parent' argument may be omitted, in which case the view will be displayed as a window on its own, when [shown](#).

The 'bounds' argument may be omitted, in which case the view will be created with its preferred size at position (0,0).

If a parent is given and there is a layout installed on it, the layout will manage the position and size of this view and the 'bounds' argument will have no effect.

#### Arguments:

**parent** The instance of [QView](#) or [QWindow](#) that the new view will become a child of.

**bounds** A [Rect](#) or a [Point](#) describing the size and position of the new [QZ\\_ServerLevelsPanelView](#). If a [Point](#) is given, its coordinates will denote the view's size, while the view's position will be (0,0). Position is measured relative to the parent's top-left corner.

#### Returns:

a new instance of [QZ\\_ServerLevelsPanelView](#)

### \* [server](#)

The server from which the GUI is reading.

#### Returns:

an instance of [Server](#)

## Inherited class methods

15 methods from [QView](#) ► [show](#)

16 methods from [QObject](#) ► [show](#)

9 methods from [Object](#) ► [show](#)

## Instance Methods

---

### - `hour_stamp`

Returns the current time in "`hh:mm:ss`" format.

**Returns:**

---

a [String](#)

### - `time_label`

### - `time_label = value`

The label displaying the current time.

**Returns:**

---

an instance of [QStaticText](#)

### - `server_label`

### - `server_label = value`

The "Server" label.

**Returns:**

---

an instance of [QStaticText](#)

### - `level_label`

### - `level_label = value`

The labels displaying the server values.

**Returns:**

---

an [Array](#) containing six separate instances of [QStaticText](#) as follows:

- [0] = Peak CPU usage
- [1] = Average CPU usage
- [2] = Number of running [UGens](#)
- [3] = Number of running [Synths](#)
- [4] = Number of [Groups](#)
- [5] = Number of loaded [SynthDefs](#)

### - `routine`

### - `routine = value`

The routine used for updating the server values.

**Returns:**

---

an instance of [Routine](#)

### - `tempo_clock`

## - `tempo_clock = value`

The scheduler used for running the routine which updates the server values.

### Returns:

an instance of [TempoClock](#)

## - `setTimeUnit (value)`

Adds a leading zero to any positive [Integer](#) which is  $\geq 0$  &&  $< 10$

### Returns:

an instance of [String](#)

### Discussion:

Example:

```
s.boot // boot the server first!
z = QZ_ServerLevelsPanelView().front // allocate memory
z.setTimeUnit ( 3 )                  // returns "03"
z.close                             // free the memory
```

## - `setFloatUnit (float)`

Adds following zeros to any [Float](#) value which rounds with less than two decimal places.

### Returns:

an instance of [String](#)

### Discussion:

Example:

```
s.boot // boot the server first!
z = QZ_ServerLevelsPanelView().front // allocate memory
z.setFloatUnit( 27 )                 // returns "27.00"
z.close                             // free the memory
```

## Inherited instance methods

138 methods from [QView](#) ► [show](#)

25 methods from [QObject](#) ► [show](#)

315 methods from [Object](#) ► [show](#)

## Examples

```
s.boot // boot the server first!

// configure the window...
(
w = QWindow(
  "My Server Panel", Rect(600,425,580,60)
).alwaysOnTop_(true).background_(Color.black);

z = QZ_ServerLevelsPanelView(w.asView);

w.onClose = { z.close }; // frees the routine memory allocated
w.front;
)
```

```
// change the time stamp color...
z.time_label.stringColor = Color.yellow
// change the "server" label colors...
z.server_label.background_(Color.blue).stringColor_(Color.white)
// change the server levels string color...
(
  l = z.level_label;
  l.size.do { arg i; l[i].stringColor = Color.magenta }
)

w.close // close the window
```

---

source: /home/garethh/.local/share/SuperCollider/Extensions/Z\_Library.d/HelpSource/Classes/QZ\_ServerLevelsPanelView.schelp  
link::Classes/QZ\_ServerLevelsPanelView::  
sc version: 3.7alpha0

```

CZ_EternalOwlCall {
    classvar <server, <>class_zpi_pointer;
    classvar <>valid_instance_count, <>invalid_instance_count;
    classvar <>allow_multiple_instances;

    /***** GENERAL PRIVATE INSTANCE VARIABLES, OBJECTS & FUNCTIONS *****/

    var number_of_subs; // number of subpatches in each patch
    var mode; // can be set to "Performance" or "Testing"
    var subpatch_flag; // 2D Array to store a flag for each subpatch
    // 2D Arrays to store functions which start or free each subpatch...
    var subpatchAction, freeSubpatch;
    var tempo_clock;
    // Variables to Enable Automatic Disk Recording:
    var record, record_start_value; // either both true or both false
    var already_recording;
    var diskout_node, diskin_node;
    var record_dpath; // references the session recording directory
    var record_path; // references the current session recording
    var diskout_buffer, diskin_buffer; // allocate disk i/o buffers
    // Group Nodes:
    var p1_NSG_group_node, p3_RG_group_node, p3_NSG_group_node;
    var p2; // 2D Array to store all synths in patch #2
    // Microphone Input Signal Synths:
    var audio_direct, audio_reverb;
    // Effects Processing Synths:
    var s_reverb, reverb, comb, tap;
    // Effects Processing Buses:
    var s_reverb_bus, reverb_bus, comb_bus, tap_bus;
    // Effects Processing Ints, Floats & Arrays:
    var pan_array;

    /***** PRIVATE VARIABLES FOR PATCH #1 & #3 (NatureSoundsGroupChannel) *****/

    var audio_dpath; // References the directory containing the audio files
    // Buffers to reference the hard-disk location of the audio files:
    var tawny_owl, pigmy_owl, otter_CostaRica, frog_Amazon, frog_Peru2, mosquito;
    // pigmy_owl = stereo & mosquito = stereo, all other files are mono
    // PlayBuf Synths for the audio files:
    var audition_node, tawny_owl_player, pigmy_owl_player, otter_CostaRica_player;
    var frog_Amazon_player, frog_Peru2_player, mosquito_player;
    // Synths for doppler effects:
    var otter_CostaRica_s_doppler, mosquito_s_doppler;
    var otter_CostaRica_f_doppler, mosquito_f_doppler;
    // Owl Harmonizer Variables:
    var fundamental_shift_rate;
    // Synth Arrays for owl harmonizer:
    var owl_sample_shifter, owl_formant_shifter;
    // Float Value Arrays for owl harmonizer:
    var formant_shift_rate_array, sampler_pan_array, formant_pan_array;
    // Infintie loop/schedule routines for the owl harmonizer:
    var owl_harmonizer_routine, freq_modulation_routine, pan_modulation_routine;
    // Fade-out routines:
    var p1_release_routine, p3_NSG_xfade_routine;

    /***** PRIVATE VARIABLES FOR PATCH #3 (ReverbGroupChannel) *****/

    var mid_mod_pulse, low_mod_pulse, bass_pulse;

    /***** GENERAL PUBLIC INSTANCE VARIABLES, OBJECTS & FUNCTIONS *****/

```



```

var <>valid_instance;
// GUI Objects...
var <>gui, gui_meter, gui_scope;
// Public variables for sending values to other classes:
var <>next_patch, <>next_subpatch, <>last_patch, <>last_subpatch;
var <dpath; // root path of the file directories

// Public functions to be sent as values by other classes:
// Recording functions:
var <startRecording, <stopRecording, <playLastRecordedSession;
// Functions for changing the effects sent to the microphone channel:
var <directMicrophoneSignal, <sendMicToBus_Reverb;
var <sendMicToBus_Reverb__7TapDelay, <sendMicToBus_Reverb_12TapDelay;
var <sendMicToBus_Reverb_12TapDelay_Comb;
// Sample & effect auditioning functions:
var <playSample, <playDoppler, <freeAudition;
// Modulation setting functions:
var <setFreqModulation, <setPanModulation;
var <freeFreqModulation, <freePanModulation;

*initClass {
  StartUp.add {
    server = Server.default;
    valid_instance_count = 0;
    invalid_instance_count = 0;
    allow_multiple_instances = false;
    this.initSynthDefs;
  }
}

*new { ^super.new.initCZ_EternalOwlCall }

*initSynthDefs {
  // allocate functions which are used to the create the synthdefs first...

  /***** GENERAL LOCAL EFFECTS PROCESSING FUNCTIONS *****/

  var stereoReverbChannel = {
    arg out_bus = 0, in_bus, amp = 1;
    var input = In.ar(bus:in_bus, numChannels:2);

    input = AllpassC.ar(
      in:input,
      maxdelaytime:0.04,
      delaytime:0.1,
      decaytime:3);

    Out.ar(out_bus, input * amp);
  };

  // Sustains audio input signal using a long atmospheric reverb algorithm:
  var atmosphericReverbChannel = {
    arg out_bus = 0, in_bus, delay_bus, comb_bus,
    fade_in = 0, gate = 1, // passed gate=0 value will fade out
    release_time = 8, delaytime, pan = 0, time_offset = 0,
    next_tap = 0, next_comb = 0;

    var input = In.ar(bus:in_bus, numChannels:1);
    var env;

    input = AllpassC.ar(in:AllpassC.ar( input,

```

```

        maxdelaytime:0.2, delaytime:delaytime, decaytime:50),
maxdelaytime:0.2, delaytime:delaytime, decaytime:1000);

// Add compressor/gate to balance & sustain the signal:
input = CompanderD.ar(input, input,
    thresh: /*0.1175*/ 0.8,
    slopeBelow: 0.8,
    slopeAbove: 1,
    clampTime: 0.01,
    relaxTime: 0.01) * 0.1;

input = input * Line.kr(0, 1, fade_in);
env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:2);

Out.ar(out_bus, Pan2.ar(input, pos:pan, level:1.25) * env);
Out.ar(delay_bus, input * env * next_tap );
Out.ar(comb_bus, input * env * next_comb);
};

// Adds a chain of delay taps:
var delayTap = {
    arg out_bus = 0, in_bus, tap_bus, comb_bus, delaytime, pan,
    level = 0.5, next_tap = 0;

    var input = In.ar(bus:in_bus, numChannels:1);

    input = DelayC.ar(in:input,
        maxdelaytime:delaytime, delaytime:delaytime);

    Out.ar(out_bus , Pan2.ar(input, pos:pan, level:level /*0.25*/));
    Out.ar(tap_bus, input * next_tap);
    // Out.ar(comb_bus, input);
};

var combChannel = {
    arg out_bus = 0, in_bus, reverb_bus, delay_bus,
    fade_in = 0, gate = 1, // passed gate=0 value will fade out
    release_time = 8, delaytime = 1,
    control_freq = 0, control_mul = 0.00249, control_add = 0.0025,
    pan = 0;

    var input = In.ar(bus:in_bus, numChannels:1);
    var env;

    delaytime = Saw.kr(freq:control_freq, /*iphase:0,*/
        mul:control_mul, add:control_add);

    input = CombC.ar(in:input, maxdelaytime:0.01,
        delaytime: delaytime,
        decaytime:4, mul:/*0.0325*/ 0.0625);
    input = BHiPass.ar(in:input, freq:250, rq:0.65);
    input = input * Line.kr(0, 1, fade_in);

    env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
        gate:gate, doneAction:2);

    // comb channel is sent back to the reverb channel
    // to create a feedback loop:
    Out.ar(out_bus, Pan2.ar(input, pos:pan, level:0.65) * env);
    Out.ar(reverb_bus, input * env);

```

```

};

/***** LOCAL FUNCTIONS FOR PATCH #1 & #3 (NatureSoundsGroupChannel) *****/

var monoSamplePlayer = {
  arg out_bus = 0, bufnum = 0, amp = 1, pan = 0, fade_in = 0,
  gate = 1, /*passed gate=0 value will fade out*/ release_time = 8,
  reverb_bus, direct = 0;
  var env;
  var input = PlayBuf.ar(numChannels:1, bufnum:bufnum, loop:1);

  input = input * Line.kr(0, 1, fade_in);
  env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:2);

  // Out.ar(out_bus,    input * amp * direct    * env);
  Out.ar(reverb_bus, input * amp * (1-direct) * env);
};

var monoToStereoSamplePlayer = {
  arg out_bus = 0, bufnum = 0, start_pos = 0,
  amp = 1, pan = 0, fade_in = 0,
  gate = 1, /*passed gate=0 value will fade out*/ release_time = 8,
  reverb_bus, direct = 0.5;
  var env;
  var input = PlayBuf.ar(numChannels:1, bufnum:bufnum,
    startPos:start_pos, loop:1);

  input = Pan2.ar(input, pos:pan, level:1);
  input = input * Line.kr(0, 1, fade_in);
  env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:2);

  Out.ar(out_bus,    input * amp * direct    * env);
  Out.ar(reverb_bus, input * amp * (1-direct) * env);
};

var stereoSamplePlayer = {
  arg out = 0, bufnum = 0, amp = 1, fade_in = 0,
  gate = 1, /*passed gate=0 value will fade out*/ release_time = 8;
  var env;
  var input = PlayBuf.ar(numChannels:2, bufnum:bufnum, loop:1);

  input = input * Line.kr(0, 1, fade_in);
  env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:2);

  Out.ar(out, input * amp * env)
};

var monoCircularSampleDoppler = {
  arg out = 0, bufnum = 0, amp = 1, fade_in = 0,
  gate = 1, /*passed gate=0 value will fade out*/ release_time = 8,
  rate = 1;
  var input, env;

  input = Pan2.ar(in:PlayBuf.ar(
    numChannels:1, // mono audio input
    bufnum:bufnum,
    rate:LFTri.kr(rate, mul:1/3, add:1), // sample rate
    loop:1),

```

```

pos: SinOsc.kr(rate, phase:1.5pi, mul:1, add:0) ); // clockwise rotation

input = input * Line.kr(0, 1, fade_in);
env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
  gate:gate, doneAction:2);

Out.ar(out, input * amp * env)
};

var stereoTriangularSampleDoppler = {
  arg out = 0, bufnum = 0, amp = 1, fade_in = 0,
  gate = 1, /*passed gate=0 value will fade out*/ release_time = 8,
  rate = 1;
  var input, env;

  input = PlayBuf.ar(
    numChannels:2, // stereo audio input
    bufnum:bufnum,
    rate:LFTri.kr(/*rate*/ SinOsc.kr(rate, mul:4, add:4),
      mul:/*1.9*/ 0.25, add:/*2.1*/ 1), // sample rate
    loop:1);
  Balance2.ar(input[0], input[1],
    pos: LFTri.kr(
      rate, iphase:3, mul:1, add:0) * -1, // anticlockwise rotation
    level:0.75);
  input = BHiPass.ar(in:input, freq:80, rq:0.8);

  input = input * Line.kr(0, 1, fade_in);
  env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:2);

  Out.ar(out, input * amp * env)
};

var monoCircularFormantDoppler = {
  arg out = 0, bufnum = 0, amp = 1, fade_in = 0,
  gate = 1, /*passed gate=0 value will fade out*/ release_time = 8,
  rate = 1;
  var input, env;

  input = Pan2.ar(in: PitchShift.ar(
    in:PlayBuf.ar(numChannels:1, bufnum:bufnum, loop:1);,
    windowSize:0.2, // grain size
    pitchRatio:LFTri.kr(rate, mul:1/3, add:1), // pitch shift ratio
    pitchDispersion:0.1,
    timeDispersion:0.004),
    pos: SinOsc.kr(rate, phase:1.5pi, mul:1, add:0) ); // clockwise rotation

  input = input * Line.kr(0, 1, fade_in);
  env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:2);

  Out.ar(out, input * amp * env)
};

var stereoTriangularFormantDoppler = {
  arg out = 0, bufnum = 0, amp = 1, fade_in = 0,
  gate = 1, /*passed gate=0 value will fade out*/ release_time = 8,
  rate = 1;
  var input, env;

```

```

input = PitchShift.ar(
    in:PlayBuf.ar(numChannels:2, bufnum:bufnum, loop:1);,
    windowSize:0.1, // grain size
    pitchRatio:LFTri.kr(rate, mul:1.9, add:2.1), // pitch shift ratio
    pitchDispersion:0,
    timeDispersion:0.004);
input = Balance2.ar(input[0], input[1],
    pos: LFTri.kr(rate, iphase:3, mul:1, add:0) * -1,
    /* anticlockwise rotation */ level:2);
input = BHiPass.ar(in:input, freq:80, rq:0.8);

input = input * Line.kr(0, 1, fade_in);
env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:2);

Out.ar(out, input * amp * env)
};

var monoSampleShift = {
    arg out_bus = 0, bufnum = 0, amp = 1, pan = 0, fade_in = 0,
    gate = 1, /*passed gate=0 value will fade out*/ release_time = 8,
    pitch_variation_freq = 0, pitch_variation_amp = 0,
    pitch_rate = 1, amp_mod_cycle_freq = 0,
    direction = 1, /*-1 to reverse*/ lowcut = 60, rq = 1,
    reverb_bus, direct = 0.5, done_action = 2;

    var env;
    var input = PlayBuf.ar(numChannels:1, bufnum:bufnum,
        rate:LFTri.kr(
            freq:pitch_variation_freq,
            mul:LFTri.kr(
                freq:amp_mod_cycle_freq,
                iphase:3,
                mul:pitch_variation_amp/2,
                add:pitch_variation_amp/2),
            add:pitch_rate * direction),
        loop:1);

    input = BHiPass.ar(in:input, freq:lowcut, rq:rq);
    input = input * Line.kr(0, 1, fade_in);
    env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
        gate:gate, doneAction:done_action);

    // Out.ar(out_bus, input * amp * direct * env);
    Out.ar(reverb_bus, input * amp * (1-direct) * env);
};

var monoToStereoSampleShift = {
    arg out_bus = 0, bufnum = 0, amp = 1, pan = 0, fade_in = 0,
    gate = 1, /*passed gate=0 value will fade out*/ release_time = 8,
    pitch_variation_freq = 0, pitch_variation_amp = 0,
    pitch_rate = 1, amp_mod_cycle_freq = 0,
    pan_variation_freq = 0, pan_variation_iphase = 0, pan_variation_amp = 0,
    direction = 1, /*-1 to reverse*/ lowcut = 60, rq = 1,
    reverb_bus, direct = 0.5, done_action = 2;

    var env;
    var input = PlayBuf.ar(numChannels:1, bufnum:bufnum,
        rate:LFTri.kr(
            freq:pitch_variation_freq,
            mul:LFTri.kr(

```

```

        freq:amp_mod_cycle_freq,
        iphase:3,
        mul:pitch_variation_amp/2,
        add:pitch_variation_amp/2),
    add:pitch_rate * direction),
    loop:1);

input = Pan2.ar(
    in:input,
    pos:LFTri.kr(
        freq: pan_variation_freq,
        iphase:pan_variation_iphase,
        mul: pan_variation_amp,
        add: pan),
    level:1);
input = BHiPass.ar(in:input, freq:lowcut, rq:rq);

input = input * Line.kr(0, 1, fade_in);
env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:done_action);

Out.ar(out_bus, input * amp * direct * env);
Out.ar(reverb_bus, input * amp * (1-direct) * env);
};

var monoToStereoFormantShift = {
    arg out_bus = 0, bufnum = 0, amp = 1, pan = 0, fade_in = 0,
    gate = 1, /*passed gate=0 value will fade out*/ release_time = 8,
    sample_rate = 1, pitch_variation_freq = 0, pitch_variation_amp = 0,
    pitch_rate = 1, amp_mod_cycle_freq = 0,
    pan_variation_freq = 0, pan_variation_iphase = 0, pan_variation_amp = 0,
    direction = 1, /*-1 to reverse*/ lowcut = 60, rq = 1,
    reverb_bus, direct = 0.5, done_action = 2;

    var env;
    var input = PlayBuf.ar(numChannels:1, bufnum:bufnum,
        rate:LFSaw.kr(
            freq:pitch_variation_freq,
            mul:LFTri.kr(
                freq:amp_mod_cycle_freq,
                iphase:3,
                mul:pitch_variation_amp/2,
                add:pitch_variation_amp/2),
            add:sample_rate * direction),
        loop:1);

    input = PitchShift.ar(in:input,
        windowSize:0.2, // grain size
        pitchRatio:pitch_rate, // pitch shift ratio
        pitchDispersion:0,
        timeDispersion:0.004);

    input = Pan2.ar(
        in:input,
        pos:LFTri.kr(
            freq: pan_variation_freq,
            iphase:pan_variation_iphase,
            mul: pan_variation_amp,
            add: pan),
        level:1);
    input = BHiPass.ar(in:input, freq:lowcut, rq:rq);

```

```

input = input * Line.kr(0, 1, fade_in);
env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
  gate:gate, doneAction:done_action);

Out.ar(out_bus, input * amp * direct * env);
// Out.ar(reverb_bus, input * amp * (1 - direct) * env);
};

/***** LOCAL FUNCTIONS FOR PATCH #2 *****/

// GH Bell Pitch Generation Function with
// Fixed Random Seed Harmonic Definitions
// Fundamental Pitch = C3 (130.81 Hz or 48.midicps)
var grandfatherClockPitch = {
  arg semitone_offset = 0, amp = 0.0875, pan = 0.0, out = 0;
  var env = EnvGen.kr(Env([1, 1, 0], #[1, 1]), doneAction:2);
  var input = Decay.ar(Impulse.ar(0.45), 1, BrownNoise.ar(0.01));
  // get fundamental frequency of the bell (i.e. C3 = 130.81 Hz)
  var midi_value = 48;
  var fundamental = midi_value.midicps;
  var bell = Klank.ar([
    /* freqs */
    1122, 4554, 6205, fundamental, 6978, 4307, 4596, 5346,
    2095, 9542, (fundamental * 2), 1347
  ] * 2.pow(semitone_offset/12),
    amp,
    /* rings */ [5, 10, 8, 10, 4, 8, 3, 4, 8, 7, 3, 10]
  ], input:input);

  bell = BHiPass.ar(in:bell, freq:40, rq:0.8);

  Out.ar(out, Pan2.ar(bell*env, pan));
};

/***** LOCAL FUNCTIONS FOR PATCH #3 (ReverbGroupChannel) *****/

var midModPulse =
{
  arg out_bus, reverb_bus,
  fade_in = 0, gate = 1, // passed gate=0 value will fade out
  release_time = 8,
  direct = 0; // controls proportion of direct/processed sound
  var pulse, env;

  // Decaying sine wave pulses with modulation
  pulse = Decay2.ar(in:Impulse.ar(freq:0.3, phase:0.25),
    attackTime:0.15,
    decayTime:1,
    mul:SinOsc.ar(freq:SinOsc.kr(freq:0.2,
      mul:44.midicps, add:51.midicps)));
  // Fundamental of mid_mod_pulse is on Eb4
  pulse = pulse * 1.5;
  pulse = pulse * Line.kr(0, 1, fade_in);

  env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:2);

  // Out.ar(out_bus, (pulse*direct) * 0.3); // signal output
  Out.ar(reverb_bus, (pulse*(1-direct)) * env); // reverb output
};

```

```

var lowModPulse =
{
  arg out_bus, reverb_bus, mul = 1, partial = 1, gain_division = 1,
  fade_in = 0, gate = 1, // passed gate=0 value will fade out
  release_time = 8,
  direct = 0; // controls proportion of direct/processed sound
  var pulse, env;
  var freq = 32.midicps; // Fundamental of low_mod_pulse is on Ab2

  // Decaying sine wave pulses with modulation
  // The optional array adds a harmonizer on partials 3, 5, 7 etc...
  pulse =
  Array.fill(1, { arg i;
    Decay2.ar(in:Impulse.ar(freq:0.3, phase:0.5),
      attackTime:0.2,
      decayTime:1,
      mul:SinOsc.ar(freq:SinOsc.kr(freq:0.2,
        mul:(freq)*mul, add:freq*partial ))) * gain_division;
  });
  pulse = Pan2.ar(pulse, 0, 0.5);
  pulse = BHiPass.ar(in:pulse, freq:60, rq:0.6);
  pulse = pulse * Line.kr(0, 1, fade_in);

  env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:2);

  // Out.ar(out_bus, (pulse*direct) * 0.5); // signal output
  Out.ar(reverb_bus, (pulse*(1-direct)) * env); // reverb output
};

var bassPulse =
{
  arg out_bus, reverb_bus,
  fade_in = 0, gate = 1, // passed gate=0 value will fade out
  release_time = 8,
  direct = 0; // controls proportion of direct/processed sound
  var pulse, env;
  var freq = 32.midicps; // Fundamental of bass_pulse is on Ab1

  // Decaying sine wave pulses with a slight modulation
  // The optional array adds a harmonizer on partials 3, 5, 7 etc...
  pulse =
  Array.fill(1, { arg i;
    Decay2.ar(in:Impulse.ar(freq:0.3, phase:0.125),
      attackTime:0.2,
      decayTime:1,
      mul:SinOsc.ar(freq:SinOsc.kr(freq:0.2,
        mul:/*(freq)*(i+1)*/ 0, add:freq*(i*2+1) ))) * 1/(i+1);
  });
  pulse = Pan2.ar(pulse, 0, /*3.5*/ 1.5);
  pulse = BHiPass.ar(in:pulse, freq:60, rq:0.6);
  pulse = pulse * Line.kr(0, 1, fade_in);

  env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:2);

  // Out.ar(out_bus, (pulse*direct) * 0.5); // signal output
  Out.ar(reverb_bus, (pulse*(1-direct)) * env); // reverb output
};

// SynthDef to record the session and save to disk:
SynthDef(\recordSession, { arg buffer;
  DiskOut.ar(buffer, In.ar(0,2))

```



```

    }).add;

    // SynthDefs & buses for effects processors:
    SynthDef(\stereoReverbChannel, stereoReverbChannel).add;
    SynthDef(\combChannel, combChannel).add;
    12.do{ arg i; SynthDef("delayTap"+i, delayTap).add; };
    SynthDef(\atmosphericReverbChannel, atmosphericReverbChannel).add;

    // SynthDefs for sample playback:
    SynthDef(\stereoSamplePlayer, stereoSamplePlayer).add;
    SynthDef(\monoToStereoSamplePlayer, monoToStereoSamplePlayer).add;
    SynthDef(\monoSamplePlayer, monoSamplePlayer).add;

    // SynthDefs for sample doppler effects:
    SynthDef(\monoCircularSampleDoppler,
        monoCircularSampleDoppler).add;
    SynthDef(\stereoTriangularSampleDoppler,
        stereoTriangularSampleDoppler).add;
    SynthDef(\monoCircularFormantDoppler,
        monoCircularFormantDoppler).add;
    SynthDef(\stereoTriangularFormantDoppler,
        stereoTriangularFormantDoppler).add;

    // SynthDefs for sample pitch shifts & frequency modulation:
    SynthDef(\monoSampleShift , monoSampleShift).add;
    SynthDef(\monoToStereoSampleShift , monoToStereoSampleShift ).add;
    SynthDef(\monoToStereoFormantShift, monoToStereoFormantShift).add;

    // SynthDef for the bell modelling synth:
    SynthDef(\grandfatherClockPitch, grandfatherClockPitch).add;

    // SynthDefs for the modular synths:
    SynthDef(\midModPulse, midModPulse).add;
    SynthDef(\lowModPulse, lowModPulse).add;
    SynthDef(\bassPulse, bassPulse).add;
}

/***** GENERAL INSTANCE METHODS FOR DYNAMICS PROCESSING *****/

masterChannel {
    {
        ReplaceOut.ar(0, Limiter.ar(
            BHiPass.ar(in:In.ar(bus:0, numChannels:2), freq:40, rq:0.6),
            level:0.99, dur:0.01))
    }.play(RootNode(server), addAction:\addToTail);
}

/***** GENERAL INSTANCE METHODS FOR EFFECTS PROCESSING *****/

freeEffects {

    // free effects channels first so that input buses can be reconfigured...
    if(comb.isPlaying) { comb.free; comb = nil };
    12.do{ arg i; if(tap[i].isPlaying) { tap[i].free; tap[i] = nil } };
    if(reverb.isPlaying) { reverb.free; reverb = nil };
}

initBuses {

    reverb_bus.free;
    reverb_bus = Bus.audio(server);

```

```

12.do{ arg i;
    tap_bus[i].free;
    tap_bus[i] = Bus.audio(server)
};
comb_bus.free;
comb_bus = Bus.audio(server);
}

directMicSignal {

    audio_reverb.free ;
    audio_reverb = nil;
    gui.setMicBusButton(0);

    ^"Direct microphone signal only!"
}

bus_Reverb {
    arg new_at_start = true, delay_bus_exists = false, comb_bus_exists = false;
    var msg = "Reverb bus/microphone send added!";
    var reverb_arg = [\in_bus, reverb_bus];
    var end_args = [\delaytime, 1/4, \pan, 0, \next_tap, 0, \next_comb, 0];
    var args = [], delay_arg = [], comb_arg = [];

    if(delay_bus_exists) {
        delay_arg = [\delay_bus, tap_bus[0]];
        end_args[5] = 1; // opens gate for delay taps
    };
    if( comb_bus_exists) { comb_arg = [\comb_bus, comb_bus] };
    args = reverb_arg++delay_arg++comb_arg++end_args;
    // args.postln; // debug

    // reset atmospheric reverb channel:
    if(new_at_start)
    { reverb = Synth.new(\atmosphericReverbChannel, args,
        target:p3_RG_group_node) }
    // else...
    { reverb = Synth.before(tap[0], \atmosphericReverbChannel, args) };
    NodeWatcher.register(reverb);

    if( audio_reverb == nil ) {
        audio_reverb = { Out.ar(reverb_bus, SoundIn.ar(bus:0,mul:1)) }.play
    };
    gui.setMicBusButton(1);

    ^msg
}

addTaps {
    arg taps = 7, new_at_start = true;
    var delaytime = 19/7, n = taps-1;
    var args = [\in_bus, tap_bus[n], \delaytime, delaytime,
        \pan, pan_array[n], \next_tap, 0];

    // sets a specified number of delay tap channels:
    if(new_at_start)
    { tap[n] = Synth.new("delayTap"++n, args, target:p3_RG_group_node) }
    // else...
    { tap[n] = Synth.before(comb, "delayTap"++n, args) };
    NodeWatcher.register(tap[n]);
}

```

```

    for(n - 1, 0) {arg i;
        tap[i] = Synth.before(tap[i+1], "delayTap"++i,
            [\in_bus, tap_bus[i], \tap_bus, tap_bus[i+1], \delaytime, delaytime,
            \pan, pan_array[i], \next_tap, 1]);
        NodeWatcher.register(tap[i]);
    }
}

bus_Reverb__7TapDelay {
    var msg = "Reverb & 7-tap delay bus/microphone send added!";

    // set 7 delay tap channels (sent from reverb channel):
    this.addTaps(7);
    // reset atmospheric reverb channel:
    this.bus_Reverb(new_at_start:false, delay_bus_exists:true);
    gui.setMicBusButton(2);

    ^msg
}

bus_Reverb_12TapDelay {
    var msg = "Reverb & 12-tap delay bus/microphone send added!";

    // set 12 delay tap channels (sent from reverb channel):
    this.addTaps(12);
    // reset atmospheric reverb channel:
    this.bus_Reverb(new_at_start:false, delay_bus_exists:true);
    gui.setMicBusButton(3);

    ^msg
}

bus_Reverb_12TapDelay_Comb {
    arg start_comb = true;
    var msg = "Reverb & 12-tap delay & comb filter bus/microphone send added!";

    // set comb filter channel:
    comb = Synth.new(\combChannel, [\in_bus, comb_bus, \pan, 0],
        target: p3_RG_group_node);
    NodeWatcher.register(comb);
    // set 12 delay tap channels (sent from reverb channel):
    this.addTaps(12, new_at_start:false);
    // reset atmospheric reverb channel:
    this.bus_Reverb(new_at_start:false, delay_bus_exists:true,
        comb_bus_exists:true);
    Routine
    {
        0.1.wait; // wait for comb filter to be sent
        if(start_comb) { subpatchAction[2][2].(); } // starts comb filter
    }.play;
    gui.setMicBusButton(4);

    ^msg
}

/***** INSTANCE METHODS FOR PATCH #1 & #3 (NatureSoundsGroupChannel) *****/

neotropicalAmbience {
    arg fade_in = 15, target;

    // free any synths which are already active:

```

```

    this.freeNeotropicalAmbience;

    pigmy_owl_player = Synth.new(\stereoSamplePlayer,
        [\bufnum, pigmy_owl, \amp, 1.625, \fade_in, fade_in],
        target:target /* p1_NSG_group_node */);
    NodeWatcher.register(pigmy_owl_player);

    frog_Amazon_player = Synth.new(\monoToStereoSampleShift,
        [\bufnum, frog_Amazon, \amp, 0.8, \pan, 1, \fade_in, fade_in,
        \reverb_bus, s_reverb_bus, \direct, 0.8],
        target:target /*p1_NSG_group_node*/);
    NodeWatcher.register(frog_Amazon_player);

    frog_Peru2_player = Synth.new(\monoToStereoSampleShift,
        [\bufnum, frog_Peru2, \amp, 0.45, \pan, -1, \fade_in, fade_in,
        \reverb_bus, s_reverb_bus, \direct, 0.8],
        target:target /*p1_NSG_group_node*/);
    NodeWatcher.register(frog_Peru2_player);
}

freeNeotropicalAmbience {

    if(pigmy_owl_player .isPlaying) { pigmy_owl_player.free };
    if(frog_Amazon_player.isPlaying) { frog_Amazon_player.free };
    if(frog_Peru2_player .isPlaying) { frog_Peru2_player.free };
    pigmy_owl_player = nil;
    frog_Amazon_player = nil;
    frog_Peru2_player = nil;
}

owlHarmonizer {
    arg mod_at_start = true, wait_before = 30, wait_after = 0, target;
    // avoid two zero wait args else the infinite loop will crash the program!

    this.freeOwlHarmonizer;
    if(mod_at_start == false) {
        freq_modulation_routine.stop; // clear any freq mod calls to the synths
        pan_modulation_routine.stop; // clear any pan mod calls to the synths
    };

    owl_harmonizer_routine = Routine
    {
        var sum = 0, interval = 2, s_amp = 1, f_amp = 1.25 /*0.625*/;
        var pan_array = Array.newClear(8);

        for(8, 4) { arg i;
            sampler_pan_array[i] = rand2(1.0);
            formant_pan_array[i] = rand2(1.0);
            formant_shift_rate_array[i] = fundamental_shift_rate * (i+1) * (4/9);
            tempo_clock[0].sched(sum) {
                owl_sample_shifter [i] = Synth.new(\monoToStereoSampleShift,
                    [\bufnum, tawny_owl, \amp, s_amp/(i+1),
                    \pan, sampler_pan_array[i], \fade_in, 10,
                    \pitch_rate, fundamental_shift_rate * (i+1),
                    \lowcut, 1500, \rq, 8,
                    \reverb_bus, s_reverb_bus, \direct, 1, \done_action, 13],
                    target:target);
                NodeWatcher.register(owl_sample_shifter[i]);
                // Change sample rate to 9/4 = 2.25 to allow
                // max PitchShift rate up to the 9th harmonic
                owl_formant_shifter[i] = Synth.new(\monoToStereoFormantShift,

```

```

        [\bufnum, tawny_owl, \amp, f_amp/(i+1),
         \pan, formant_pan_array[i],
         \fade_in, 10, \sample_rate, 9/4,
         \pitch_rate, formant_shift_rate_array[i],
         \lowcut, 1125, \rq, 8, \done_action, 13], target:target);
NodeWatcher.register(owl_formant_shifter[i]);
if(mod_at_start)
{ this.freqModLoop(wait_before, wait_after) }
};
sum = sum + interval;
};

for(3, 1) { arg i;
sampler_pan_array[i] = rand2(1.0);
formant_pan_array[i] = rand2(1.0);
formant_shift_rate_array[i] = fundamental_shift_rate * (i+1);
tempo_clock[0].sched(sum) {
owl_sample_shifter [i] = Synth.new(\monoToStereoSampleShift,
[\bufnum, tawny_owl, \amp, s_amp/(i+1),
 \pan, sampler_pan_array[i], \fade_in, 10,
 \pitch_rate, fundamental_shift_rate * (i+1),
 \lowcut, 1000, \rq, 6.5,
 \reverb_bus, s_reverb_bus, \direct, 1, \done_action, 13],
target:target);
NodeWatcher.register(owl_sample_shifter[i]);
owl_formant_shifter[i] = Synth.new(\monoToStereoFormantShift,
[\bufnum, tawny_owl, \amp, f_amp/(i+1),
 \pan, formant_pan_array[i], \fade_in, 10,
 \pitch_rate, formant_shift_rate_array[i],
 \lowcut, 750, \rq, 6.5, \done_action, 13],
target:target);
NodeWatcher.register(owl_formant_shifter[i]);
if(mod_at_start)
{ this.freqModLoop(wait_before, wait_after) }
};
sum = sum + interval;
};

tempo_clock[0].sched(sum) {
sampler_pan_array[0] = 0;
formant_pan_array[0] = 0;
formant_shift_rate_array[0] = fundamental_shift_rate;
owl_sample_shifter[0] = Synth.new(\monoToStereoSampleShift,
[\bufnum, tawny_owl, \amp, s_amp/3, \pan, 0, \fade_in, 10,
 \pitch_rate, fundamental_shift_rate,
 \lowcut, 80, \rq, 0.6,
 \reverb_bus, s_reverb_bus, \direct, 1, \done_action, 13],
target:target);
NodeWatcher.register(owl_sample_shifter[0]);
owl_formant_shifter[0] = Synth.new(\monoToStereoFormantShift,
[\bufnum, tawny_owl, \amp, f_amp/3, \pan, 0, \fade_in, 10,
 \pitch_rate, fundamental_shift_rate,
 // reverse owl call on fundamental formant shift:
 \direction, -1, \lowcut, 80, \rq, 0.6, \done_action, 13],
target:target);
NodeWatcher.register(owl_formant_shifter[0]);
this.freqModLoop(wait_before, wait_after);
// debug...
// ("sampler_pan_array =" + sampler_pan_array).postln;
// ("formant_pan_array =" + formant_pan_array).postln;
// ("formant_shift_rate_array =" + formant_shift_rate_array).postln;

```

```

    };
    }.play
}

freeOwlHarmonizer {

    // free any synths which are already active...
    9.do{ arg i;
        if( owl_sample_shifter[i].isPlaying ) {
            owl_sample_shifter[i].free;
            owl_sample_shifter[i] = nil;
        };
        if( owl_formant_shifter[i].isPlaying ) {
            owl_formant_shifter[i].free;
            owl_formant_shifter[i] = nil;
        };
    };
    tempo_clock[0].clear; // clear previously scheduled owl harmonizer events
    owl_harmonizer_routine.stop; // stop if the routine is already running!
}

freqModLoop {
    arg wait_before = 30, wait_after = 0;
    // avoid two zero wait args else the infinite loop will crash the program!

    freq_modulation_routine.stop; // stop if the routine is already running!
    freq_modulation_routine = Routine
    {
        // Infinite loop resets the random freq mod values perpetually...
        {
            0.1.wait;
            wait_before.wait;
            // "freq loop! ".post; // debug
            // cancel direction change on the fundamental formant pitch:
            if( owl_formant_shifter[0].isPlaying ) {
                owl_formant_shifter[0].set(\direction, 1);
            };

            if(frog_Amazon_player.isPlaying ) {
                frog_Amazon_player.set(\pitch_variation_freq, rand(13) + 6,
                    \pitch_variation_amp, rand(1.0),
                    \amp_mod_cycle_freq, 1/24);
            };
            if(frog_Peru2_player.isPlaying ) {
                frog_Peru2_player.set(\pitch_variation_freq, rand(13) + 6,
                    \pitch_variation_amp, rand(1.0),
                    \amp_mod_cycle_freq, 1/24);
            };

            for(8, 0) { arg i;
                // frequency modulation variables:
                var s_pv_freq, s_pv_amp, f_pv_freq, f_pv_amp, f_p_rate;
                // reverse playback modulation variables:
                var s_direction, f_direction;

                /* Randomly generate modulation parameters
                    for the sample shifts: */
                s_pv_freq = rand(12) + 1;
                s_pv_amp = rand(1.0);
                // s_direction = [1,-1].choose;
                /* Randomly generate modulation parameters

```

```

        for the formant shifts: */
        f_pv_freq = rand(5.0) + 1;
        // positive f_pv_amp value for ascending LFSaw,
        // negative for descending...
        f_pv_amp = rand2(1.0);
        // f_direction = [1,-1].choose;

        if( owl_sample_shifter [i].isPlaying ) {
            owl_sample_shifter [i].set(
                \pitch_variation_freq, s_pv_freq,
                \pitch_variation_amp, s_pv_amp * (i+1),
                \amp_mod_cycle_freq, 1/24);
        };
        if( owl_formant_shifter[i].isPlaying ) {
            owl_formant_shifter[i].set(
                \pitch_variation_freq, f_pv_freq,
                \pitch_variation_amp, f_pv_amp * (i+1),
                \amp_mod_cycle_freq, 1/24);
        };
    };
    wait_after.wait
}.loop
}.play
}

freeFreqMod {

    freq_modulation_routine.stop;

    // reset reverse direction on the fundamental formant pitch:
    if( owl_formant_shifter[0].isPlaying ) {
        owl_formant_shifter[0].set(\direction, -1);
    };

    if(frog_Amazon_player.isPlaying) {
        frog_Amazon_player.set(
            \pitch_variation_freq, 0,
            \pitch_variation_amp, 0,
            \amp_mod_cycle_freq, 0)
    };
    if(frog_Peru2_player.isPlaying) {
        frog_Peru2_player.set(
            \pitch_variation_freq, 0,
            \pitch_variation_amp, 0,
            \amp_mod_cycle_freq, 0)
    };

    9.do{ arg i;
        if( owl_sample_shifter [i].isPlaying ) {
            owl_sample_shifter [i].set(
                \pitch_variation_freq, 0,
                \pitch_variation_amp, 0,
                \amp_mod_cycle_freq, 0)
        };
        if( owl_formant_shifter[i].isPlaying ) {
            owl_formant_shifter[i].set(
                \pitch_variation_freq, 0,
                \pitch_variation_amp, 0,
                \amp_mod_cycle_freq, 0);
        }
    }
}

```

```

}

panModLoop {

    pan_modulation_routine.stop;
    pan_modulation_routine = Routine
    {
        // Infinite loop resets the random pan mod values perpetually...
        {
            // "pan loop! ".post; // debug
            if(frog_Amazon_player.isPlaying) {
                frog_Amazon_player.set(\pan, 0,
                    \pan_variation_freq, 1/(rand(4.0)),
                    \pan_variation_iphase, rand(4.0),
                    \pan_variation_amp, rand2(1.0));
            };
            if(frog_Peru2_player.isPlaying) {
                frog_Peru2_player.set(\pan, 0,
                    \pan_variation_freq, 1/(rand(4.0)),
                    \pan_variation_iphase, rand(4.0),
                    \pan_variation_amp, rand2(1.0));
            };

            sampler_pan_array[0] = rand2(1.0);
            formant_pan_array[0] = rand2(1.0);
            for(8, 0) { arg i;
                if( owl_sample_shifter [i].isPlaying ) {
                    owl_sample_shifter [i].set(\pan, 0,
                        \pan_variation_freq, 1/(rand(4.0)),
                        \pan_variation_iphase, rand(4.0),
                        \pan_variation_amp, sampler_pan_array[i]);
                };
                if( owl_formant_shifter[i].isPlaying ) {
                    owl_formant_shifter[i].set(\pan, 0,
                        \pan_variation_freq, 1/(rand(4.0)),
                        \pan_variation_iphase, rand(4.0),
                        \pan_variation_amp, formant_pan_array[i]);
                };
            };
            15.wait
        }.loop
    }.play
}

freePanMod {

    pan_modulation_routine.stop;

    // clear pan modulation and reset original pan values...
    if(frog_Amazon_player.isPlaying, {
        frog_Amazon_player.set(\pan, 1,
            \pan_variation_freq, 0,
            \pan_variation_iphase, 0,
            \pan_variation_amp, 0)
    });
    if(frog_Peru2_player.isPlaying) {
        frog_Peru2_player.set(\pan, -1,
            \pan_variation_freq, 0,
            \pan_variation_iphase, 0,
            \pan_variation_amp, 0)
    };
}

```



```

9.do{ arg i;
  if( owl_sample_shifter [i].isPlaying ) {
    owl_sample_shifter [i].set(
      \pan, if(i == 0, {0}, { sampler_pan_array[i] }),
      \pan_variation_freq, 0,
      \pan_variation_iphase, 0,
      \pan_variation_amp, 0)
  };
  if( owl_formant_shifter[i].isPlaying ) {
    owl_formant_shifter[i].set(
      \pan, if(i == 0, {0}, { formant_pan_array[i] }),
      \pan_variation_freq, 0,
      \pan_variation_iphase, 0,
      \pan_variation_amp, 0)
  }
}

/***** INSTANCE METHODS FOR PATCH #2 *****/

// GH Additive Synthesizer with sine wave input and pulse wave filter
// on up to the first 12 harmonic partials of the input frequency
additiveSynth {
  arg midi_value = 48, input_partial = 2, input_gain = 1, pan = 0, isynth = 0,
  jsynth = 0, /* 2D synth indices to free memory */ bell = true,
  fade_in = 0.02, gate = 1, release_time = 8;
  var input_frequency = midi_value.midicps * input_partial;
  var partial, partial_gain_division, filter_input_level, semitone_offset;
  var input = nil, out = 0, i = 0, partial_list = List.new;
  var env;

  semitone_offset = (input_frequency.cpsmidi - 48);
  if(bell)
  {
    this.bellPitchSynth(semitone_offset:semitone_offset,
      pan:pan, amp:1/25);
  };

  /* A loop to reject any inaudible frequencies above 20,000 Hz
  This will make the program more efficient, reduce lost
  speaker power and save computer memory...
  */
  // i = 0;
  while{ i < 12 }
  {
    partial_list.add;
    partial_list[i] = (i+1) * input_frequency;
    if( partial_list[i] > 20000 )
    {
      partial_list.pop;
      i = 12 // breaks the loop
    }
    // else...
    { i = i + 1 };
  };

  // env value in case a synthdef is required at a later stage:
  env = EnvGen.kr(Env.cutoff(releaseTime:release_time),
    gate:gate, doneAction:2);

```

```

p2[isynth][jsynth] = Array.fill( (partial_list.size - 1), {
  arg count;
  // Avoid distorted filter on fundamental pitches:
  // filter_input_level = if (count == 0, 0, {input_gain});
  filter_input_level = input_gain;
  // Reduce excessive gain-level on upper partials:
  partial_gain_division = 1/(count + 2);

  input = { (SinOsc.ar(partial_list[count + 1],
    mul: max(0 , Pulse.kr(count + 2/32, mul: filter_input_level))
  ) * partial_gain_division) };
  // input = input * Line.kr(0, 1, fade_in);
  // input = input * env;
  { Pan2.ar(input, pan, 0.03) }.play
  // NodeWatcher.register(p2[isynth][jsynth][count])
});

^p2[isynth][jsynth]
}

bellPitchSynth {
  arg semitone_offset = 0, pan = 0, amp = 0.0875;
  var synth;
  // arg semitone_offset allows pitch transpositions

  // 0.1.wait;
  synth = Synth(\grandfatherClockPitch,
    [\semitone_offset, semitone_offset, \amp, amp, \pan, pan]
  );

  ^synth
}

init_p2 {
  // Initialize 2D Array so that all synths
  // & routines in patch #2 can be stored:
  2.do{ arg i; p2[i] = Array.newClear(3); };
  4.do{ arg i; p2[i+2] = Array.newClear(4); };
  p2[6] = Array.newClear(6);
  p2[7] = nil; // reserved for bell synth bassline routine
  p2[8] = Array.newClear(6);
}

freeAdditiveSub {
  arg i;

  // Free each additive synth 3D array element...
  p2[i].size.do{ arg j;
    p2[i][j].size.do{ arg k;
      // if(p2[0][j][k].isPlaying)
      p2[i][j][k].free;
      p2[i][j][k] = nil;
    }
  };
  tempo_clock[1][i].clear; // clears all scheduled events
  tempo_clock[1][9].clear; // clear routine to free other additive synths
}

freeSynths_p2 {
  arg i_min, i_max, j_min, j_max, sum, interval;

```

```

Routine
{
    for(i_min,i_max) {arg i;
        for(j_min,j_max) {arg j;
            tempo_clock[1][9].sched( sum ) {
                if( p2[i][j] != nil ) {
                    p2[i][j].size.do{ arg k;
                        if( p2[i][j][k] != nil ) {
                            p2[i][j][k].release(interval);
                            p2[i][j][k].onFree{ p2[i][j][k] = nil; }
                        }
                    }
                }
            };
            sum = sum + interval;

            tempo_clock[1][9].sched( sum ) {
                if( p2[i][j] != nil ) {
                    {
                        gui.postMsg("p2["++i++"]["++j++"] free! ",
                            line_feed:false).post
                    }.defer; // debug data
                    p2[i][j] = nil;
                }
            }
        }
    }.play;

    ^sum; // sum returned by the method
}

/***** INSTANCE METHODS FOR PATCH #3 (ReverbGroupChannel) *****/

// [ nothing to include at present! ]

/***** GENERAL INSTANCE METHODS FOR DATA HANDLING *****/

initSubpatchFlags {

    number_of_subs.size.do{ arg i;
        // set subpatch flags for each patch...
        subpatch_flag[i] = Array.newClear(number_of_subs[i]);
        number_of_subs[i].do{ arg j; subpatch_flag[i][j] = false };
    };
}

// function creates a validation string when a subpatch is started
validSubpatch {
    |pch, sub, symbol = "started"|
    var unit = "";
    if(sub < 10) { unit = "0" };

    ^("subpatch #"++pch++"."++unit++sub++ "++symbol++!")
}

invalidSubpatch {
    |pch, sub|
    var unit = "";
    if(sub < 10) { unit = "0" };

```

```

    ^("Error: subpatch #"++pch++"."++unit++sub++" already started!")
}

setNextSubpatchValue {
    |pch, sub|
    var match = false;
    var i = 0;

    while { i < (number_of_subs.size - 1) }
    {
        if( pch == (i + 1) && sub == number_of_subs[i] )
        {
            next_patch = i + 2;
            next_subpatch = 1;
            match = true;
            i = (number_of_subs.size - 1); // breaks the loop
        }
        // else...
        { i = i + 1 }
    };

    if(match.not) {
        next_patch = pch;
        next_subpatch = sub + 1;
    };

    // debug...
    /*("last pch =" + last_patch + "last sub =" + last_subpatch).postln;
    ("next pch =" + next_patch + "next sub =" + next_subpatch).postln;*/
}

setLastSubpatchValue {
    |pch, sub|
    var match = false;
    var i = 1;

    while { i < number_of_subs.size }
    {
        if( pch == (i + 1) && sub == 1 )
        {
            last_patch = i;
            last_subpatch = number_of_subs[ i - 1 ];
            match = true;
            i = number_of_subs.size; // breaks the loop
        }
        // else...
        { i = i + 1 }
    };

    if(match.not) {
        last_patch = pch;
        last_subpatch = sub - 1;
    };

    // debug...
    /*("last pch =" + last_patch + "last sub =" + last_subpatch).postln;
    ("next pch =" + next_patch + "next sub =" + next_subpatch).postln;*/
}

validateSubpatch {

```

```

|flag, pch, sub|
var msg;

if( (flag == false) || (mode == "Testing") )
{
    if( (next_patch == pch) && (next_subpatch == sub)
        || (mode == "Testing") )
    {
        subpatchAction[pch-1][sub-1].();
        msg = this.validSubpatch(pch, sub);
        // if(mode == "Performance")
        // {
        flag = true;
        last_patch = pch;
        last_subpatch = sub;
        this.setNextSubpatchValue(pch, sub)
        // }
    }
    // else
    { msg = "Error: patches must be started in adjacent order!" }
}
// else
{ msg = this.invalidSubpatch(pch, sub) };
gui.postMessage(msg.post, line_feed:false); // posts a validation message

^flag // method returns a boolean variable
}

/***** GENERAL PUBLIC INSTANCE METHODS *****/

subpatch {
|pch = 1, sub = 01|
var exists = true;

if(pch.isInteger && sub.isInteger) {
    // check that the subpatch value exists...
    // size of number_of_subs array = number of patches in the program
    if( (pch <= 0) || (pch > number_of_subs.size) ) { exists = false };
    number_of_subs.size.do{ arg i;
        if( (pch == (i+1)) && ( (sub <= 0) || (sub > number_of_subs[i]) ) )
        { exists = false };
    };

    if(exists)
    {
        subpatch_flag[pch-1][sub-1] = this.validateSubpatch(
            subpatch_flag[pch-1][sub-1], pch, sub);
        ^""
    }
    // else...
    { ^"Error: invalid subpatch value!" };
}
// else...
{ ^"Invalid input: you must supply two integers as arguments..." }
}

freeSub {
|pch = 1, sub = 01|
var exists = true;

if(exists)

```

```

    {
        next_patch    = pch;
        next_subpatch = sub;
        this.setLastSubpatchValue( pch, sub );
        freeSubpatch[pch-1][sub-1].();
        ^this.validSubpatch(pch, sub, "freed")
    }
    // else...
    { ^"Error: invalid subpatch value!" }
}

freeAllSynths {

    // Free patch #1 synths, routines & scheduled events...
    2.do{ arg i; freeSubpatch[0][i].() };
    p1_release_routine.stop;
    // Free patch #2 synths, routines & scheduled events...
    10.do{ arg i; freeSubpatch[1][i].() };
    // Free patch #3 synths, routines & scheduled events...
    // (if not already freed by the above funtion calls)
    freeSubpatch[2][1].(); // free modular synths
    if(otter_CostaRica_player.isPlaying) { otter_CostaRica_player.free };
    if(mosquito_s_doppler.isPlaying)     { mosquito_s_doppler.free   };
    otter_CostaRica_player = nil;
    mosquito_s_doppler     = nil;
    p3_NSG_xfade_routine.stop;

    // Free audition node...
    if(audition_node.isPlaying) { audition_node.free };
    audition_node = nil;

    // Reset the subpatch directors...
    next_patch = 1; next_subpatch = 1;
    last_patch = 1; last_subpatch = 0;
    gui.setNextLabel;

    // Silence any ReverbGroupChannel delay/predelay signals...
    p3_RG_group_node.set(\gate, 0, \release_time, 0);
    // Free/reset effects channels...
    this.freeEffects;
    gui.postMessage("\nAll synths freed!".postln);

    ^this.bus_Reverb;
}

reopenGUI {

    if(gui == nil)
    {
        if(valid_instance_count != 0)
        {
            if(valid_instance)
            {
                /*
                    free any redundant analysis synths
                    before opening a new GUI window...
                */
                gui_meter.freeSynths;
                gui_scope.freeScopeSynth;

                gui = QCZ_EternalOwlCall_GUI(this);
            }
        }
    }
}

```

```

        ^gui
    }
    // else...
    {
        var msg =
            "\nYou have attempted to reopen a GUI window on an invalid "
            ++ "instance of the performance interface class!";

        msg.error;
        ^""
    }
}
// else...
{
    var msg =
        "\nUnable to reopen the GUI window because the program "
        ++ "was forcefully crashed using a Cmd+Period call!\n"
        ++ "To reopen the GUI try running the program again...";

    msg.warn;
    ^""
}
}
// else...
{
    ^gui.postMessage(
        "A GUI window is already open!\n"
        ++ "This may have been minimized by the user...\n");
}
}

/***** DEBUG METHODS FOR PATCH #2 *****/

// Debug function posts data about synths stored in the 2D Array p2:
p2_DebugData {

    var msg = "\n";

    msg = msg ++ "p2 is a 2-dimensional Array containing "
    ++ "all synths & routines in patch #2:\n";
    9.do{ arg i;
        msg = msg ++
            ("patch_2_0" ++ (i + 1) ++ " = p2[" ++ i ++ "] = " ++ p2[i] ++ "\n")
    };

    ^msg
}

/***** INSTANCE METHODS FOR INITIALIZATION *****/

initProgram {
    arg directory_path;
    /*
        The directory path is generally either
        = this.class.filenameSymbol.asString.dirname
        or...
        = thisProcess.nowExecutingPath.dirname
    */

    /***** GENERAL INITIALIZATION CODE *****/

```

```

dpath = directory_path;
"".postln;
// open a GUI window, sending the current object as a pointer...
if(QCZ_EternalOwlCall_GUI.class_gui_pointer == nil)
{
    gui = QCZ_EternalOwlCall_GUI(this);
    gui_meter = gui.z_meter_view;
    gui_scope = gui.z_scope_view;
}
// else...
{
    /* If a corresponding GUI window already exists, close/overwrite
       it and reboot the program to ensure full stability.. */
    var msg =
        "A GUI window is already open!\n"
        ++ "Rebooting and overwriting the existing "
        ++ "GUI window to ensure full stability...";

    gui = QCZ_EternalOwlCall_GUI.class_gui_pointer;
    gui.close;
    msg.inform;
    ^this.programStart(dpath);
};
gui.postMessage(
    ("Directory path for reading/writing = " ++ dpath ++ "\n").postln);

this.masterChannel; // processes dynamics on the RootNode
record = record_start_value;
record_dpath = dpath ++ "recordings.dir";

// make a "recordings.dir" directory if it doesn't already exist...
if(File.type(record_dpath) == \not_found)
{ File.mkdir(record_dpath) }
// else...
{
    // if "recordings.dir" file is not a directory
    // delete it and create a new directory...
    if(File.type(record_dpath) != \directory)
    {
        File.delete(record_dpath);
        File.mkdir (record_dpath);
    }
};
if(record) {
    startRecording();
    gui.postMessage("Record mode enabled!...".postln);
};

this.initSubpatchFlags;

p1_NSG_group_node = Group.new; NodeWatcher.register(p1_NSG_group_node);
p3_RG_group_node = Group.new; NodeWatcher.register(p3_RG_group_node );
p3_NSG_group_node = Group.new; NodeWatcher.register(p3_NSG_group_node);

tempo_clock[0] = TempoClock.new(11/15); // Sets 1st tempo clock to 44 BPM
tempo_clock[1] = Array.newClear(10); // Array of tempo clocks for patch #2
// Sets 2nd set of tempo clocks to 50 BPM...
10.do{ arg i; tempo_clock[1][i] = TempoClock.new(5/6); };
tempo_clock[2] = TempoClock.new(23/30); // Sets 3rd tempo clock to 46 BPM

// Add direct microphone signal, left + right pair from a mono input:

```



```

audio_direct = {
    Out.ar(bus:0, channelsArray:SoundIn.ar(bus:[0,0],mul:0.1))
}.play;

/** INITIALIZATION CODE FOR PATCH #1 & #3 (NatureSoundsGroupChannel) */

/* Read .aiff files from the audio.dir directory (located
   in the same directory as the program script)...
*/
audio_dpath = dpath +/+ "audio.dir";

tawny_owl      = Buffer.read(server,(audio_dpath
    +/+ "TawnyOwl_Germany_04.aiff"));
pigmy_owl      = Buffer.read(server,(audio_dpath
    +/+ "Otter_Peru_PigmyOwl_03.aiff"));
otter_CostaRica = Buffer.read(server,(audio_dpath
    +/+ "Otter_CostaRica_Streams&Birdsong_01.aiff"));
frog_Amazon     = Buffer.read(server,(audio_dpath
    +/+ "NeotropicalFrog_Amazon_Brasil_01.aiff"));
frog_Peru2      = Buffer.read(server,(audio_dpath
    +/+ "NeotropicalFrog_Peru2_01.aiff"));
mosquito        = Buffer.read(server,(audio_dpath
    +/+ "Mosquito_CostaRica_01.aiff"));

freq_modulation_routine.stop;
pan_modulation_routine.stop;

/***** INITIALIZATION CODE FOR PATCH #2 *****/

this.init_p2; // initialize 2D Array for patch #2

/***** INITIALIZATION CODE FOR EFFECTS PROCESSORS *****/

{ gui.postMessage("Initializing effects processors...".postln) }.defer;
// 4.wait; // wait for synthdefs to be sent to the server
s_reverb = Synth.new(\stereoReverbChannel,
    [\in_bus, s_reverb_bus, \amp, 1]);

reverb = Synth.new(\atmosphericReverbChannel,
    [\in_bus, reverb_bus, \delaytime, 1/4, \pan, 0,
    \next_tap, 0, \next_comb, 0], target:p3_RG_group_node);

audio_reverb = { Out.ar(reverb_bus, SoundIn.ar(bus:0,mul:1)) }.play;

{
    gui.postMessage( "Program initialized!\n"
        ++ "Post Window data will be displayed here...\n").postln)
}.defer;
}

// Main Function:
programStart {
    arg directory_path;

    Routine
    {
        Buffer.freeAll(server); // frees buffers already stored in memory
        // reboot the server to clear memory...
        "".postln;
        server.quit;
        0.1.wait;
    }
}

```

```

        "The program is booting, please wait!...\n".postln;
server.waitForBoot(
    onComplete:{ this.initProgram(directory_path) },
    onFailure: {
        var msg;

        if(valid_instance_count > 0)
        { valid_instance_count = valid_instance_count - 1 };

        "".postln;
        msg = "\nThe server failed to boot!"
        ++ "\nTry running the program again...\n";
        msg.error
    }
)
}.play(AppClock);
^""
}

initCZ_EternalOwlCall {
    // verify that the instance is valid...
    if(valid_instance_count == 0 || allow_multiple_instances)
    {
        valid_instance = true;
        valid_instance_count = valid_instance_count + 1;
        class_zpi_pointer = this;
    }
    // else
    {
        var msg;

        valid_instance = false;
        invalid_instance_count = invalid_instance_count + 1;

        msg = "\nThis performance interface class has "
        ++ "already been instantiated once!\n"
        ++ "For safety reasons the class' default valid "
        ++ "instance count cannot exceed 1.\n"
        ++ "Cmd+Period will reset the 'valid_instance_count' "
        ++ "classvar to zero\n"
        ++ "If a software developer requires multiple "
        ++ "valid instances the 'allow_multiple_instances' "
        ++ "classvar should be set to true.\n"
        ++ "This should only be required in exceptional circumstances!\n";

        msg.error;

        ^this
    };

    // variable, object & function initializations for 'this' instance:

    /***** GENERAL PRIVATE INSTANCE VARIABLES, OBJECTS & FUNCTIONS *****/

    number_of_subs = [5,11,6]; // number of subpatches in each patch
    mode = "Testing"; // can be set to "Performance" or "Testing"
    // 2D Array to store a flag for each subpatch...
    subpatch_flag = Array.newClear(number_of_subs.size);
    // 2D Arrays to store functions which start or free each subpatch...
    subpatchAction = Array.newClear(number_of_subs.size);

```

```

freeSubpatch = Array.newClear(number_of_subs.size);
tempo_clock = [nil,nil,nil];

// Variables to Enable Automatic Disk Recording:
record=true; record_start_value=true; // either both true or both false
already_recording = false;
record_path = nil; // references the current session recording

p2 = Array.newClear(9); // 2D Array to store all synths in patch #2

// Microphone Input Signal Synths:
audio_direct = nil;
audio_reverb = nil;

// Allocate Delay Channel Synth Array:
tap = Array.newClear(12);

// Effects Processing Buses:
s_reverb_bus = Bus.audio(server, numChannels:2);
reverb_bus = Bus.audio(server, numChannels:1);
comb_bus = Bus.audio(server, numChannels:1);
tap_bus = Array.newClear(12);
12.do{ arg i; tap_bus[i] = Bus.audio(server) };

// Effects Processing Ints, Floats & Arrays:
pan_array = [0, -1/3, 1/3, -2/3, 2/3, -1, 1, -2/3, 2/3, -1/3, 1/3, 0];

// GUI Object...
gui = nil;

/***** PRIVATE VARIABLES FOR PATCH #1 & #3 (NatureSoundsGroupChannel) *****/

// Owl Harmonizer Variables:
fundamental_shift_rate = 0.9;
// Synth Arrays for owl harmonizer:
owl_sample_shifter = Array.newClear(9);
owl_formant_shifter = Array.newClear(9);
// Float Value Arrays for owl harmonizer:
formant_shift_rate_array = Array.newClear(9);
sampler_pan_array = Array.newClear(9);
formant_pan_array = Array.newClear(9);

/***** PRIVATE VARIABLES FOR PATCH #3 (ReverbGroupChannel) *****/

mid_mod_pulse = Array.newClear(5);
low_mod_pulse = Array.newClear(5);
bass_pulse = Array.newClear(5);

/***** SET PRIVATE SUBPATCH ACTION & FREE FUNCTIONS FOR EACH PATCH *****/

number_of_subs.size.do{ arg i;
  subpatchAction[i] = Array.newClear(number_of_subs[i]);
  freeSubpatch [i] = Array.newClear(number_of_subs[i]);
};

/***** PRIVATE SUBPATCH FUNCTIONS FOR PATCH #1 *****/

// Subpatch #1.01...
subpatchAction[0][0] =
{
  // 0.2.wait; // wait for synthdefs to be sent to the server

```

```

    this.neotropicalAmbience(target:p1_NSG_group_node);

    // free synth if already active
    if(otter_CostaRica_f_doppler.isPlaying)
    { otter_CostaRica_f_doppler.free };
    otter_CostaRica_f_doppler = Synth.new(\monoCircularFormantDoppler,
        [\bufnum, otter_CostaRica, \amp, 0.65, \fade_in, 45, \rate, 1/16],
        target:p1_NSG_group_node);
    NodeWatcher.register(otter_CostaRica_f_doppler);
};

// Subpatch #1.02...
subpatchAction[0][1] =
{ this.owlHarmonizer(mod_at_start:false, target:p1_NSG_group_node) };

// Subpatch #1.03...
subpatchAction[0][2] = { this.panModLoop; };

// Subpatch #1.04...
subpatchAction[0][3] =
{
    p1_release_routine.stop;
    p1_release_routine = Routine
    {
        tempo_clock[0].clear; // clear events previously scheduled
        if(p1_NSG_group_node.isPlaying) {
            p1_NSG_group_node.set(\gate, 0, \release_time, 142.5)
        };
        140.wait; // free infinite loops before the end of the fade-out...
        freq_modulation_routine.stop;
        pan_modulation_routine .stop;
        { gui.postMsg("modulation routines free!".postln) }.defer
    }.play
};

// Subpatch #1.05...
subpatchAction[0][4] =
{
    // free effects channels first so that input buses can be reconfigured...
    this.freeEffects;
    this.bus_Reverb__7TapDelay;
};

/***** PRIVATE SUBPATCH FUNCTIONS FOR PATCH #2 *****/

// Subpatch #2.01...
// Additive synth on 2nd, 3rd & 5th partials of C3 (48.midi cps = 130.81 Hz)
subpatchAction[1][0] =
{
    Routine
    {
        this.freeAdditiveSub(0);
        this.additiveSynth(pan:-2/3, isynth:0, jsynth:0);
        tempo_clock[1][0].sched(5) { this.additiveSynth(input_partial:3,
            pan:-2/3, isynth:0, jsynth:1)
        };
        tempo_clock[1][0].sched(9) { this.additiveSynth(input_partial:5,
            pan:-2/3, isynth:0, jsynth:2)
        };
    }.play
};

```

```

};

// Subpatch #2.02...
// Additive synth on 5th, 3rd & 2nd partials of D3 (50.midicps = 146.83 Hz)
subpatchAction[1][1] =
{
    Routine
    {
        this.freeAdditiveSub(1);
        this.additiveSynth(midi_value:50, input_partial:5,
            pan:2/3, isynth:1, jsynth:0);
        tempo_clock[1][1].sched(8/3) { this.additiveSynth(50, 3, pan:2/3,
            isynth:1, jsynth:1) };
        tempo_clock[1][1].sched(16/3) { this.additiveSynth(50, 2, pan:2/3,
            isynth:1, jsynth:2) };
    }.play
};

// Subpatch #2.03...
/* Additive synth on 2nd, 3rd, 5th & 7th
   partials of Db2 (37.midicps = 69.30 Hz) */
subpatchAction[1][2] =
{
    Routine
    {
        this.freeAdditiveSub(2);
        this.additiveSynth(midi_value:37, input_partial:2,
            input_gain:1/3, pan:1, isynth:2, jsynth:0);
        tempo_clock[1][2].sched(2) {
            this.additiveSynth(37, 3, 1/3, 1, 2, 1) };
        tempo_clock[1][2].sched(4) {
            this.additiveSynth(37, 5, 1/3, 1, 2, 2) };
        tempo_clock[1][2].sched(7) {
            this.additiveSynth(37, 7, 1/3, 1, 2, 3) };
    }.play
};

// Subpatch #2.04...
/* Additive synth on 7th, 5th, 3rd & 2nd
   partials of Eb2 (39.midicps = 77.78 Hz) */
subpatchAction[1][3] =
{
    Routine
    {
        this.freeAdditiveSub(3);
        this.additiveSynth(midi_value:39, input_partial:7, input_gain:1/3,
            pan:-1, isynth:3, jsynth:0);
        tempo_clock[1][3].sched(4/3)
        { this.additiveSynth(39, 5, 1/3, -1, 3, 1) };
        tempo_clock[1][3].sched(4)
        { this.additiveSynth(39, 3, 1/3, -1, 3, 2) };
        tempo_clock[1][3].sched(20/3)
        { this.additiveSynth(39, 2, 1/3, -1, 3, 3) };
    }.play
};

// Subpatch #2.05...
/* Additive synth on 2nd, 5th, 3rd & 7th
   partials of F2 (41.midicps = 87.31 Hz) */
subpatchAction[1][4] =
{

```

```

Routine
{
    this.freeAdditiveSub(4);
    this.additiveSynth(midi_value:41, input_partial:2, input_gain:1/3,
        pan:1/3, isynth:4, jsynth:0);
    tempo_clock[1][4].sched(8/5)
    { this.additiveSynth(41, 5, 1/3, 1/3, 4, 1) };
    tempo_clock[1][4].sched(16/5)
    { this.additiveSynth(41, 3, 1/3, 1/3, 4, 2) };
    tempo_clock[1][4].sched(28/5)
    { this.additiveSynth(41, 7, 1/3, 1/3, 4, 3) };
}.play
};

// Subpatch #2.06...
/* Additive synth on 2nd, 3rd, 5th & 7th
   partials of G2 (43.midicps = 98.00 Hz) */
subpatchAction[1][5] =
{
    Routine
    {
        this.freeAdditiveSub(5);
        this.additiveSynth(midi_value:43, input_partial:2, input_gain:1/3,
            pan:-1/3, isynth:5, jsynth:0);
        tempo_clock[1][5].sched(4/5)
        { this.additiveSynth(43, 3, 1/3, -1/3, 5, 1) };
        tempo_clock[1][5].sched(12/5)
        { this.additiveSynth(43, 5, 1/3, -1/3, 5, 2) };
        tempo_clock[1][5].sched(24/5)
        { this.additiveSynth(43, 7, 1/3, -1/3, 5, 3) };
    }.play
};

// Subpatch #2.07...
// Additive synth on 7th, 11th & 13th partials of both C3 & D3
subpatchAction[1][6] =
{
    Routine
    {
        this.freeAdditiveSub(6);
        this.additiveSynth(midi_value:48, input_partial:7, input_gain:1,
            pan:-2/3, isynth:6, jsynth:0);
        tempo_clock[1][6].sched(5/2)
        { this.additiveSynth(48, 11, 1, -2/3, 6, 1) };
        tempo_clock[1][6].sched(9/2)
        { this.additiveSynth(48, 13, 1, -2/3, 6, 2) };
        tempo_clock[1][6].sched(6)
        { this.additiveSynth(50, 13, 1, 2/3, 6, 3) };
        tempo_clock[1][6].sched(22/3)
        { this.additiveSynth(50, 11, 1, 2/3, 6, 4) };
        tempo_clock[1][6].sched(26/3)
        { this.additiveSynth(50, 7, 1, 2/3, 6, 5) };
    }.play
};

// Subpatch #2.08...
// Adds a looped bass line of deep bell-like pitches
subpatchAction[1][7] =
{
    var i = 0, sum = 0, speed = 1.25;
    // bell_array values are supplied as semitone_offset args to the

```

```

// bellPitch function containing the following pitches offset from C3:
var pitch_string = ["C3", "D3", "Db2", "Ab2", "Eb2", "Bb2", "F2", "Gb1", "G2"];
var bell_array = [ 0, 2, -11, -4, -9, -2, -7, -18, -5];
var pan_array = [-1/4, 1/4, -1, 3/4, 1, -3/4, 1/2, 0, -1/2];

p2[7].stop; // stop any bell-loop routines already in memory
p2[7] = Routine
{
    {
        9.do{arg i;
            // lower pitches are exponentially longer:
            this.bellPitchSynth(bell_array[i] /*- 13*/,
                pan_array[i], /*amp:0.4*/);
            // this.bellPitchSynth(bell_array[i]+12, pan_array[i]);
            ( (1/speed) * 2.pow( (bell_array[i] * -1)/12 ) ).wait;
        };
    }.loop
}.play
};

// Subpatch #2.09...
// Additive synth on 17th, 19th & 23rd partials of both C3 & D3
subpatchAction[1][8] =
{
    Routine
    {
        this.freeAdditiveSub(8);
        this.additiveSynth(midi_value:48, input_partial:17, input_gain:1,
            pan:-2/3, isynth:8, jsynth:0);
        tempo_clock[1][8].sched(1)
        { this.additiveSynth(48, 19, 1, -2/3, 8, 1) };
        tempo_clock[1][8].sched(2)
        { this.additiveSynth(48, 23, 1, -2/3, 8, 2) };
        tempo_clock[1][8].sched(7/2)
        { this.additiveSynth(50, 23, 1, 2/3, 8, 3) };
        tempo_clock[1][8].sched(25/6)
        { this.additiveSynth(50, 19, 1, 2/3, 8, 4) };
        tempo_clock[1][8].sched(41/6)
        { this.additiveSynth(50, 17, 1, 2/3, 8, 5) };
    }.play
};

// Subpatch #2.10...
/* Gradually silences each synth in patch #2
and frees related computer memory */
subpatchAction[1][9] =
{
    Routine
    {
        var interval = /*15/7*/ 11/7, loop_start = [0,0,0,0];
        var message_time = 0;

        tempo_clock[1][9].clear;
        // clears any previously scheduled free routine

        loop_start[0] = 0;
        loop_start[1] = interval * 16;
        loop_start[2] = interval * 22;
        loop_start[3] = interval * 28;
        message_time = (interval * 34) + 0.01;
        // i.e. just after the process
    }
}

```

```

p2[7].stop;
p2[7] = nil;
{ gui.postMessage("p2[7] free! ", line_feed:false).post }.defer;
// free [2] to [5] (4 synths each)...
this.freeSynths_p2(2,5,0,3, loop_start[0], interval);
// free [0] to [1] (3 synths each)...
this.freeSynths_p2(0,1,0,2, loop_start[1], interval);
// free [6] (6 synths)...
this.freeSynths_p2(6,6,0,5, loop_start[2], interval);
// free [8] (6 synths)...
this.freeSynths_p2(8,8,0,5, loop_start[3], interval);
tempo_clock[1][9].sched( message_time ) {
    { gui.postMessage("\nAll p2 synths freed!".postln) }.defer;
};

if(tawny_owl_player.isPlaying) { tawny_owl_player.free };
tawny_owl_player = Synth.before(
    s_reverb, \monoToStereoSamplePlayer,
    [\bufnum, tawny_owl, \amp, 0.6875,
     \pan, 0, \fade_in, message_time,
     \reverb_bus, s_reverb_bus, \direct, 1]);
NodeWatcher.register(tawny_owl_player);
}.play
};

subpatchAction[1][10] = { this.directMicSignal };

/***** PRIVATE SUBPATCH FUNCTIONS FOR PATCH #3 *****/

// Subpatch #3.01...
// Sends audio input to the atmospheric reverb algorithm:
subpatchAction[2][0] =
{
    if(tawny_owl_player.isPlaying)
    { tawny_owl_player.set(\gate, 0, \release_time, 60) };

    /* free effects channels first so that
       input buses can be reconfigured... */
    this.freeEffects;
    this.bus_Reverb_12TapDelay_Comb(start_comb:false)
};

// Subpatch #3.02...
// Sends generator pulses to the atmospheric reverb algorithm:
subpatchAction[2][1] =
{
    // 0.2.wait; // wait for synthdef to be sent to the server

    // set mid mod pulse signal:
    if(mid_mod_pulse[0].isPlaying) { mid_mod_pulse[0].free };
    mid_mod_pulse[0] = Synth.new(\midModPulse,
        [\out_bus, [0,1], \reverb_bus, reverb_bus],
        target:p3_RG_group_node);
    NodeWatcher.register(mid_mod_pulse[0]);

    // set low mod pulse signal, harmonized up to the 5th partial:
    5.do{ arg i;
        if(low_mod_pulse[i].isPlaying) { low_mod_pulse[i].free };
        low_mod_pulse[i] = Synth.new(\lowModPulse,
            [\out_bus, [0,1], \reverb_bus, reverb_bus, \mul, (i+1),

```



```

        \partial, (i*2+1), \gain_division, (1/(i+1))],
        target:p3_RG_group_node);
    NodeWatcher.register(low_mod_pulse[i]);
};

// set bass pulse signal:
if(bass_pulse[0].isPlaying) { bass_pulse[0].free };
bass_pulse[0] = Synth.new(\bassPulse,
    [\out_bus, [0,1], \reverb_bus, reverb_bus],
    target:p3_RG_group_node);
NodeWatcher.register(bass_pulse[0]);
};

// Subpatch #3.03...
// Adds comb filters to the atmospheric reverb channel:
subpatchAction[2][2] =
{
    // Array used with previous settings...
    var kr_array = [1/17 ,1/3, 1/23, 1/5, 1/13] * 2;

    if(reverb.isPlaying) { reverb.set(\next_comb, 1) };
    if(comb.isPlaying) {comb.set(
        \control_freq, 6/5,
        \control_mul, 0.00249 /* 0.003875 */,
        \control_add, 0.0025)
    }
};

// Subpatch #3.04...
// Xfades reverberated synths with modulated neotropical nature sounds:
subpatchAction[2][3] =
{
    // ReverbGroupChannel Xfades Out, leaving just the bass pulse:
    if( mid_mod_pulse[0].isPlaying ) {
        mid_mod_pulse[0].set(\gate, 0, \release_time, 60)
    };
    if(comb.isPlaying) { comb.set(\gate, 0, \release_time, 60) };
    5.do{ arg i;
        if( low_mod_pulse[i].isPlaying ) {
            low_mod_pulse[i].set(\gate, 0, \release_time, 60);
        }
    };

    // NatureSoundsGroupChannel Xfades In:
    if(otter_CostaRica_player.isPlaying) { otter_CostaRica_player.free };
    if(mosquito_s_doppler.isPlaying) { mosquito_s_doppler.free };

    this.neotropicalAmbience(fade_in:60, target:p3_NSg_group_node);

    this.owlHarmonizer(mod_at_start:true, wait_before:0, wait_after:30,
        target:p3_NSg_group_node);
    this.panModLoop;

    // startPos offset so that the otter recording fades-in to the
    // 'colourful splashes' towards the end of the audio file...
    otter_CostaRica_player = Synth.new(\monoToStereoSamplePlayer,
        [\bufnum, otter_CostaRica, \start_pos, 48000 * 225,
        \amp, 0.675, \pan, 0, \fade_in, 60,
        \direct, 1], target:p3_NSg_group_node);
    NodeWatcher.register(otter_CostaRica_player);
};

```

```

// 0.1.wait; // wait for synthdef to be sent to the server
mosquito_s_doppler = Synth.new(\stereoTriangularSampleDoppler,
    [\bufnum, mosquito, \amp, 0.5625, \fade_in, 60, \rate, 1/32],
    target:p3_NSG_group_node);
NodeWatcher.register(mosquito_s_doppler);
};

// Subpatch #3.05...
// Xfades modulated neotropical nature sounds
// with reverberated/delayed tawny owl call:
subpatchAction[2][4] =
{
    if( tawny_owl_player.isPlaying ) { tawny_owl_player.free };

    9.do{ arg i;
        if( owl_sample_shifter [i].isPlaying ) {
            owl_sample_shifter [i].set(\gate, 0,
                \release_time, 60, \done_action, 2)
        };
        if( owl_formant_shifter[i].isPlaying ) {
            owl_formant_shifter[i].set(\gate, 0,
                \release_time, 60, \done_action, 2)
        };
    };
    if(mosquito_s_doppler.isPlaying) {
        mosquito_s_doppler.set(\gate, 0, \release_time, 60)
    };
    if(frog_Amazon_player.isPlaying) {
        frog_Amazon_player.set(\gate, 0, \release_time, 60)
    };
    if(frog_Peru2_player.isPlaying) {
        frog_Peru2_player .set(\gate, 0, \release_time, 60)
    };

    tawny_owl_player = Synth.new(\monoSampleShift,
        [\bufnum, tawny_owl, \amp, 2.25, \pan, 0, \fade_in, 60,
            \pitch_variation_freq, 2/7, \pitch_variation_amp, 1/4,
            \amp_mod_cycle_freq, 1/31, \lowcut, 500, \rq, 1,
            \reverb_bus, reverb_bus, \direct, 0],
        target:p3_NSG_group_node);
    NodeWatcher.register(tawny_owl_player);

    p3_NSG_xfade_routine.stop;
    p3_NSG_xfade_routine = Routine
    {
        57.5.wait; // free infinite loops before the end of the xfade...
        freq_modulation_routine.stop;
        pan_modulation_routine .stop;
        { gui.postMessage("modulation routines free!".postln) }.defer
    }.play
};

// Subpatch #3.06...
// Fade out synths:
subpatchAction[2][5] =
{
    tempo_clock[0].clear; // clear events previously scheduled
    if( p3_RG_group_node .isPlaying ) {
        p3_RG_group_node .set(\gate, 0, \release_time, 30)
    };
    if( p3_NSG_group_node.isPlaying ) {

```

```

        p3_NSG_group_node.set(\gate, 0, \release_time, 30)
    };
};

/***** PRIVATE FREE SUBPATCH FUNCTIONS FOR PATCH #1 *****/

freeSubpatch[0][0] =
{
    this.freeNeotropicalAmbience;
    if(otter_CostaRica_f_doppler.isPlaying )
    { otter_CostaRica_f_doppler.free };
    otter_CostaRica_f_doppler = nil;
};

freeSubpatch[0][1] =
{
    this.freeOwlHarmonizer;
    freq_modulation_routine.stop; // clear any freq mod calls to the synths
    pan_modulation_routine.stop; // clear any pan mod calls to the synths
};

freeSubpatch[0][2] = { this.freePanMod };

freeSubpatch[0][3] =
{
    p1_release_routine.stop;
    // restart all previous subpatches in patch #1...
    3.do{ arg i; subpatchAction[0][i].() }
};

freeSubpatch[0][4] = { this.freeEffects; this.bus_Reverb };

/***** PRIVATE FREE SUBPATCH FUNCTIONS FOR PATCH #2 *****/

7.do{ arg i; freeSubpatch[1][i] = { this.freeAdditiveSub(i) } };

freeSubpatch[1][7] = { p2[7].stop };

freeSubpatch[1][8] = { this.freeAdditiveSub(8) };

freeSubpatch[1][9] =
{
    tempo_clock[1][9].clear; // clear events previously scheduled
    if(tawny_owl_player.isPlaying) { tawny_owl_player.free };
    tawny_owl_player = nil;
};

freeSubpatch[1][10] = { this.freeEffects; this.bus_Reverb__7TapDelay };

/***** PRIVATE FREE SUBPATCH FUNCTIONS FOR PATCH #3 *****/

freeSubpatch[2][0] =
{ this.freeEffects; this.bus_Reverb__7TapDelay; this.directMicSignal };

freeSubpatch[2][1] =
{
    if(mid_mod_pulse[0].isPlaying) { mid_mod_pulse[0].free };
    mid_mod_pulse[0] = nil;
    5.do{ arg i;
        if(low_mod_pulse[i].isPlaying) { low_mod_pulse[i].free };
        low_mod_pulse[i] = nil
    }
};

```

```

    };
    if(bass_pulse[0].isPlaying) { bass_pulse[0].free };
    bass_pulse[0] = nil
};

freeSubpatch[2][2] =
{
    if(reverb.isPlaying) { reverb.set(\next_comb, 0) };
    if(comb.isPlaying) { comb.set(
        \control_freq, 0,
        \control_mul, 0.00249,
        \control_add, 0.0025)
    }
};

freeSubpatch[2][3] =
{
    {
        subpatchAction[2][0].(); // restores ReverbGroupChannel
        subpatchAction[2][1].(); // restarts modular synths
    }.defer;
    // 0.3.wait; // wait for synthdefs to be sent
    { subpatchAction[2][2].(); }.defer; // restarts comb filter

    if(otter_CostaRica_player.isPlaying) { otter_CostaRica_player.free };
    if(mosquito_s_doppler.isPlaying) { mosquito_s_doppler.free };
    otter_CostaRica_player = nil;
    mosquito_s_doppler = nil;

    this.freeNeotropicalAmbience;
    freeSubpatch[0][1].() // frees owl harmonizer
};

freeSubpatch[2][4] =
{
    p3_NSG_xfade_routine.stop;
    subpatchAction[2][3].(); // restarts nature sounds from subpatch # 3.04
    if( tawny_owl_player.isPlaying ) { tawny_owl_player.free };
    tawny_owl_player = nil;
};

freeSubpatch[2][5] =
{
    {
        subpatchAction[2][0].(); // restores ReverbGroupChannel
        subpatchAction[2][1].(); // restarts modular synths
    }.defer;
    // 0.3.wait; // wait for synthdefs to be sent
    {
        // restarts comb filter & nature sounds xfades...
        3.do{ arg i; subpatchAction[2][i + 2].() }
    }.defer;
};

/***** GENERAL PUBLIC INSTANCE VARIABLES, OBJECTS & FUNCTIONS *****/

// Public variables for sending values to other classes...
next_patch = 1; next_subpatch = 1;
last_patch = 1; last_subpatch = 0;
dpath = nil;

```

```

startRecording = {
    if( already_recording.not )
    {
        record = true;

        // create an output file for this buffer, leave it open
        record_path = record_dpath +/+ "EternalOwlCall_"
        ++ Date.localtime.stamp ++ ".aiff";

        diskout_buffer = Buffer.alloc(server, 2.pow(16), 2);
        diskout_buffer.write(record_path, "aiff", "int16", -1, 0, true);

        // 2.wait;
        // create the diskout node, making sure it comes after the source
        diskout_node = Synth.tail(nil, \recordSession,
            [\buffer, diskout_buffer]);
        diskout_node.onFree
        {
            stopRecording(); // also saves a file
            diskout_buffer.free; // free the buffer
            diskout_buffer = nil;
        };
        already_recording = true;

        { gui.setRecordModeButton(\on) }.defer;
        "Record mode enabled!"
    }
    // else...
    {
        { gui.setRecordModeButton(\on) }.defer;
        "The session is already recording!"
    }
};

stopRecording = {
    if(already_recording)
    {
        diskout_buffer.close; // close the buffer/soundfile and save
        {
            var msg = "A file has been recorded to disk!";

            if(gui != nil)
            {
                gui.setRecordModeButton(\off);
                gui.postMsg(msg.postln);
            }
            // else...
            { msg.postln }
        }.defer;
        record = false;
        already_recording = false;
        ""
    }
    // else...
    {
        var flag = if(record) { \on } /* else... */ { \off };
        { gui.setRecordModeButton(flag) }.defer;
        "Unable to stop recording, please try again!"
    }
}

```

```

};

// Public functions to be sent as values by other classes...
playLastRecordedSession = {

    if(record.not)
    {
        if(record_path != nil)
        {
            disk_in_buffer.free;
            disk_in_buffer = Buffer.read(server,record_path);

            playSample.(symbol:"Session", channels:2,
                        rate:1, start_pos_secs:0, loop:0);

            // return...
            record_path.basename
        }
        // else...
        {
            gui.freeRecPlayButton();
            ("No file has been recorded during the current session."
            ++ "\nStart record mode to enable this...")
        }
    }
    // else...
    {
        gui.freeRecPlayButton();
        "Record mode must be stopped to play a file back!..."
    }
};

// Microphone & bus routing commands ...
directMicrophoneSignal = {
    gui.postMessage("").postln;
    gui.postMessage( this.directMicSignal )
};
sendMicToBus_Reverb = {
    gui.postMessage("").postln;
    this.freeEffects;
    gui.postMessage( this.bus_Reverb )
};
sendMicToBus_Reverb__7TapDelay = {
    gui.postMessage("").postln;
    this.freeEffects;
    gui.postMessage( this.bus_Reverb__7TapDelay )
};
sendMicToBus_Reverb_12TapDelay = {
    gui.postMessage("").postln;
    this.freeEffects;
    gui.postMessage( this.bus_Reverb_12TapDelay )
};
sendMicToBus_Reverb_12TapDelay_Comb = {
    gui.postMessage("").postln;
    this.freeEffects;
    gui.postMessage( this.bus_Reverb_12TapDelay_Comb )
};

// Function indexes a bufnum linked to a symbol and plays the audio file:
playSample = {
    arg symbol, channels = 1, rate = 1, start_pos_secs = 0, loop = 1;

```

```

var bufnum = switch(symbol)
{ "Session"      } { disk_in_buffer  }
{ "TawnyOwl"     } { tawny_owl      }
{ "PigmyOwl"     } { pigmy_owl      }
{ "Otter"        } { otter_CostaRica }
{ "FrogAmazon"   } { frog_Amazon    }
{ "FrogPeru2"    } { frog_Peru2     }
{ "Mosquito"     } { mosquito       }
// Default function:
{ nil };

if(bufnum != nil) {
  if(audition_node.isPlaying) { audition_node.free };
  // Pan2 will be ignored on stereo files...
  audition_node = { Pan2.ar(PlayBuf.ar(
    numChannels:channels,
    bufnum:bufnum,
    rate:rate,
    startPos:48000 * start_pos_secs,
    loop:loop),
    pos:0)}.play;
  NodeWatcher.register(audition_node);

  bufnum.path.basename // returns the filename
}
// else...
{ "\""+symbol+"\" is not a valid symbol!" };
};

// Function indexes a SynthDef linked to a symbol and plays the Synth:
playDoppler = {
  arg symbol;

  var valid_symbol = switch(symbol)
  { "OtterSampleDoppler" } { true }
  { "MosquitoSampleDoppler" } { true }
  { "OtterFormantDoppler" } { true }
  { "MosquitoFormantDoppler" } { true }
  // Default function:
  { false };

  if(valid_symbol) {
    if(audition_node.isPlaying) { audition_node.free };
    audition_node = switch(symbol)
    { "OtterSampleDoppler" }
    { Synth.new(\monoCircularSampleDoppler,
      [\bufnum, otter_CostaRica, \rate, 1/16]) }
    { "MosquitoSampleDoppler" }
    { Synth.new(\stereoTriangularSampleDoppler,
      [\bufnum, mosquito, \amp, 0.75, \rate, 1/32]) }
    { "OtterFormantDoppler" }
    { Synth.new(\monoCircularFormantDoppler,
      [\bufnum, otter_CostaRica, \rate, 1/16]) }
    { "MosquitoFormantDoppler" }
    { Synth.new(\stereoTriangularFormantDoppler,
      [\bufnum, mosquito, \amp, 0.75, \rate, 1/32]) }
    // Default function:
    { "Software Error: invalid symbol!".postln };
    NodeWatcher.register(audition_node);
    audition_node // returns the node ID
  }
};

```

```

    }
    // else...
    { "\"" ++ symbol ++ "\" is not a valid symbol!" }
};

freeAudition = {
    var msg = nil;

    if(audition_node.isPlaying) {
        audition_node.free;
        msg = "sample freed!"
    };
    audition_node = nil;
    gui.freePlayButtons();
    gui.postMsg(msg.postln);
    "~"
};

setFreqModulation = { this.freqModLoop(wait_before:0, wait_after:30);
    "Frequency modulation set!" };
setPanModulation = { this.panModLoop; "Pan modulation set!" };
freeFreqModulation = { this.freeFreqMod; "Frequency modulation free!" };
freePanModulation = { this.freePanMod; "Pan modulation free!" };

this.programStart(this.class.filenameSymbol.asString.dirname)
}
}

/***** CLASS METHOD & FUNCTION CALLS [ FOR TESTING PURPOSES ]... *****/

/**

z = CZ_EternalOwlCall() // this boots the program and opens the GUI

// if you close and destroy the GUI window while audio synths are still running
// you can reopen the GUI window with one of the two following commands...

z.reopenGUI
CZ_EternalOwlCall.class_zpi_pointer.reopenGUI

z.programStart(CZ_EternalOwlCall.filenameSymbol.asString.dirname) // reboots
z.freeAllSynths

// Start/Restart Each Subpatch ...
// Patch #1 ... Atmospheric reverb is sent to the microphone signal
z.subpatch(1,01) // Ambient neotropical nature sounds fade in
z.subpatch(1,02) // Harmonized owl call emerges and freq mod is added
z.subpatch(1,03) // Pan modulation is added to the owl & frog sounds
z.subpatch(1,04) // Audio signal fades out over 142.5 seconds
z.subpatch(1,05) // Adds 7 delay taps to the atmospheric reverb channel

// Patch #2 ... Additive Synths...
z.subpatch(2,01) // 2nd, 3rd & 5th partials of C3
z.subpatch(2,02) // 5th, 3rd & 2nd partials of D3
z.subpatch(2,03) // 2nd, 3rd, 5th & 7th partials of Db2
z.subpatch(2,04) // 7th, 5th, 3rd & 2nd partials of Eb2
z.subpatch(2,05) // 2nd, 5th, 3rd & 7th partials of F2
z.subpatch(2,06) // 2nd, 3rd, 5th & 7th partials of G2
z.subpatch(2,07) // 7th, 11th & 13th partials of both C3 & D3
z.subpatch(2,08) // Adds a looped bass line of deep bell-like pitches
z.subpatch(2,09) // 17th, 19th & 23rd partials of both C3 & D3

```



```

z.subpatch(2,10) // Silences each synth in patch #2 & owl call xfades in
z.subpatch(2,11) // Removes reverb/delay from the mic input... END OF SECTION

// Patch #3 ...
z.subpatch(3,01) // Adds reverb & 12 delay taps to the synths and mic input
z.subpatch(3,02) // Adds a bass pulse & modulated sine waves
z.subpatch(3,03) // Adds a comb filter
z.subpatch(3,04) // xfades reverberated synths & modulated nature sounds
z.subpatch(3,05) // xfades nature sounds & reverberated/delayed owl call
z.subpatch(3,06) // Fades out to silence... END OF SECTION

// Free or Reverse Each Subpatch ...
// Patch #1 ...
z.freeSub(1,01) // | Frees PlayBuf synths from corresponding subpatch... |
z.freeSub(1,02) // |
z.freeSub(1,03) // Stops pan modulation and restores original pan values
z.freeSub(1,04) // Restarts all previous subpatches in patch #1
z.freeSub(1,05) // Removes 7-tap delay from the reverb channel
// Patch #2 ...
z.freeSub(2,01) // |
z.freeSub(2,02) // |
z.freeSub(2,03) // |
z.freeSub(2,04) // |
z.freeSub(2,05) // | Frees additive synths from corresponding subpatch... |
z.freeSub(2,06) // |
z.freeSub(2,07) // |
z.freeSub(2,08) // |
z.freeSub(2,09) // |
z.freeSub(2,10) // Stops owl call and clears routine to free additive synths
z.freeSub(2,11) // Adds 7-tap delay and reverb back onto the mic signal
// Patch #3 ...
z.freeSub(3,01) // Restores 7-tap delay/reverb with only a direct mic signal
z.freeSub(3,02) // Frees modular synths
z.freeSub(3,03) // Silences the comb filter
z.freeSub(3,04) // Restores ReverbGroupChannel & restarts modular synths...
z.freeSub(3,05) // Restarts the modulated nature sounds fade-in
z.freeSub(3,06) // Restarts all previous subpatches in patch #3

// Free modulation routines (subpatch #1.02, #1.03 & #3.04)...
z.setFreqModulation.()
z.setPanModulation.()
z.freeFreqModulation.()
z.freePanModulation.()

// Microphone & bus routing commands ...
z.directMicrophoneSignal.()
z.sendMicToBus_Reverb.()
z.sendMicToBus_Reverb__7TapDelay.()
z.sendMicToBus_Reverb_12TapDelay.()
z.sendMicToBus_Reverb_12TapDelay_Comb.()

// Playback & recording commands ...
z.playLastRecordedSession.()
z.startRecording.()
z.stopRecording.()

// Audio file auditioning commands ...
z.playSample.(symbol:"TawnyOwl" ,channels:1, rate:1, start_pos_secs:0, loop:1)
z.playSample.(symbol:"PigmyOwl" ,channels:2, rate:1, start_pos_secs:0, loop:1)
// start_pos_secs:285, to get 'colourful splashes' at end of otter file...
z.playSample.(symbol:"Otter" ,channels:1, rate:1, start_pos_secs:0, loop:1)

```

```

z.playSample.(symbol:"FrogAmazon",channels:1, rate:1, start_pos_secs:0, loop:1)
z.playSample.(symbol:"FrogPeru2" ,channels:1, rate:1, start_pos_secs:0, loop:1)
z.playSample.(symbol:"Mosquito" ,channels:2, rate:1, start_pos_secs:0, loop:1)
// Doppler auditioning commands ...
z.playDoppler.(symbol:"OtterSampleDoppler"      )
z.playDoppler.(symbol:"MosquitoSampleDoppler"   )
z.playDoppler.(symbol:"OtterFormantDoppler"     )
z.playDoppler.(symbol:"MosquitoFormantDoppler")
// free audition playback...
z.freeAudition.()

// Debug command for patch #2 ...
z.p2_DebugData

// Commands to post ServerOptions data ...
s.options.memSize
s.options.numAudioBusChannels
// Set the memSize & numAudioBusChannels size to twice the default:
s.options.memSize = 16384;
s.options.numAudioBusChannels = 256;

*/

/***** REDUNDANT/POTENTIALLY USEFUL CODE... *****/

/*
Routine
{  var message;

// this will play back the last recorded session...
SynthDef(\playSession, { arg buffer = 0;
Out.ar(0, DiskIn.ar(2, buffer)); }).send(s);

0.1.wait;
diskin_node = Synth.basicNew(\playSession);
message = { arg buffer;
diskin_node.addToHeadMsg( nil, [\buffer, buffer] ) };
diskin_buffer = Buffer.cueSoundFile(s, record_path, 0, 2,
completionMessage: message);
}.play;

*/

```



```

QCZ_EternalOwlCall_GUI : QWindow {
    classvar <server, <>class_gui_pointer;

    // Boolean Variables...
    var full_screen_before_new_window, pmode;
    // Integers...
    var row, col;
    // Main Window...
    var master_grid, gui_columns; /* 2D Array */
    var <zpi; // points to the corresponding Z_PerformanceInterface()
    // Non-Default Colors...
    var sky_blue, violet, orange, mid_blue, dark_blue;
    var light_purple, purple, dark_purple;
    // Fonts...
    var plain_font, bold_font;
    // Labels...
    var next_label, title_label, mode_label, rec_label, last_rec_label;
    var audition_label, doppler_label, mic_bus_label, modulation_label;
    var post_label, copyright_label;
    // Buttons...
    var next_button;
    // Arrays of Labels...
    var number_of_subs, patch_label, subpatch_label; /* 2D Array */
    var sample_string, sample_label, doppler_name_string, doppler_name_label;
    var mic_effect_string, mic_effect_label, mod_string, mod_label;
    // Arrays of Button Object Pointers + Functions & Arguments...
    var start_pointer; /* 2D Array */
    var rec_play_pointer; /* Single Value */
    var audio_play_pointer, audio_play_args;
    var doppler_play_pointer, doppler_play_args;
    var mic_bus_pointer, mic_bus_function;
    var mod_set_pointer, mod_set_function, mod_free_function;
    // Debug & Metering Views...
    var <>post_window, <>z_meter_view, <>z_scope_view;
    var <>z_analyzer_view, <>z_server_levels_panel_view;
    // Confirm Dialog Window...
    var dialog = nil;
    // Sub Info Window...
    var sub_info_window, sub_info_textview, sub_info_layout, info_path;
    // Mod Info Window
    var mod_info_window, mod_info_textview, mod_info_layout;
    // Key Responders...
    var space_ocr, esc_ocr, main_kres, main_ocr;
    var dialog_ocr, sub_info_ocr, mod_info_ocr;
    // Functions to pass as args for confirming options...
    var confirmPlayLastRecordedSession, confirmPlayAudition, confirmPlayEffect;
    var confirmFreeAudition, confirmModSet, confirmModFree;

    *initClass {
        StartUp.add {
            server = Server.default;
            class_gui_pointer = nil;
        }
    }

    *new { arg zpi_pointer,
        name = "Eternal Owl Call : Real-Time Performance Interface",
        bounds = Window.availableBounds, resizable = true, border = true,
        server = this.server, scroll = true;

        ^super.new.initQCZ_EternalOwlCall_GUI(

```

```

        zpi_pointer, name, bounds, resizable, border, server, scroll
    )
}

// Instance Methods...
addLabel { arg label = StaticText(), string = "Add String!",
    string_color = Color.green, font_size = 18,
    alignment = \center, bold = false;

    label.string = "  ++ string ++  ";
    label.background = Color.black;
    label.stringColor = string_color;
    label.align = alignment;
    label.font = Font(Font.default, font_size, bold);
}

addPlayLabels { arg label_array, string_array;

    label_array.size.do{ arg i;
        label_array[i] = StaticText();
        this.addLabel(label_array[i], string_array[i]++":",
            Color.green, 14, \right);
    }
}

addPlayButtons { arg label = StaticText(), column_offset = 0,
    play_string = "Play", replay_string = "Replay", stop_string = "Stop",
    pointer, play_function, play_args = [], stop_function,
    mod_buttons = false;

    var c = column_offset;
    gui_columns[c] = gui_columns[c].add(label);

    gui_columns[c + 1] = gui_columns[c + 1].add(
        Button().states_([[play_string , Color.blue, dark_purple],
            [replay_string , Color.blue, Color.black]])
        .font_(bold_font)
    );
    pointer = gui_columns[c + 1][ gui_columns[c + 1].size - 1 ];
    pointer.action = { arg b;
        // if(mod_buttons == false) { this.freePlayButtons() };
        // b.value = 1;
        this.postMessage(play_function.(*play_args).println);
    };

    gui_columns[c + 2] = gui_columns[c + 2].add(
        Button().states_([[stop_string , Color.gray, dark_blue]])
        .font_(bold_font)
    );
    gui_columns[c + 2][ gui_columns[c + 2].size - 1 ].action = {
        stop_function.(*play_args);
        // pointer.value = 0;
    };

    // return...
    ^pointer
}

startAction { |button, i, j|

    this.postMessage( zpi.subpatch( (i + 1), (j + 1) ) );
}

```

```

        button.value = 1; // remains on "Restart"
        "".postln;
        this.setNextLabel;
    }

    freeStartAction { |button, i, j|

        this.postMsg( zpi.freeSub( (i + 1), (j + 1) ).postln );
        button.value = 0;
        this.setNextLabel;
    }

    freeAllSynthsAction {

        this.postMsg(zpi.freeAllSynths.postln);

        // reset the start buttons for each subpatch...
        start_pointer.size.do{ arg i;
            start_pointer[i].size.do{ arg j;
                start_pointer[i][j].value = 0;
            }
        };
        this.freePlayButtons;
        mod_set_pointer.size.do{ arg i; mod_set_pointer[i].value = 0 };
    }

    setMode { arg value;
        switch(value)
        { 0 } {
            pmode = false;
            this.postMsg("Mode set to \"Testing\"").postln)
        }
        { 1 } {
            pmode = true;
            this.postMsg("Mode set to \"Performance\"").postln)
        }
    }

    setFullScreenBeforeNewWindow {
        case
        { gui_columns[8][0].value == 0 } { full_screen_before_new_window = true }
        { gui_columns[8][0].value == 1 } { full_screen_before_new_window = false };
        gui_columns[8][0].valueAction = 1;
    }

    openDialog { arg bounds = \default, message = \default,
        button_strings = \default, completion_function = nil;

        this.setFullScreenBeforeNewWindow;

        if( bounds == \default ) { bounds = Rect(500,300,205,150) };
        if( message == \default ) {
            message = "Are you sure that you wish to confirm this action?..."
        };
        if( button_strings == \default ) {
            button_strings = ["Confirm", "Cancel"]
        };

        if( dialog != nil ) { dialog.close };

        dialog = QZ_ConfirmDialog(button_strings);
    }

```

```

dialog.bounds = bounds;
dialog.message.string = message;
dialog.completion_function = [ completion_function, nil ];
dialog.view.keyDownAction = dialog_ucl;
dialog.onClose = {
    if(full_screen_before_new_window) { gui_columns[8][0].valueAction = 0 };
};
}

closeOtherWindows {

    if(dialog != nil) { dialog.close; dialog = nil };
    if(sub_info_window != nil) { sub_info_window.close; sub_info_window = nil };
    if(mod_info_window != nil) { mod_info_window.close; mod_info_window = nil };
}

postMsg { arg value, line_feed = true;
    var ln = "\n";

    if(line_feed == false) { ln = "" };

    post_window.setString( (value.asString ++ ln), post_window.string.size );
    post_window.stringColor = Color.white;
    // move the scrollbar to the end of the string...
    post_window.select( post_window.string.size, 0 );
    post_window.refresh;

    // return...
    ^value.asString
}

setRecordModeButton { arg mode = \on;
    switch(mode)
    { \on } { gui_columns[5][4].value = 0 }
    { \off } { gui_columns[5][4].value = 1 };
}

setMicBusButton { arg i;
    5.do{ arg j;
        if(i != j) { mic_bus_pointer[j].value = 0 }
    };
    mic_bus_pointer[i].value = 1;

    // function returns the button object pointer...
    ^mic_bus_pointer[i]
}

freeRecPlayButton { rec_play_pointer.value = 0; }

freePlayButtons {
    rec_play_pointer.value = 0;
    audio_play_pointer .size.do{ arg i; audio_play_pointer [i].value = 0 };
    doppler_play_pointer.size.do{ arg i; doppler_play_pointer[i].value = 0 };
}

setNextLabel {
    var unit = "";

    if( zpi.next_subpatch < 10, { unit = "0" } );
    next_label.string = "Next Subpatch = "
    ++ zpi.next_patch ++ "." ++ unit ++ zpi.next_subpatch;
}

```

```

        if( zpi.next_patch == number_of_subs.size
            && zpi.next_subpatch == (number_of_subs[ number_of_subs.size - 1 ] + 1) )
        { next_label.string = "Next Subpatch = END" }
    }

initQCZ_EternalOwlCall_GUI {
    arg zpi_pointer, name, bounds, resizable, border, server, scroll;

    if(server.pid == nil)
    {
        (
            "\nServer Process ID = nil\n"
            ++ "You may need to boot the server...\n").error;
        ^this
    };

    if(zpi_pointer.isKindOf(CZ_EternalOwlCall).not)
    {
        var msg =
            "\n'" ++ zpi_pointer ++ "' is an invalid zpi_pointer arg!\n"
            ++ "This must be an instance of the CZ_EternalOwlCall() class or a "
            ++ "similar compatible child class or Z_PerformanceInterafce() class.\n";

        // this.close;
        msg.error;
        ^this
    };

    /* Create a publically accessible pointer to the
       corresponding Z_PerformanceInterface() object... */
    zpi = zpi_pointer;
    if(zpi.valid_instance == false)
    {
        var msg =
            "\nUnable to open the GUI window because the "
            ++ "corresponding 'zpi_pointer' arg is invalid!\n";

        "".postln;
        msg.error;
        ^this
    };

    if(class_gui_pointer == nil)
    {
        // make the classvar point to the current QObject...
        class_gui_pointer = this;
    }
    // else...
    {
        var msg =
            "\nThe performance interface GUI window has already been opened!"
            ++ "\nYou cannot open multiple windows...\n";

        // this.close;
        "".postln;
        class_gui_pointer.postMessage(msg.error);
        ^this
    };

    // variable, object & function initializations:

```



```

// Boolean Variables...
full_screen_before_new_window = true;
pmode = false;

// Integers...
row = 0; col = 0;

// Main Window...
master_grid = GridLayout();
gui_columns = Array.newClear(9); // 2D Array

// Non-Default Colors...
sky_blue      = Color(0.6, 0.8, 0.8);
violet        = Color.blue.blend(Color.magenta, blend:0.5);
orange        = Color.red .blend(Color.yellow , blend:0.5);
mid_blue      = Color().darken(Color.blue , 0.25);
dark_blue     = Color().darken(Color.blue , 0.75);
light_purple  = Color().darken(Color.magenta, 0.25);
purple        = Color().darken(Color.magenta, 0.5 );
dark_purple   = Color().darken(Color.magenta, 0.75);

// Fonts...
plain_font = Font(Font.default, bold:false);
bold_font  = Font(Font.default, bold:true );

// Labels...
next_label      = StaticText();
title_label     = StaticText();
mode_label      = StaticText();
rec_label       = StaticText();
last_rec_label  = StaticText();
audition_label  = StaticText();
doppler_label   = StaticText();
mic_bus_label   = StaticText();
modulation_label = StaticText();
post_label      = StaticText();
copyright_label = StaticText();

// Buttons...
next_button = Button().states_([["Next Subpatch (SPACEBAR)",
    Color.blue, dark_purple]]).font_(bold_font);

// Arrays of Labels...
number_of_subs = [5,11,6];
patch_label    = Array.newClear(3);
subpatch_label = Array.newClear(3); // 2D Array
sample_string  = [
    "Tawny Owl Sample",
    "Pigmy Owl Sample",
    "Otter Sample",
    "Frog (Amazon) Sample",
    "Frog (Peru) Sample",
    "Mosquito Sample"
];
sample_label   = Array.newClear(6);
doppler_name_string = [
    "Otter Sample Doppler" ,
    "Mosquito Sample Doppler",
    "Otter Formant Doppler",
    "Mosquito Formant Doppler"
];

```

```

doppler_name_label = Array.newClear(4);
mic_effect_string = [
    "Direct Microphone Signal Only",
    "Microphone to Reverb Channel",
    "Mic to Reverb + 7 Tap Delay",
    "Mic to Reverb + 12 Tap Delay",
    "Reverb + 12 Tap Delay + Comb Filter"
];
mic_effect_label = Array.newClear(5);
mod_string = ["Frequency Modulation", "Pan Modulation"];
mod_label = Array.newClear(2);

// Arrays of Button Object Pointers + Functions & Arguments...
start_pointer = Array.newClear(3); // 2D Array
rec_play_pointer = nil; // Single Value

audio_play_pointer = Array.newClear(6);
audio_play_args = [
    ["TawnyOwl" , 1, 1, 0, 1],
    ["PigmyOwl" , 2, 1, 0, 1],
    ["Otter" , 1, 1, 0, 1],
    ["FrogAmazon" , 1, 1, 0, 1],
    ["FrogPeru2" , 1, 1, 0, 1],
    ["Mosquito" , 2, 1, 0, 1]
];

doppler_play_pointer = Array.newClear(4);
doppler_play_args = [
    "OtterSampleDoppler" ,
    "MosquitoSampleDoppler",
    "OtterFormantDoppler",
    "MosquitoFormantDoppler"
];

mic_bus_pointer = Array.newClear(5);
mic_bus_function = [
    zpi.directMicrophoneSignal,
    zpi.sendMicToBus_Reverb,
    zpi.sendMicToBus_Reverb_7TapDelay,
    zpi.sendMicToBus_Reverb_12TapDelay,
    zpi.sendMicToBus_Reverb_12TapDelay_Comb
];

mod_set_pointer = Array.newClear(2);
mod_set_function = [ zpi.setFreqModulation , zpi.setPanModulation ];
mod_free_function = [ zpi.freeFreqModulation, zpi.freePanModulation ];

// Debug & Metering Views...
post_window = TextView();
z_meter_view = QZ_MeterView().maxWidth_(151);
z_scope_view = QZ_ScopeView();
z_analyzer_view = QZ_AnalyzerView();
z_server_levels_panel_view = QZ_ServerLevelsPanelView().minWidth_(470);

z_scope_view.xZoom = 0.375;
z_scope_view.yZoom = 4.0;

// Confirm Dialog Window...
dialog = nil;

// Sub Info Window...

```

```

sub_info_window    = nil;
sub_info_textview  = nil;
sub_info_layout    = nil;

// Mod Info Window
mod_info_window    = nil;
mod_info_textview  = nil;
mod_info_layout    = nil;

// Key Responders...
space_ocr          = UnicodeResponder();
esc_ocr             = UnicodeResponder();
main_kres           = KeyResponder  ();
main_ocr            = UnicodeResponder();
dialog_ocr          = UnicodeResponder();
sub_info_ocr        = UnicodeResponder();
mod_info_ocr        = UnicodeResponder();

// Functions (to pass as arguments)...
confirmPlayLastRecordedSession = {
    // to send as a value, so must be a function and not a class method!

    if( pmode ) {

        // reset the previous button value prior to user confrimation...
        rec_play_pointer.value = switch(rec_play_pointer.value)
        { 0 } { 1 }
        { 1 } { 0 };

        this.openDialog(
            bounds: Rect(500,300,245,125),
            message:"Play the last recorded session?...",
            completion_function:{
                this.freePlayButtons;
                rec_play_pointer.value = 1;
                this.postMessage(zpi.playLastRecordedSession.().postln)
            }
        );
        dialog.message.align = \center;

        "Confirm?..."
    }
    // else...
    {
        this.freePlayButtons;
        rec_play_pointer.value = 1;
        zpi.playLastRecordedSession.()
    }
};

confirmPlayAudition = { arg i;
    // to send as a value, so must be a function and not a class method!

    if( pmode ) {

        // reset the previous button value prior to user confrimation...
        audio_play_pointer[i].value = switch(
            audio_play_pointer[i].value)
        { 0 } { 1 }
        { 1 } { 0 };
    }
};

```

```

        this.openDialog(
            bounds: Rect(500,300,285,125),
            message:"Audition the " ++ sample_string[i] ++ "?...",
            completion_function:{
                this.freePlayButtons;
                audio_play_pointer[i].value = 1;
                this.postMsg(zpi.playSample.(*audio_play_args[i])).postln()
            }
        );
        dialog.message.align = \center;

        "Confirm?..."
    }
    // else...
    {
        this.freePlayButtons;
        audio_play_pointer[i].value = 1;
        zpi.playSample.(*audio_play_args[i])
    };
};

confirmPlayEffect = { arg i;
    // to send as a value, so must be a function and not a class method!

    if( pmode ) {

        // reset the previous button value prior to user confrimation...
        doppler_play_pointer[i].value = switch(
            doppler_play_pointer[i].value)
        { 0 } { 1 }
        { 1 } { 0 };

        this.openDialog(
            bounds: Rect(500,300,210,140),
            message:"Audition the " ++ doppler_name_string[i]
            ++ " Effect?...",
            completion_function:{
                this.freePlayButtons;
                doppler_play_pointer[i].value = 1;
                this.postMsg(zpi.playDoppler.(*doppler_play_args[i])).postln()
            }
        );
        dialog.message.align = \center;

        "Confirm?..."
    }
    // else...
    {
        this.freePlayButtons;
        doppler_play_pointer[i].value = 1;
        zpi.playDoppler.(*doppler_play_args[i])
    };
};

confirmFreeAudition = {
    // to send as a value, so must be a function and not a class method!

    if( pmode ) {
        this.openDialog(
            bounds: Rect(500,300,210,125),
            message:"Free the audition synth?...",

```

```

        completion_function:{ zpi.freeAudition.() }
    );
    dialog.message.align = \center;

    "Confirm?..."
}
// else...
{ zpi.freeAudition.() };
};

confirmModSet = { arg i;
    // to send as a value, so must be a function and not a class method!

    if( pmode ) {

        // reset the previous button value prior to user confrimation...
        mod_set_pointer[i].value = switch(mod_set_pointer[i].value)
        { 0 } { 1 }
        { 1 } { 0 };

        this.openDialog(
            bounds: Rect(500,300,250,140),
            message:"Set the " ++ mod_string[i]
            ++ " for Subpatches #1.0" ++ (i + 2) ++ " & #3.04?...",
            completion_function:{
                mod_set_pointer[i].value = 1;
                this.postMsg(mod_set_function[i].()).postln
            }
        );
        dialog.message.align = \center;

        "Confirm?..."
    }
    // else...
    {
        mod_set_pointer[i].value = 1;
        mod_set_function[i].()
    };
};

confirmModFree = { arg i;
    // to send as a value, so must be a function and not a class method!

    if( pmode ) {

        this.openDialog(
            bounds: Rect(500,300,255,140),
            message:"Free the " ++ mod_string[i]
            ++ " for Subpatches #1.0" ++ (i + 2) ++ " & #3.04?...",
            completion_function:{
                mod_set_pointer[i].value = 0;
                this.postMsg(mod_free_function[i].()).postln
            }
        );
        dialog.message.align = \center;

        "Confirm?..."
    }
    // else...
    {
        mod_set_pointer[i].value = 0;

```

```

        this.postMessage(mod_free_function[i].()).postln)
    };
};

/*
    code realating to the extension of the class' QWindow instance
    (i.e. the 'this' instance)...
*/

// Set Main Window (i.e. the 'this' instance)...
GUI.qt;
this.initQWindow(name, bounds, resizable, border, scroll);
// this.background = violet; // Light background useful for grid debugging
this.background = Color.black;
this.alwaysOnTop_(false);
this.bounds_(Window.availableBounds); /* set initial bounds before
                                         calling full screen */
this.fullScreen;

// Set Path for Subpatch Info Window...
info_path = /* thisProcess.nowExecutingPath.dirname */
"" ++ zpi.dpath ++ "info.dir" ++ "subpatch.info";

// Reset the subpatch directors...
zpi.next_patch = 1; zpi.next_subpatch = 1;
zpi.last_patch = 1; zpi.last_subpatch = 0;

// Set Labels...
this.addLabel(next_label , "Next Subpatch = 1.01", Color.green, 17);
this.addLabel(title_label,
    "Eternal Owl Call : Real-Time Performance Interface",
    light_purple, 34, \topLeft);
this.addLabel(mode_label , "Mode of Operation =" ,
    Color.green, 17, \right);
this.addLabel(rec_label , "Record Mode =" ,
    Color.green, 17, \right);
this.addLabel(last_rec_label , "Last Recorded Session:",
    Color.green, 14, \right);
this.addLabel(audition_label , "Audio Sample Auditioning ..." ,
    Color.red , 17, bold:true);
this.addLabel(doppler_label , "Doppler Effect Auditioning ..." ,
    Color.red , 17, bold:true);
this.addLabel(mic_bus_label , "Microphone Bus Routing ..." ,
    Color.red , 17, bold:true);
this.addLabel(modulation_label, "Modulation Routine Settings ..." ,
    Color.red , 17, bold:true);
this.addLabel(post_label , "Post Window & Debugging ..." ,
    Color.blue, 17, bold:true);
this.addLabel(copyright_label , "Interface Copyright (c) : "
    ++ "Mr Gareth Olubunmi Hughes (1 July 2014 - Present)",
    purple, 15, \right);

number_of_subs.size.do{ arg i;
    subpatch_label[i] = Array.newClear(number_of_subs[i]);
    start_pointer [i] = Array.newClear(number_of_subs[i]);

    subpatch_label[i].size.do{ arg j;
        var unit = "";

        if( j < 9, { unit = "0" });
        subpatch_label[i][j] = StaticText();
    }
}

```

```

        this.addLabel(subpatch_label[i][j],
            (" " ++ (i+1) ++ "." ++ unit ++ (j+1)),
            Color.green, 15, \left);
    }
};

this.addPlayLabels(sample_label, sample_string);
this.addPlayLabels(doppler_name_label, doppler_name_string);
this.addPlayLabels(mic_effect_label, mic_effect_string);
this.addPlayLabels(mod_label, mod_string);

// Set Debug & Metering Views...
post_window.string =
"Server Sample Rate = " ++ QZ_AnalyzerView.sample_rate ++ "Hz\n"
++ "Server Process ID = " ++ server.pid ++ "\n\n";
post_window.editable = false;
post_window.font = Font.monospace(11, false);
post_window.stringColor = Color.white;
post_window.background = Color.black;

// post_window.string = "Post Window data will be displayed here...\n\n";
// post_window.focusColor = Color.green;
// post_window.canFocus = false;

// 1st Column...
gui_columns[0] = [ nil,nil ];
next_button.action = {
    if( zpi.next_patch == number_of_subs.size
        && zpi.next_subpatch == (
            number_of_subs[ number_of_subs.size - 1 ] + 1 )
    )
    { this.postMsg(
        "the piece has ended, there are no more subpatches!".postln)
    }
    // else...
    { start_pointer[ zpi.next_patch - 1 ][ zpi.next_subpatch - 1 ]
        .valueAction = 1
    }
};

// 2nd Column...
gui_columns[1] = [ nil,nil,nil ];

// 3rd Column...
gui_columns[2] = [ nil,nil,Button().states_
    ([["Info?...", Color.red.alpha_(0.75), Color.gray]])
    .font_(bold_font)
];
gui_columns[2][2].action = { arg b;
    var combined_kdrg = KeyDownResponderGroup(
        main_kres, main_ocr, sub_info_ocr, space_ocr
    );

    if(sub_info_window == nil)
    {
        this.setFullScreenBeforeNewWindow;

        sub_info_window = Window("Subpatch Info...", Rect(235,0,688,696),
            resizable:true, scroll:true);
        sub_info_textview = TextView();
        sub_info_layout = StackLayout(sub_info_textview);
    }
};

```

```

sub_info_textview.open(info_path);

sub_info_textview.font = Font.monospace(14, false);
sub_info_textview.background = Color.black;
sub_info_textview.stringColor = Color.magenta;
sub_info_textview.editable = false;
sub_info_textview.canFocus = false;
// sub_info_textview.syntaxColorize; // does not work in QtCollider!

sub_info_window.layout_(sub_info_layout);
sub_info_window.view.keyDownAction = combined_kdrg;
sub_info_window.alwaysOnTop_(true);
sub_info_window.visible = true;
sub_info_window.onClose = {
    if(full_screen_before_new_window) {
        gui_columns[8][0].valueAction = 0
    };
    sub_info_window = nil;
    sub_info_textview = nil;
    sub_info_layout = nil;
};
}
// else...
{
    this.postMessage(
        "the subpatch information window is already open!".inform)
}
};

// Columns 1, 2, 3 Merged...
row = 3;
subpatch_label.size.do{ arg i;
    patch_label[i] = StaticText();
    this.addLabel(patch_label[i], ("Patch # " ++ (i+1) ++ " ..."),
        Color.white, 17.5, bold:true);
    gui_columns[0] = gui_columns[0].add(nil);
    if( i > 0, {
        gui_columns[1] = gui_columns[1].add(nil);
        gui_columns[2] = gui_columns[2].add(Button().states_
            ([["Info?..."], Color.red.alpha_(0.75), Color.gray]])
            .font_(bold_font)
        );
    });
    subpatch_label[i].size.do{ arg j;
        gui_columns[0] = gui_columns[0].add(subpatch_label[i][j]);

        gui_columns[1] = gui_columns[1].add( Button().states_
            ([["Start" , Color.blue, dark_purple],
             ["Restart" , Color.blue, Color.black]]).font_(bold_font)
        );
        gui_columns[1][row].maxWidth = 73;
        gui_columns[1][row].action = { arg b;
            var unit = if(j < 9) { "0" } { "" };
            var suffix = "" ++ (i + 1) ++ "." ++ unit ++ (j + 1) ++ "?...";

            if( (b.value == 0) && pmode )
            {
                b.value = 1;
                this.openDialog(
                    bounds: Rect(500,300,210,125),

```



```

        message:"Restart subpatch #" ++ suffix,
        completion_function:{ this.startAction(b, i, j) }
    );
    dialog.message.align = \center;
}
// else...
{
    if(( zpi.next_patch != (i+1) || zpi.next_subpatch != (j+1))
        && pmode
    )
    {
        b.value = 0;
        this.openDialog(
            bounds: Rect(350,345,495,140),
            message:"You are attempting to start a subpatch"
            ++ " in the wrong sequential order!\n"
            ++ "Are you sure that you wish to start subpatch #"
            ++ suffix,
            completion_function:{ this.startAction(b, i, j) }
        );
        dialog.message.align = \center;
    }
    // else...
    { this.startAction(b, i, j) }
}
};

gui_columns[2] = gui_columns[2].add( Button().states_
    ([["Free" , Color.gray, dark_blue]]).font_(bold_font)
);
gui_columns[2][row].maxWidth = 73;
start_pointer[i][j] = gui_columns[1][row];
gui_columns[2][row].action = { arg b;

    if( pmode )
    {
        var unit = if(j < 9) { "0" } { "" };

        b.value = 1;
        this.openDialog(
            bounds: Rect(500,300,210,125),
            message:"Free subpatch #"
            ++ (i + 1) ++ "." ++ unit ++ (j + 1) ++ "?...",
            completion_function: {
                this.freeStartAction(start_pointer[i][j], i, j)
            }
        );
        dialog.message.align = \center;
    }
    // else...
    { this.freeStartAction(start_pointer[i][j], i, j) }
};
row = row + 1;
};
row = row + 1;
};

gui_columns[2][8] .action = { gui_columns[2][2].valueAction = 0 };
gui_columns[2][20].action = { gui_columns[2][2].valueAction = 0 };

// 4th Column...
gui_columns[3] = [ nil,nil,

```

```

        Button().states_([["Free All Synths", orange, dark_blue]])
        .font_(bold_font),
        nil, nil,
    ];
    gui_columns[3][2].minWidth = 130;
    gui_columns[3][2].action = {

        if( pmode ) {
            this.openDialog(
                bounds: Rect(500,300,210,125),
                message:"Free all synths?...",
                completion_function:{ this.freeAllSynthsAction() }
            );
            dialog.message.align = \center;
        }
        { this.freeAllSynthsAction() }
    };

    // 5th Column...
    gui_columns[4] = [ nil,nil,
        Button().states_([["Exit GUI", orange, dark_blue]])
        .font_(bold_font),
        nil, nil,
    ];
    gui_columns[4][2].action = { this.minimize };

    // 6th Column...
    gui_columns[5] = [ nil,nil,
        Button().states_([["Reboot Program" , orange, dark_blue]])
        .font_(bold_font),
        Button().states_([["TESTING" , Color.yellow, dark_blue],
            ["PERFORMANCE" , Color.blue, dark_purple]])
        .font_(bold_font),
        Button().states_([["On" , Color.red, mid_blue],
            ["Off" , Color.gray, dark_blue]])
        .font_(bold_font)
    ];

    gui_columns[5][2].maxWidth = 125;
    gui_columns[5][2].action = { arg b;

        this.openDialog(
            bounds: Rect(500,300,210,150),
            message:"Are you sure that you want to reboot the program?...",
            completion_function:{
                "Closing the GUI...".postln;
                this.close;
                zpi.programStart(zpi.dpath);
            }
        )
    };

    gui_columns[5][3].maxWidth = 125;
    gui_columns[5][3].action = { arg b;
        var string = switch(b.value)
        { 0 } { "\"Testing\"" }
        { 1 } { "\"Performance\"" };
        var target = b.value;

        // reset the previous button value prior to user confirmation...
        b.value = switch(b.value)

```

```

    { 0 } { 1 }
    { 1 } { 0 };

    this.openDialog(
        bounds: Rect(395,305,330,125),
        message:"Change the mode setting to " ++ string ++ "?...",
        completion_function:{
            this.setMode(target);
            b.value = target;
        }
    );
    dialog.message.align = \center;
};

gui_columns[5][4].maxWidth = 73;
gui_columns[5][4].action = { arg b;
    var flag;
    var function = case
    { b.value == 0 } {
        flag = "On";
        zpi.startRecording
    }
    { b.value == 1 } {
        flag = "Off";
        zpi.stopRecording
    };

    // reset the previous button value prior to user confrimation...
    b.value = switch(b.value)
    { 0 } { 1 }
    { 1 } { 0 };

    if( pmode ) {
        this.openDialog(
            bounds: Rect(500,300,210,125),
            message:"Switch Record Mode " ++ flag ++ "?...",
            completion_function:{ this.postMsg(function.().postln) }
        );
        dialog.message.align = \center;
    }
    // else...
    { this.postMsg(function.().postln) };
};

// Columns 4, 5, 6 Merged...
rec_play_pointer = this.addPlayButtons(last_rec_label, 3,
    pointer: rec_play_pointer,
    play_function: confirmPlayLastRecordedSession,
    stop_function: confirmFreeAudition
);
gui_columns[4][5].maxWidth = 73;
gui_columns[5][5].maxWidth = 73;
3.do{ arg i; gui_columns[i + 3] = gui_columns[i + 3].add(nil); };

sample_label.size.do{ arg i;
    audio_play_pointer[i] = this.addPlayButtons(sample_label[i], 3,
        pointer: audio_play_pointer[i],
        play_function: confirmPlayAudition,
        play_args: [i],
        stop_function: confirmFreeAudition
    );
};

```

```

        gui_columns[4][7 + i].maxWidth = 73;
        gui_columns[5][7 + i].maxWidth = 73;
    };
    3.do{ arg i; gui_columns[i + 3] = gui_columns[i + 3].add(nil); };

    doppler_name_label.size.do{ arg i;
        doppler_play_pointer[i] = this.addPlayButtons(doppler_name_label[i], 3,
            pointer: doppler_play_pointer[i],
            play_function: confirmPlayEffect,
            play_args: [i],
            stop_function: confirmFreeAudition
        );
        gui_columns[4][14 + i].maxWidth = 73;
        gui_columns[5][14 + i].maxWidth = 73;
    };

    7.do{ 2.do{ arg i; gui_columns[i + 3] = gui_columns[i + 3].add(nil); } };
    gui_columns[5] = gui_columns[5].add(nil);

    5.do{ arg i;
        gui_columns[5] = gui_columns[5].add(
            Button().states_([["Off" , Color.gray, dark_blue],
                ["On" , Color.red, mid_blue]]).font_(bold_font)
        );
        mic_bus_pointer[i] = gui_columns[5][19 + i];
        gui_columns[5][19 + i].action = { arg b;

            // reset the previous button value prior to user confrimation...
            mic_bus_pointer[i].value = switch(mic_bus_pointer[i].value)
            { 0 } { 1 }
            { 1 } { 0 };

            if( pmode ) {
                this.openDialog(
                    bounds: Rect(500,300,315,140),
                    message:"Change the Send Setting to \n\"
                    ++ mic_effect_string[i] ++ "\"?...\"",
                    completion_function:{
                        this.setMicBusButton(i);
                        mic_bus_function[i].().postln
                    }
                );
                dialog.message.align = \center;
            }
            // else...
            {
                this.setMicBusButton(i);
                mic_bus_function[i].().postln
            }
        };
        gui_columns[5][19 + i].maxWidth = 73;
    };
    gui_columns[5][20].value = 1;

    gui_columns[5] = gui_columns[5].add(
        Button().states_([["Further Info?..." ,
            Color.red.alpha_(0.75), Color.gray]]
            .font_(bold_font)
    );
    gui_columns[5][24].maxWidth = 110;
    gui_columns[5][24].action = { arg b;

```

```

var combined_kdrg = KeyDownResponderGroup(
    main_kres, main_ucr, mod_info_ucr, space_ucr
);

if(mod_info_window == nil)
{
    this.setFullScreenBeforeNewWindow();

    mod_info_window = Window("Modulation Info...",
        Rect(662,83,305,103), resizable:false, scroll:true
    );
    mod_info_textview = TextView();
    mod_info_layout = StackLayout(mod_info_textview);

    mod_info_textview.string =
        "These buttons randomly set or free "
    ++ "the values sent to the frequency & "
    ++ "panoramic modulation routines used "
    ++ "with subpatches #1.02, #1.03 & #3.04...";

    mod_info_textview.font = Font(Font.default, size:15, bold:true);
    // mod_info_textview.background = Color.black;
    mod_info_textview.stringColor = purple;
    mod_info_textview.editable = false;
    mod_info_textview.canFocus = false;

    mod_info_window.layout_(mod_info_layout);
    mod_info_window.view.keyDownAction = combined_kdrg;
    mod_info_window.alwaysOnTop_(true);
    mod_info_window.visible = true;
    mod_info_window.onClose = {
        if(full_screen_before_new_window) {
            gui_columns[8][0].valueAction = 0
        };
        mod_info_window = nil;
        mod_info_textview = nil;
        mod_info_layout = nil;
        combined_kdrg.free; combined_kdrg = nil;
    };
}
// else...
{
    this.postMessage(
        "the modulation information window is already open!".inform)
}
};

mod_label.size.do{ arg i;
    mod_set_pointer[i] = this.addPlayButtons(
        mod_label[i], 3, "Set", "Reset", "Free",
        pointer: mod_set_pointer[i],
        play_function: confirmModSet,
        play_args: [i],
        stop_function: confirmModFree,
        mod_buttons:true
    );
    gui_columns[4][25 + i].maxWidth = 73;
    gui_columns[5][25 + i].maxWidth = 73;
};

// 7th Column...

```

```

gui_columns[6] = [ nil,nil,nil ];

// 8th Column...
gui_columns[7] = [ nil,nil,nil ];

// 9th Column...
gui_columns[8] = [
    Button().states_(
        [ "EXIT Full Screen (F11)" , Color.green, dark_blue ],
        [ "ENTER Full Screen (F11)", Color.green, dark_blue ] ])
    .font_(bold_font),
    Button().states_(["Help (F1)", /*orange*/ Color.red,
        dark_blue])).font_(bold_font).minWidth_(95),
    Button().states_(["Clear Post Window", /*orange*/ Color.white,
        dark_blue])).font_(bold_font)
];

gui_columns[8][0].action = { arg b;
    case
    { b.value == 0 } {
        this.bounds_( Rect(100,100,100,100) );
        /* ensures that any window bounds that have gone outside
           of the viewable screen area are reduced */
        this.bounds_( Window.availableBounds );
        /* sets the bounds that will be required following
           a window.endFullScreen call in advance */
        this.fullScreen;
    }
    { b.value == 1 } {
        this.endFullScreen;
        this.bounds_( Window.availableBounds );
        /* not required, but wise to set the bounds to
           this value again from a logic perspective */
    }
};

gui_columns[8][1].action = {
    this.postMessage("Opening the help PDF file...".postln);
    (
        zpi.dpath
        ++ "help.dir"
        ++ "Eternal Owl Call User Guide_03.pdf"
    ).openOS
};

gui_columns[8][2].action = {
    post_window.string = "";
    post_window.focus(true);
    post_window.refresh;
};

// Columns 7, 8, 9 Merged...
7.do{ 3.do{ arg i; gui_columns[i + 6] = gui_columns[i + 6].add(nil) } };

gui_columns[6] = gui_columns[6].add(Button().states_(
    ["Patch # 2 Debug Data", /*orange*/ Color.white, dark_blue]))
    .font_(bold_font)
);
gui_columns[6][10].action = {
    this.postMessage(zpi.p2_DebugData.postln)
};

```

```

gui_columns[7] = gui_columns[7].add(Button().states_([
    ["Memory Size", /*orange*/ Color.white, dark_blue]])
    .font_(bold_font)
);
gui_columns[7][10].action = {
    this.postMessage(server.options.memSize.postln)
};

gui_columns[8] = gui_columns[8].add(Button().states_([
    ["Number of Audio Bus Channels", /*orange*/ Color.white, dark_blue]])
    .font_(bold_font)
);
gui_columns[8][10].action = {
    this.postMessage(server.options.numAudioBusChannels.postln)
};

gui_columns[6][10].maxWidth = 160;
gui_columns[7][10].maxWidth = 100;
gui_columns[8][10].maxWidth = 215;

// Add all cells manually according to the size of the gui grid array...
gui_columns.size.do{ arg i; // size of each column
    gui_columns[i].size.do{ arg j; // size of each row
        master_grid.add(gui_columns[i][j], j, i);
        if(gui_columns[i][j] != nil) { gui_columns[i][j].canFocus = false }
    }
};

// 1st Column (Spanned Cells)...
master_grid.addSpanning(next_button , row:0 , column:0, columnSpan:3);
master_grid.addSpanning(next_label , row:1 , column:0, columnSpan:3);
master_grid.addSpanning(patch_label[0], row:2 , column:0, columnSpan:2);
master_grid.addSpanning(patch_label[1], row:8 , column:0, columnSpan:2);
master_grid.addSpanning(patch_label[2], row:20, column:0, columnSpan:2);

// 4th Column (Spanned Cells)...
master_grid.addSpanning(title_label, row:0, column:3,
    rowSpan:2, columnSpan:5
);
master_grid.setAlignment(gui_columns[3][2], \right);
master_grid.addSpanning(mode_label , row:3, column:3, columnSpan:2);
master_grid.addSpanning(rec_label , row:4, column:3, columnSpan:2);
master_grid.addSpanning(audition_label , row:6 , column:3, columnSpan:3);
master_grid.addSpanning(doppler_label , row:13, column:3, columnSpan:3);
master_grid.addSpanning(mic_bus_label , row:18, column:3, columnSpan:3);
mic_effect_label.size.do{ arg i;
    master_grid.addSpanning(mic_effect_label[i], row:19+i, column:3,
        columnSpan:2
    );
};

master_grid.addSpanning(modulation_label, row:24, column:3, columnSpan:2);

// 7th Column (Spanned Cells)...
master_grid.setAlignment(gui_columns[8][1], \right);
master_grid.addSpanning(post_label , row:2, column:6, columnSpan:2);
master_grid.addSpanning(post_window, row:3, column:6,
    rowSpan:7, columnSpan:3);

master_grid.addSpanning(z_meter_view, row:11, column:6, rowSpan:5);
master_grid.setAlignment(z_meter_view, \right);
z_meter_view.master_settings_button/*states_

```

```

([ ["MASTER SETTINGS", Color.green, dark_blue] ])*/*.canFocus_(false);

master_grid.addSpanning(z_scope_view, row:11, column:7,
    rowSpan:5, columnSpan:2
);
z_scope_view.scope_label.stringColor_(Color.blue);
z_scope_view.scope_settings_button/*.*.states_
([ ["SCOPE SETTINGS", Color.green, dark_blue] ])*/*.canFocus_(false);

master_grid.addSpanning(z_analyzer_view, row:16, column:6,
    rowSpan:8, columnSpan:3
);
z_analyzer_view.db_readings_height_(142).freq_readings_width_(429);
z_analyzer_view.number_of_db_readings_(7);
z_analyzer_view.freqscope_label.align_(\left);
z_analyzer_view.freqscope_settings_button/*.*.states_
([ ["ANALYZER SETTINGS", Color.green, dark_blue] ])*/*.canFocus_(false);
z_analyzer_view.left_button.canFocus_(false);
z_analyzer_view.right_button.canFocus_(false);

master_grid.addSpanning(z_server_levels_panel_view, row:24, column:6,
    rowSpan:2, columnSpan:3
);
z_server_levels_panel_view.server_label.background_(dark_blue)
    .align_(\left);
master_grid.setAlignment(z_server_levels_panel_view, \right);
master_grid.addSpanning(copyright_label, row:26, column:6, columnSpan:3);

// Row & Column Stretching...
// master_grid.setRowStretch(1,1);
// master_grid.setColumnStretch(0,1);
master_grid.setColumnStretch(1,1);
master_grid.setColumnStretch(2,1);
master_grid.setColumnStretch(3,1);
master_grid.setColumnStretch(4,1);
master_grid.setColumnStretch(5,1);
master_grid.setColumnStretch(6,2);
master_grid.setColumnStretch(7,2);
master_grid.setColumnStretch(8,2);

// master_grid.setMinColumnWidth(3, 215);
master_grid.setMinColumnWidth(5, 150);
master_grid.setMinColumnWidth(6, 225);

// Set a UnicodeResponder for the main window:
// Ctrl+Backspace = Frees the last subpatch...
main_uqr.register(8, cntl:true, function:{
    var cell_skip = switch(zpi.last_patch)
    { 1 } { 2 }
    { 2 } { 8 }
    { 3 } { 20 };

    if( (zpi.last_patch != 1) || (zpi.last_subpatch != 0) )
    { gui_columns[2][cell_skip + zpi.last_subpatch].valueAction = 0 }
    // else...
    { this.postMessage("There are no previous subpatches to restart!".postln) }
});
// Ctrl+Esc = "Free All Synths"...
main_uqr.register(27, cntl:true, function:{
    gui_columns[3][2].valueAction = 0
});

```



```

// Ctrl+Shift+P = Clears the post window...
main_uqr.register(16, shift:true, cntl:true, function:{
    gui_columns[8][2].valueAction = 0
});
// Ctrl+B = Reboots the program...
main_uqr.register(2, cntl:true, function:{
    gui_columns[5][2].valueAction = 0
});
// Ctrl+Q = Quits the program...
main_uqr.register(17, cntl:true, function:{
    this.openDialog(
        bounds: Rect(500,300,210,125),
        message:"Quit the program?...",
        completion_function:{ this.close }
    );
    dialog.message.align = \center;
});
// Ctrl+. = Overrides the default Server.killAll command...
/*
    main_uqr.register(46, cntl:true, function:{
        this.postMessage( "crashed!".postln )
    });
*/
CmdPeriod.removeAll;
CmdPeriod.add({

    z_scope_view.scope_synth.free;
    z_scope_view.scope_synth = nil;
    z_meter_view.freeUGens;
    TempoClock.clear;
    this.close;
    if( CZ_EternalOwlCall.valid_instance_count > 0 ) {
        CZ_EternalOwlCall.valid_instance_count =
            CZ_EternalOwlCall.valid_instance_count - 1;
    };
    zpi.valid_instance = false;

    this.postMessage("").postln;
    this.postMessage( (
        "\n\nThe Z_PerformanceInterface() and QZ_PerformanceInterface_GUI() "
        ++ "objects were forcefully crashed using a Cmd+Period call!...\n"
        ++ "All active application windows have been closed "
        ++ "and all active UGens have been freed.").warn );
});
// Alt+I = Opens the subpatch information window...
main_uqr.register(105, opt:true, function:{
    gui_columns[2][2].valueAction = 0
});
/*
    Esc = Exits the GUI screen
    (sent to separate variable without keyword args)...
*/
esc_uqr.register(27, false, false, false, false, {
    gui_columns[4][2].valueAction = 0
});
/*
    Set the KeyResponder for the main window
    (i.e. for function key commands not supported
    by the UnicodeResponder class):
*/

```

```

// F1 = Opens the help .pdf file...
main_kres.register(67, function:{ gui_columns[8][1].valueAction = 0 });
// F11 = Exits/enters full screen mode...
main_kres.register(95, function:{
    var v = switch(gui_columns[8][0].value)
        { 0 } { 1 }
        { 1 } { 0 };

    gui_columns[8][0].valueAction = v;
});

/*
    Set a UnicodeResponder to override the
    default spacebar functionality in Qt...
*/
// Spacebar = Plays the next subpatch...
space_ocr.register(32, function:{
    next_button.focus(true);
    next_button.valueAction = 0;
});

// Set a UnicodeResponder for the dialog window:
// Retrun = "Confirm"...
dialog_ocr.register(13, function:{ dialog.button[0].valueAction = 0 });
// Esc = "Cancel"...
dialog_ocr.register(27, function:{ dialog.button[1].valueAction = 0 });

// Set a UnicodeResponder for the subpatch & modulation information window:
// Return = Closes the window...
sub_info_ocr.register(13, function:{ sub_info_window.close });
// Esc = Closes the window, overrides the Esc key function in main_ocr...
sub_info_ocr.register(27, function:{ sub_info_window.close });

// Set an identical UnicodeResponder for the modulation information window:
mod_info_ocr.register(13, function:{ mod_info_window.close });
mod_info_ocr.register(27, function:{ mod_info_window.close });

this.layout_(master_grid);
this.view.keyDownAction = KeyDownResponderGroup(
    main_kres, main_ocr, space_ocr, esc_ocr
);
post_window.keyDownAction = space_ocr;
this.onClose = {
    this.closeOtherWindows;
    CmdPeriod.removeAll;
    CmdPeriod.add({
        if( CZ_EternalOwlCall.valid_instance_count > 0 ) {
            CZ_EternalOwlCall.valid_instance_count =
                CZ_EternalOwlCall.valid_instance_count - 1;
        };
        zpi.valid_instance = false;
    });
    if(zpi != nil) { zpi.gui = nil };
    class_gui_pointer = nil;
};
this.front;

^this
}
}

```

```

/***** REDUNDANT/POTENTIALLY USEFUL CODE... *****/

/**

// old reboot function...
completion_function:{
    Routine {
        {
            this.postMessage("The program is booting, please wait...");
            zpi.programStart(zpi.dpath);
        }.defer;

        // reset the metering & oscilloscope views...
        7.wait;
        {
            z_meter_view = QZ_MeterView();

            z_scope_view.close;
            z_scope_view = QZ_ScopeView();
            master_grid.addSpanning(z_scope_view,
                row:11, column:7, rowSpan:5, columnSpan:2);
            z_scope_view.scope_label.stringColor_(Color.blue);
            z_scope_view.scope_settings_button.states_
                ([ ["SCOPE SETTINGS", Color.green, dark_blue] ]);

            this.postMessage("\nProgram initialized!".postln);
        }.defer
    }.play
}

*/

```

```

QZ_AnalyzerView : QView {
  classvar <server, <sample_rate;

  var freqscope_grid, <>freqscope_label;
  var freq_readings, <number_of_freq_readings;
  var <freq_readings_width, <freq_readings_height;
  var db_readings , <number_of_db_readings;
  var <db_readings_width , <db_readings_height;
  var <readings_color, nyquist_freq_limit;

  // relating to the display of instances of the FreqScopeView class...
  var freqscope_bounds;
  var <>freqscope_left_view , <>freqscope_left , <grid_color_left;
  var <>freqscope_right_view, <>freqscope_right, <grid_color_right;
  var <wave_color_left, <wave_color_right;
  var freqscope_stack, killFreqscopes;

  // Buttons...
  var <>freqscope_settings_button;
  var <>left_button;
  var <>right_button;

  *initClass { StartUp.add { server = Server.default; } }

  *new { arg parent = nil, bounds = Rect(750,450,160,300);
    ^super.new.initQZ_AnalyzerView(parent, bounds)
  }

  // printOn { arg stream; } // overrides unhelpful post window messages

  setLabel { arg label = QStaticText(), string = "Add String!",
    string_color = Color.green, font_size = 17.5, alignment = \center,
    background_color = Color.black;

    // label.string = "  "++ string ++"  ";
    label.string = string;
    label.background = background_color;
    label.stringColor = string_color;
    label.align = alignment;
    label.font = Font(Font.default, font_size, bold:true);

    ^label
  }

  formatFreqValue { arg float, unit_suffix = "", suffix = false;
    var string, display_value;

    if(float >= 1000)
    { display_value = float/1000 }
    // else...
    { display_value = float };

    display_value = case
    { float < 1000 } { display_value.round(1 ) }
    { (float >= 1000) && (float < 10000) } { display_value.round(0.1) }
    { float >= 10000 } { display_value.round(1 ) };

    // in case display_value has been rounded to 1000...
    if (display_value == 1000) { string = "1k"; }
    // else...
    {

```

```

        if(float >= 1000)
        { string = display_value.asString ++ "k" }
        // else...
        { string = display_value.asString };
    };

    if(suffix) { string = string ++ unit_suffix };

    ^string
}

drawReadings { arg view, width, height, number_of_readings = 5,
    unit_suffix = "", last_string = true, min_reading = 0, max_reading = 100,
    direction = \horizontal, shape = \lin, reading_position = \tail,
    line_color = grid_color_left;

    view.drawFunc = { var last_string_point;
        QPen.fillColor    = readings_color;
        QPen.strokeColor = line_color;
        QPen.font = Font(Font.default, 11, bold:true);

        number_of_readings.do{ arg i;
            var suffix = false;
            var line_indent = 0, h_string_indent = 0, v_string_indent = 0;
            var reading, string_point, line_p1, line_p2;
            var factor = 1/(number_of_readings - 1);
            var fft_scaling_points = 2.pow(10);
            /* i.e. there are 2048 FFT buffer points with two (positive and
            negative) used for scoping each individual frequency reading...
            The readings are scaled logarithmically to 1023 (i.e. 2.pow(10)
            - 1) in order for the range to be between zero and the Nyquist
            frequency limit... */

            reading = case
            { shape == \lin } {
                max_reading * i/(number_of_readings - 1)
            }
            { shape == \log } {
                ( (fft_scaling_points.pow(i * factor) - 1)/
                (fft_scaling_points - 1) * nyquist_freq_limit )
            };

            if(i == 0) {
                suffix = true;
                line_indent = case
                { direction == \horizontal } { 1 }
                { direction == \vertical } { 2 };
                if(direction == \horizontal) { h_string_indent = 1 };
                v_string_indent = -2;
            };
            if(i == (number_of_readings - 1)) {
                line_indent = case
                { direction == \horizontal } { -1 }
                { direction == \vertical } { 0 };
            };

            string_point = case
            { direction == \horizontal } {
                Point( ((width * i * factor) + 2) + h_string_indent, height - 15 )
            }
            { direction == \vertical } {

```

```

        Point( width - 22 + h_string_indent,
              ( (height * i * factor) - 2 - v_string_indent) )
    };

    line_p1 = case
    { direction == \horizontal } {
        Point( (width * i * factor) + line_indent, 0)
    }
    { direction == \vertical    } {
        Point(0, (height * i * factor) - 1 + line_indent)
    };

    line_p2 = case
    { direction == \horizontal } {
        Point( (width * i * factor) + line_indent, height)
    }
    { direction == \vertical    } {
        Point(width, (height * i * factor) - 1 + line_indent)
    };

    last_string_point = case
    { direction == \horizontal } { Point(width - 22, height - 16) }
    { direction == \vertical    } { Point(width - 22, height - 16) };

    QPen.stringAtPoint( this.formatFreqValue
                       (reading, unit_suffix, suffix), string_point );
    QPen.line( line_p1, line_p2 );
};

if(last_string == true) {
    QPen.stringAtPoint( this.formatFreqValue(max_reading),
                      last_string_point);
};
QPen.fillStroke;
};

^view
}

setReadingsLineColor { arg line_color = grid_color_left;
    freq_readings = this.drawReadings(
        freq_readings,
        width: freq_readings_width,
        height: freq_readings_height,
        number_of_readings: number_of_freq_readings,
        unit_suffix: "Hz",
        last_string: false,
        max_reading: nyquist_freq_limit,
        shape: \log,
        line_color: line_color
    );
    freq_readings.refresh;

    db_readings = this.drawReadings(
        db_readings,
        width: db_readings_width,
        height: db_readings_height,
        number_of_readings: number_of_db_readings,
        unit_suffix: "dB" ,
        last_string: false,
        max_reading: -96,

```

```

        direction:\vertical,
        line_color:line_color
    );
    db_readings.refresh;

    ^this
}

setFreqscopes { arg index = 0;
    freqscope_left_view.background_(grid_color_left)
    .maxWidth_(freq_readings_width + 1).maxHeight_(db_readings_height + 1);
    freqscope_left = FreqScopeView(freqscope_left_view , freqscope_bounds);
    // freqscope_left.active = true;
    freqscope_left.freqMode = 1;
    freqscope_left.waveColors = wave_color_left;

    freqscope_right_view.background_(grid_color_right)
    .maxWidth_(freq_readings_width + 1).maxHeight_(db_readings_height + 1);
    freqscope_right = FreqScopeView(freqscope_right_view, freqscope_bounds);
    freqscope_right.inBus = 1;
    // freqscope_right.active = true;
    freqscope_right.freqMode = 1;
    freqscope_right.waveColors = wave_color_right;

    case
    { index == 0 } { freqscope_left.active = true }
    { index == 1 } { freqscope_right.active = true };
    freqscope_stack.index = index;
}

initQZ_AnalyzerView { arg parent, bounds;

    // variable and object initializations...
    this.bounds = bounds;

    freqscope_label = QStaticText();
    freq_readings    = QUIView();
    number_of_freq_readings = 12;
    freq_readings_width  = 400;
    freq_readings_height = 15;
    db_readings         = QUIView();
    number_of_db_readings = 7;
    db_readings_width   = 25;
    db_readings_height  = 150;
    readings_color = Color.red;

    freqscope_bounds = Rect(0,0, freq_readings_width, db_readings_height);
    freqscope_left_view  = QVIEW();
    grid_color_left      = Color.yellow;
    freqscope_right_view = QVIEW();
    grid_color_right     = Color.cyan;
    wave_color_left      = [ Color.magenta ];
    wave_color_right     = [ Color.green  ];
    freqscope_stack = QStackLayout(freqscope_left_view, freqscope_right_view);
    killFreqscopes = {
        freqscope_left.kill ; // you must have this on a freqscope instance!
        freqscope_right.kill;
    };

    freqscope_settings_button = QButton().maxHeight_(20).maxWidth_(170).
    states = [ ["ANALYZER SETTINGS", Color.blue, Color.green] ];

```

```

left_button = QPushButton().maxWidth_(50).states = [
    ["L", Color.magenta, Color.blue.alpha_(0.25)]];
right_button = Button().maxWidth_(50).states = [
    ["R", Color.green, Color.blue.alpha_(0.25)]];

sample_rate = FreqScopeView.server.sampleRate;
if(sample_rate != nil)
{ ("Server Sample Rate = " ++ sample_rate ++ "Hz").inform }
// else...
{
    (
        "\n"
        ++ "An instance of the QZ_AnalyzerView class "
        ++ "could not locate a valid server!\n"
        ++ "You may need to boot the server...\n").error;
    ^this
};
nyquist_freq_limit = sample_rate/2;

// code realating to the extension of the class' QView instance
// (i.e. the 'this' instance)...
this.setLabel(freqscope_label, "Frequeuncy Analyzer ..." ,
    Color.blue, 17, \center);
left_button.font = Font.new(Font.default, 35, bold:true);
right_button.font = Font.new(Font.default, 35, bold:true);

freq_readings.background = Color.black;
db_readings .background = Color.black;
this.setReadingsLineColor(grid_color_left);
this.setFreqscopes(0);

freqscope_grid = QGridLayout.rows(
    [ [freqscope_label, columns:3], nil, nil, freqscope_settings_button ],
    [ [left_button, rows:2], nil, [freq_readings, columns:2] ],
    [ nil, [db_readings, rows:2], [freqscope_stack, rows:2, columns:2] ],
    [ right_button ]
);

freqscope_stack.index = 0;
left_button .action = {
    this.setReadingsLineColor(grid_color_left);
    freqscope_stack.index = 0;
    if(freqscope_left.active.not) { freqscope_left .active_(true); };
    if(freqscope_right.active ) { freqscope_right.active_(false); };
};
right_button.action = {
    this.setReadingsLineColor(grid_color_right);
    freqscope_stack.index = 1;
    if(freqscope_right.active.not) { freqscope_right.active_(true); };
    if(freqscope_left .active ) { freqscope_left .active_(false); };
};

freq_readings.maxHeight_(freq_readings_height);
db_readings.maxWidth_(db_readings_width);

this.background = Color.black;
this.layout_(freqscope_grid);
this.onClose_(killFreqscopes);

this.initQView(parent);
if(parent != nil) { parent.layout = this.layout };

```



```

    ^this
}

refreshReadings { arg index = 0;
    var color = grid_color_left;

    if(freqscope_stack.index == 1) { color = grid_color_right };
    this.setReadingsLineColor(color);

    killFreqscopes.value;
    freqscope_left .destroy;
    freqscope_right.destroy;
    freqscope_left = nil;
    freqscope_right = nil;
    freqscope_bounds = Rect(0,0, freq_readings_width, db_readings_height);
    this.setFreqscopes(index);
}

number_of_freq_readings_ { arg n;
    number_of_freq_readings = n;
    this.refreshReadings(freqscope_stack.index)
}

number_of_db_readings_ { arg n;
    number_of_db_readings = n;
    this.refreshReadings(freqscope_stack.index)
}

freq_readings_width_ { arg width;
    freq_readings_width = width;
    this.refreshReadings(freqscope_stack.index)
}

freq_readings_height_ { arg height;
    freq_readings_height = height;
    this.refreshReadings(freqscope_stack.index)
}

db_readings_width_ { arg width;
    db_readings_width = width;
    this.refreshReadings(freqscope_stack.index)
}

db_readings_height_ { arg height;
    db_readings_height = height;
    this.refreshReadings(freqscope_stack.index)
}

readings_color_ { arg color;
    readings_color = color;
    this.refreshReadings(freqscope_stack.index)
}

grid_color_left_ { arg color;
    grid_color_left = color;
    this.refreshReadings(freqscope_stack.index)
}

grid_color_right_ { arg color;
    grid_color_right = color;
    this.refreshReadings(freqscope_stack.index)
}

```

```

    }

    wave_color_left_ { arg color;
        wave_color_left = color;
        this.refreshReadings(freqscope_stack.index)
    }

    wave_color_right_ { arg color;
        wave_color_right = color;
        this.refreshReadings(freqscope_stack.index)
    }
}

QZ_Analyzer : QWindow {
    var window_layout, <>z_analyzer_view;

    *new { arg name = "Z_Analyzer", bounds = Rect(750,450,536,231),
        resizable = false, border = true, server, scroll = false;

        ^super.new.initQZ_Analyzer(name, bounds, resizable, border, server, scroll)
    }

    initQZ_Analyzer { arg name, bounds, resizable, border, server, scroll;

        this.initQWindow(name, bounds, resizable, border, scroll);

        z_analyzer_view = QZ_AnalyzerView.new(bounds:bounds);
        // window_layout = QGridLayout.rows([z_analyzer_view]);
        window_layout = QStackLayout(z_analyzer_view);
        this.background = Color.green; // this color should be hidden
                                        // by the child view!

        this.alwaysOnTop = true;
        this.layout_(window_layout);
        this.front;
        ^this
    }
}

```



```

QZ_ConfirmDialog : QWindow {
    classvar <server;

    var dialog_grid, <purple, message_string, <>message;
    var <>button_string_array, <>string_color_array;
    var <>button, <>completion_function, <>args; // Arrays

    *initClass { StartUp.add { server = Server.default; } }

    *new { arg
        button_strings      = ["Confirm" , "Cancel" ],
        button_string_colors = [Color.blue, Color.red],
        name = "Confirm Dialog...", bounds = Rect(500,300,205,150),
        resizable = true, border = false, server, scroll = false;

        ^super.new.initQZ_ConfirmDialog(button_strings, button_string_colors,
            name, bounds, resizable, border, server, scroll)
    }

    initQZ_ConfirmDialog { arg button_strings, button_string_colors,
        name, bounds, resizable, border, server, scroll;

        // variable and object initializations...
        button_string_array = button_strings;
        string_color_array = button_string_colors;
        this.initQWindow(name, bounds, resizable, border, scroll);

        dialog_grid = GridLayout();
        purple = Color().darken(Color.magenta, 0.5 );
        message_string = "Are you sure that you wish to confirm this action?...";
        message = StaticText().string_(message_string).stringColor_(purple)
            .font_( Font(Font.default, 15.25, bold:true) );

        button          = Array.newClear(button_string_array.size);
        completion_function = Array.newClear(button_string_array.size);
        args             = Array.newClear(button_string_array.size);

        /* code realating to the extension of the class' QWindow instance
           (i.e. the 'this' instance)... */

        // create whitespace for the message text...
        dialog_grid.add( nil, 1, 0 );

        button_string_array.size.do{ arg i;

            completion_function[i] = {
                var msg = (
                    "A 'completion_function["++i++"]' "
                    ++ "value must be set for the \""
                    ++ button_string_array[i]
                    ++ "\" dialog button to work correctly!\n"
                    ++ "If no action is required the function "
                    ++ "value should be set to nil.\n"
                    ++ "Arguments can be supplied by setting "
                    ++ "a value for 'args["++i++ "]"'..."
                );

                msg.inform
            };

            args[i] = [];

```

```

        button[i] = Button().states_([ [button_string_array[i],
            string_color_array[i] ] ]).action = {
            this.close;
            completion_function[i].(*args[i])
        };
        button[i].maxWidth_(80);
        dialog_grid.add( button[i], 1, (i + 1) );
    };

    // create further whitespace for the message text...
    dialog_grid.add( nil, 1, (button_string_array.size + 1) );

    dialog_grid.addSpanning( message, 0, 0,
        columnSpan:(button_string_array.size + 2) );

    this.alwaysOnTop = true;
    this.background = Color.white;
    this.layout_(dialog_grid);
    this.front;

    ^this
}
}

```

```

QZ_MeterView : QView {
  classvar <server;

  var meter_grid;
  var <>input_bus , <>input_signal , <>input_level , <>input_level_routine ;
  var <>output_bus, <>output_signal, <>output_level, <>output_level_routine;
  var <>master_settings_button;
  var <>in_label, <>out_label, <>max_label, <>min_label;
  var <>separator_line;

  *initClass {
    StartUp.add {
      server = Server.default;
      this.initSynthDefs;
    }
  }

  *new { arg parent = nil, bounds = Rect(750,450,160,300);
    ^super.new.initQZ_MeterView(parent, bounds)
  }

  *initSynthDefs {
    /*
      outputs a control-rate ugen from an input signal
      to a bus so that the signal's value can be read...
    */
    SynthDef(\monoOutput, { |bus, channel|
      Out.kr( bus, Amplitude.kr( In.ar(channel) ) )
    }).add;
  }

  // update the indicator's value with a routine...
  updateIndicator { arg level_indicator, bus, routine;
    routine = Routine
    {
      {
        {
          bus.get({
            // get current value from the bus
            arg value; {
              // value.postln;
              // set indicator's value...
              level_indicator.value_(
                value.ampdb.linlin(-80, 0, 0, 1) );
              // set indicator's peak value
              level_indicator.peakLevel_(
                value.ampdb.linlin(-120, 0, 0, 1) );
            }.defer(); // schedule in the AppClock
          });
          0.1.wait; // indicator will be updated every 0.1 seconds
        }.loop
      }.fork
    };
    AppClock.play(routine);

    ^level_indicator
  }

  addLevelIndicator { arg bus, routine, critical = 0.0; // critical must be a float!
    var level_indicator = LevelIndicator(bounds:Rect(10, 10, 20, 160));

```

```

    level_indicator.numTicks = 9;
    level_indicator.numMajorTicks = 3;
    level_indicator.drawsPeak = true;
    level_indicator.warning = -2.dbamp;
    level_indicator.critical = /*-0.001.dbamp*/ critical.dbamp;
    this.updateIndicator(level_indicator, bus);

    ^level_indicator
}

setLabel { arg label = StaticText(), string = "Add String!",
    string_color = Color.green, font_size = 17.5, alignment = \center,
    background_color = Color.black;

    label.string = "  ++ string ++ ";
    label.background = background_color;
    label.stringColor = string_color;
    label.align = alignment;
    label.font = Font(Font.default, font_size, bold:true);

    ^label
}

freeUGens {
    // Free LevelIndicator routines and
    // ugens when the window is closed...
    input_signal.size.do { arg i;
        input_level[i].free;
        input_level[i] = nil;
        if(input_signal[i].isPlaying) {
            input_signal[i].free;
            input_signal[i] = nil;
        };
        input_level_routine[i].stop;
        input_level_routine[i].free;
        input_level_routine[i] = nil;
        input_bus[i].free;
        input_bus[i] = nil;
    };
    output_signal.size.do{ arg i;
        output_level[i].free;
        output_level[i] = nil;
        if(output_signal[i].isPlaying) {
            output_signal[i].free;
            output_signal[i] = nil;
        };
        output_level_routine[i].stop;
        output_level_routine[i].free;
        output_level_routine[i] = nil;
        output_bus[i].free;
        output_bus[i] = nil;
    }
}

freeSynths {
    input_signal.size.do { arg i;
        input_signal[i].free;
        input_signal[i] = nil;
        output_signal[i].free;
        output_signal[i] = nil;
    }
}

```

```

}

initQZ_MeterView { arg parent, bounds;

    // variable and object initializations...
    this.bounds = bounds;

    master_settings_button = Button().maxHeight_(20).maxWidth_(140).states = [
        ["MASTER SETTINGS", Color.blue, Color.green]
    ];
    in_label = StaticText();
    out_label = StaticText();
    max_label = StaticText();
    min_label = StaticText();
    separator_line = View();

    input_bus          = Array.newClear(2);
    output_bus         = Array.newClear(2);
    input_signal        = Array.newClear(2);
    input_level         = Array.newClear(2);
    input_level_routine = Array.newClear(2);
    output_signal       = Array.newClear(2);
    output_level        = Array.newClear(2);
    output_level_routine = Array.newClear(2);

    /* code realating to the extension of the class' QView instance
       (i.e. the 'this' instance)... */

    // write amplitude data to the control bus arrays...
    input_bus.size.do{ arg i;
        input_bus[i] = Bus.control(server, numChannels:1);
        input_signal[i] = {
            Out.kr(input_bus[i], Amplitude.kr(SoundIn.ar(i)))
        }.play;
        NodeWatcher.register(input_signal[i]);
    };
    output_bus.size.do{ arg i;
        output_bus[i] = Bus.control(server, numChannels:1);

        // 0.1.wait;
        output_signal[i] = Synth.new(\monoOutput,
            [\bus, output_bus[i], \channel, i],addAction:\addToTail);
        /* \addToTail so that synths already playing
           in the RootNode will be read */
        NodeWatcher.register(output_signal[i]);
    };
    // Add the LevelIndicators...
    input_level.size.do{ arg i;
        input_level[i] = this.addLevelIndicator(
            bus:input_bus[i],
            routine:input_level_routine[i]
        )
    };
    output_level.size.do{ arg i;
        output_level[i] = this.addLevelIndicator(
            bus:output_bus[i],
            routine:output_level_routine[i],
            critical:-0.001)
        /* the output meters will go red to indicate
           a clip if 0.001 db is exceeded */
    };
}

```



```

this.setLabel(in_label , "Inputs" , Color.black, 12, \left, Color.gray);
in_label.string = "Inputs"; /* removes some of the whitespace
                             on the label string */

in_label.maxHeight = 15;
this.setLabel(out_label, "Outputs", Color.black, 12, \left , Color.gray);
out_label.string = "Outputs";
out_label.maxHeight = 15;
this.setLabel(max_label, "0", Color.black, 12, \topRight , Color.gray);
max_label.string = "0";
this.setLabel(min_label, "-80", Color.black, 12, \bottomRight, Color.gray);
min_label.string = "-80";

separator_line.background = Color.blue;

meter_grid = GridLayout.rows(
    [ [master_settings_button, columns:6] ],
    [ nil, [in_label, columns:2], nil, nil,
      [out_label, columns:2], nil ],
    [ [max_label],
      [input_level[0], rows:2],
      [input_level[1], rows:2],
      [separator_line.maxWidth_(5), rows:2],
      [output_level[0], rows:2],
      [output_level[1], rows:2] ],
    [ min_label ]
);

meter_grid.setAlignment(master_settings_button, \center);

this.maxWidth_(250 /*150*/ ).maxHeight_(150);

meter_grid.setColumnStretch(0,1);
meter_grid.setColumnStretch(1,2);
meter_grid.setColumnStretch(2,2);
meter_grid.setColumnStretch(3,1);
meter_grid.setColumnStretch(4,2);
meter_grid.setColumnStretch(5,2);

this.background = Color.gray;
this.layout_(meter_grid);
this.onClose_({
    this.freeUGens;
    TempoClock.clear;
    /* clears any actions scheduled by the AppClock, however calling
       the AppClock.clear method should be avoided as it will crash
       any future events from being scheduled on the default AppClock
       pending a full server reboot...
    */
});

this.initQView(parent);
if(parent != nil) { parent.layout = this.layout };

^this
}
}

QZ_Meter : QWindow {
    var window_layout, <>z_meter_view;

```

```

*new { arg name = "Z_Meter", bounds = Rect(750,450,254,154),
    resizable = false, border = true, server, scroll = true;

    ^super.new.initQZ_Meter(name, bounds, resizable, border, server, scroll)
}

initQZ_Meter { arg name, bounds, resizable, border, server, scroll;

    this.initQWindow(name, bounds, resizable, border, scroll);

    z_meter_view = QZ_MeterView.new(bounds:bounds);
    // window_layout = QGridLayout.rows([z_meter_view]);
    window_layout = QStackLayout(z_meter_view);
    this.background = Color.blue; // this color should be hidden
                                   // by the child view!

    this.alwaysOnTop = true;
    this.layout_(window_layout);
    this.onClose = { z_meter_view.freeSynths };
    this.front;
    ^this
}
}

```

```

QZ_ScopeView : QView {
  classvar <server;

  var scope_grid;
  var <>scope_label, <>left_label, <>right_label;
  var <>scope_settings_button;
  var <>scope_synth, <>scope_display, <>scope_buffer;
  var <xZoom, <yZoom;

  *initClass {
    StartUp.add {
      server = Server.default;
      this.initSynthDefs;
    }
  }

  *new { arg parent = nil, bounds = Rect(750,450,160,300);
    ^super.new.initQZ_ScopeView(parent, bounds)
  }

  *initSynthDefs {
    /*
      reads values from an audio bus, using ScopeOut2
      to write it to a ScopeBuffer...
    */
    SynthDef(\monoscope, { arg bus = 0, bufnum, zoom = 1.0;
      var input;
      input = In.ar(bus, 2);
      // ScopeOut2 writes the audio to the buffer
      ScopeOut2.ar( input, bufnum, 4096, 1024.0/zoom )
    }).add
  }

  setlabel { arg label = StaticText(), string = "Add String!",
    string_color = Color.green, font_size = 17.5, alignment = \center,
    background_color = Color.black;

    // label.string = "  ++ string ++  ";
    label.string = string;
    label.background = background_color;
    label.stringColor = string_color;
    label.align = alignment;
    label.font = Font(Font.default, font_size, bold:true);

    ^label
  }

  freeScopeSynth {
    scope_synth.free;
    scope_synth = nil;
  }

  xZoom_ { arg float = 1.0;
    xZoom = float;
    scope_display.xZoom = float;
    // scope_synth.set( \cycle, (1024 * float).asFloat.reciprocal );
    scope_synth.set( \zoom, float );
  }

  yZoom_ { arg float = 1.0;
    yZoom = float;

```

```

    scope_display.yZoom = float;
}

initQZ_ScopeView { arg parent, bounds;

    // variable and object initializations...
    this.bounds = bounds;

    scope_label = StaticText();
    left_label = StaticText();
    right_label = StaticText();
    scope_settings_button = Button().maxHeight_(20).maxWidth_(140).states = [
        ["SCOPE SETTINGS", Color.blue, Color.green]];
    scope_buffer = ScopeBuffer.alloc(server, 2);

    /* code realating to the extension of the class' QView instance
       (i.e. the 'this' instance)... */
    this.setLabel(scope_label, "Oscilloscope ..." , Color.red, 17, \left);
    this.setLabel(left_label , "L " , Color.yellow, 35, \left);
    this.setLabel(right_label, "R " , Color.cyan , 35, \left);

    scope_display = QScope2();
    scope_display.waveColors = [Color.yellow, Color.cyan];
    scope_display.bufnum = scope_buffer.bufnum;
    scope_display.server_(server);

    // listening to the bus, using ScopeOut2 to write it to the buffer
    scope_synth = Synth.new(\monoscope,
        args:[\bus, 0, \bufnum, scope_buffer.bufnum],
        target:RootNode(server),
        addAction:\addToTail
        // make sure it goes after what you are scoping
    );
    NodeWatcher.register(scope_synth);

    this.xZoom = 1.5;
    this.yZoom = 2.5;

    scope_grid = GridLayout.rows(
        [ [scope_label, columns:2], nil, scope_settings_button ],
        [ left_label, [scope_display, rows:2, columns:2] ],
        [ right_label ]
    );

    this.background = Color.black;
    this.layout_(scope_grid);
    scope_display.start;
    this.onClose_{
        scope_display.stop;
        scope_display.free; scope_display = nil;
        if(scope_synth.isPlaying) {
            scope_synth.free;
            scope_synth = nil;
        };
        scope_buffer.free; scope_buffer = nil;
    };

    this.initQView(parent);
    if(parent != nil) { parent.layout = this.layout };

    ^this

```

```

    }
}

QZ_Scope : QWindow {
    var window_layout, <>z_scope_view;

    *new { arg name = "Z_Scope", bounds = Rect(675,125,300,100),
        resizable = true, border = true, server, scroll = false;

        ^super.new.initQZ_Scope(name, bounds, resizable, border, server, scroll)
    }

    initQZ_Scope { arg name, bounds, resizable, border, server, scroll;

        this.initQWindow(name, bounds, resizable, border, scroll);

        z_scope_view = QZ_ScopeView.new(bounds:bounds);
        // window_layout = QGridLayout.rows([z_scope_view]);
        window_layout = QStackLayout(z_scope_view);
        this.background = Color.magenta; // this color should be hidden
                                         // by the child view!

        this.alwaysOnTop = true;
        this.layout_(window_layout);
        this.onClose = { z_scope_view.freeScopeSynth };
        this.front;
        ^this
    }
}

```

```

QZ_ServerLevelsPanelView : QView {
    classvar <server;

    var panel_layout, <hour_stamp, <>time_label, <>server_label, <>level_label;
    var <>routine, <>tempo_clock;

    *initClass { StartUp.add { server = Server.default; } }

    *new{ arg parent = nil, bounds = Rect(600,425,580,60);
        ^super.new.initQZ_ServerLevelsPanelView(parent, bounds)
    }

    setTimeUnit { arg value;
        var unit = "";

        if(value < 10) { unit = "0" };

        // return as string...
        ^(unit ++ value)
    }

    setFloatUnit { arg float;
        var unit = "";

        if( (float % 1) == 0)          { unit = ".00" }
        // else...
        { if( ((float * 10) % 1) == 0) { unit = "0" } };

        // return as string...
        ^(float.asString ++ unit)
    }

    initQZ_ServerLevelsPanelView { arg parent, bounds;
        // check for valid server first and exit if invalid...
        ("Server Process ID = " ++ server.pid ++ "\n").inform;
        if(server.pid == nil)
        { (
            "\n"
            ++ "An instance of the QZ_ServerLevelsPanelView "
            ++ "class could not locate a valid server!\n"
            ++ "You may need to boot the server...\n").error;
            ^this
        };

        // variable and object initializations...
        this.bounds = bounds;

        /* create an independant TempoClock instance in order for the server
           reading routine to survive a TempoClock.clear method call on the
           default TempoClock...
        */
        tempo_clock = TempoClock();

        time_label = StaticText().string_("Test!").stringColor_(Color.cyan)
            .background_(Color.black).align_(\left).font_( Font(Font.default,
                size:14, bold:true) ).minWidth_(92);

        server_label = StaticText().string_(" Server : ").stringColor_(Color.gray)
            .background_(Color.black).align_(\center).font_( Font(Font.default,
                size:15, bold:true) ).minWidth_(60);

        level_label = Array.newClear(6);
    }
}

```

```

panel_layout = HLayout(time_label, server_label);
6.do{ arg i;
    var min_width = if(i < 2) { 69 } { 57 };

    level_label[i] = StaticText().minWidth_(min_width);
    level_label[i].background_(Color.black)
    .stringColor_(Color.green).align_(\right)
    .font_( Font(Font.default, size:14, bold:true) );
    panel_layout.add(level_label[i]);
};

/* code realating to the extension of the class' QView instance
   (i.e. the 'this' instance)...
*/
routine = Routine {
    { var hour   = this.setTimeUnit(Date.getDate.hour);
      var minute = this.setTimeUnit(Date.getDate.minute);
      var second = this.setTimeUnit(Date.getDate.second);
      var peak_string = this.setFloatUnit( (server.peakCPU.round(0.01))
      .asString.asFloat );
      var avg_string  = this.setFloatUnit( (server.avgCPU .round(0.01))
      .asString.asFloat );
      /* .asString.asFloat will ensure that any remaining
         smaller decimal places are completely truncated */

      hour_stamp = hour ++ ":" ++ minute ++ ":" ++ second;
      {
          time_label .string_(hour_stamp);
          level_label[0].string_(peak_string ++ "%");
          level_label[1].string_(avg_string ++ "%");
          level_label[2].string_(( server.numUGens      ).asString ++ "u");
          level_label[3].string_(( server.numSynths     ).asString ++ "s");
          level_label[4].string_(( server.numGroups    ).asString ++ "g");
          level_label[5].string_(( server.numSynthDefs  ).asString ++ "d");
      }.defer;
      0.25.wait;
    }.loop
};
tempo_clock.play(routine);

this.bounds_(bounds);
this.background_(Color.black);
this.layout_(panel_layout);
this.onClose_({
    routine.stop;
    routine.free; routine = nil;
    tempo_clock.clear;
    tempo_clock.free; tempo_clock = nil;
});

this.initQView(parent);
if(parent != nil) { parent.layout = this.layout };

^this
}
}

QZ_ServerLevelsPanel : QWindow {
    var window_layout, <>z_server_levels_panel_view;

```

```

*new { arg name = "Z_ServerLevelsPanel", bounds = Rect(600,425,580,60),
    resizable = false, border = true, server, scroll = false;

    ^super.new.initQZ_ServerLevelsPanel(name, bounds, resizable,
                                         border, server, scroll)
}

initQZ_ServerLevelsPanel { arg name, bounds, resizable,
                           border, server, scroll;

    this.initQWindow(name, bounds, resizable, border, scroll);

    z_server_levels_panel_view = QZ_ServerLevelsPanelView();

    this.background_(Color.blue);
    window_layout = HLayout(z_server_levels_panel_view);
    this.layout_(window_layout);
    this.alwaysOnTop_(true);
    this.front;
    ^this
}
}

```



