The Siren 7.5 Reference Manual

Music-Models-Representation

Class: **MusicMagnitude**

Environment: Siren
Superclass: Magnitude
Category: Music-Models-Representation
Instance variables: value
Class variables: TypeClasses TypeCodes

Instances of the subclasses of MusicMagnitude are used to represent scalar magnitudes in musical objects. The class MusicMagnitude is a place for the music magnitudes to share their state and basic accessing behavior.

MusicMagnitudes generally implement precise mixed-mode arithmetic and comparison methods using a per-species generality table and auto-coercion within a species.

The"type abstraction" classes (Duration, Pitch, Amplitude, etc.) represent "what"; are never instantiated; their class creation methods answer instances of their species, but they manage coersion within the classes of their species. The "representational abstraction" classes (NominalMagnitude, OrdinalMagnitude, NumericalMagnitude, RatioMagnitude, etc.) represent "how"; they share value model semantics and coercion techniques. Note that the concrete implementation classes therefore answer their species by property model rather than by implementation subclass (i.e., HertzPitch species --> Pitch rather than NumericalMagnitude).

There are many examples in the implementation classes and the Siren outline.

Instance variables:
value the magnitude's value

accessing

, anArgument
Answer en Event with the given magnitude as its first property.

value
Answer the instance variable 'value'.

value: theValue
Accept the argument, 'theValue', as the new instance variable 'value'.

printing

printOn: aStream
Print the receiver on the argument as a MM declaration.

printOn: aStream parens: useParens
Print the receiver on the argument as a MM declaration.

printString
Answer a String whose characters are a description of the receiver.

printStringParens: bool
Answer a String whose characters are a description of the receiver.

printValue

storeOn: aStream
Store the receiver on the argument as a MM declaration.

units
Answer the units string of the receiver.

private

species
Answer MusicMagnitude.

converting

adaptToNumber: rcvr andSend: selector
If I am involved in arithmetic with a number, convert me to a number.

coerceTo: type
Coerce the receiver to the given class and answer a new object.

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

msec
Answer a MSecondDuration.

usec
Answer a MSecondDuration.

testing

isImmediate
Answer whether the receiver has an immediate representation.

isMusicMagnitude
Answer true for the receiver and all subsclasses.

comparing

< aValue
Answer whether the receiver is less than the argument

= aValue
Answer whether the receiver and the argument are equal.

hash
Answer a SmallInteger unique to the receiver.

arithmetic

*** aValue**
Answer the product of the receiver and the argument

+ aValue
Answer the sum of the receiver and the argument, doing generality-based

class coercion within a species using auto-generated coercion messages.

- aValue
Answer the difference of the receiver and the argument

/ aValue
Answer the quotient of the receiver and the argument

MetaClass: MusicMagnitude class

instance creation

value: aValue
Answer and instance with the argument as its value.

objects from disk

initializeTypeCodes
Set up the class dictionary of subclass typecodes.

readDataFrom: refStream
Store an instance of some music magnitude subclass on the given reference stream compactly and quickly.

store: inst on: refStream
Store an instance of some music magnitude subclass on the given reference stream compactly and quickly.

class constants

color
Answer the default color to display instances with (hack for making pretty graphs).

propertyName
Answer the receiver class's default property name.

relativeMember
Answer the receiver class's species member used as a relative magnitude.

Class: **PField**

Environment: Siren
Superclass: Siren.MusicMagnitude
Category: Music-Models-Representation
Instance variables: name field

Instances of PField magnitudes are used for arbitrary parameters of musical events oriented towards formats with positional parameter declarations, e.g., Music-V-style notelist formats (see uses).

Instance variables:
field field in the score
name the parameter's name (optional)

Examples:
PField field: 5 value: 0.7 name: #position

PField field: 9 value: 'legato' name: #expression

accessing

field
Answer the instance variable 'field'.

field: theNumber
Accept the argument, 'theNumber', as the new instance nariable 'field'.

name
Answer the instance variable 'name'.

name: theName
Accept the argument, 'theName', as the new instance nariable 'name'.

MetaClass: PField class

instance creation

field: aNumber value: aValue
Answer an initialized instance.

field: aNumber value: aValue name: aName
Answer an initialized instance.

Class: **MusicModel**

Environment: Siren
Superclass: Siren.MusicMagnitude
Category: Music-Models-Representation
Class instance variables: generalityTable

MusicModel adds the class instance variable generalityTable that is used for "abstract" music magnitude models such as Pitch and Amplitude.

See the class methods for generality.

Class inst Vars:
generalityTable Integer)> The species generality table

MetaClass: MusicModel class

Instance variables: generalityTable

generality

generality
Answer the class inst var for the generality table.

generalize: aMag and: otherMag
Answer an array of the two arguments with the most general first.

examples

examples
Pitch generalize: (#c pitch) and: (21 key)

Class: **Chroma**

Environment: Siren
Superclass: Siren.MusicModel
Category: Music-Models-Representation

*Class Chroma is the abstract representational class for the pitch and mode-element species.
Instances of its subclass species are used to model pitches, gamut members, and frequencies.*

MetaClass: Chroma class

Class: **Ergon**

Environment: Siren
Superclass: Siren.MusicModel
Category: Music-Models-Representation

*Class Ergon is the abstract representational class for the amplitude/loudness/dynamic species.
Instances of its subclass species are used to model loudness values.*

MetaClass: Ergon class

Class: **Amplitude**

Environment: Siren
Superclass: Siren.Ergon
Category: Music-Models-Representation

*Instances of classes whose species is Amplitude are used for the loudness parameters of musical events.
Amplitudes come in several flavors, as in the classes RatioMagnitude, MIDIVelocity and SymbolicLoudness.
The class Amplitude is abstract, its class creation method answers an instance of a concrete Amplitude/Loudness class.*

Examples:
Amplitude value: 0.77 "create a ratio instance - range 0.0 to 1.0 (cmusic)"
Amplitude value: 77 "create an MIDI instance - range 0 to 127"
Amplitude value: #mp "create a symbolic instance - range #ppp to #fff"

See also the class example.

MetaClass: Amplitude class

instance creation

value: aValue
Answer a new instance of a member of my species.

class constants

color
Answer the default color to display instances with.

initialize
Set up the class inst var, a generality table.

initializeGenerality
Set up the class inst var, a generality table.

mostGeneral
Answer the most general-purpose duration--relative

propertyName
Answer the receiver class's default property name.

relativeMember
Answer the receiver class's species member used as a relative magnitude.

species
Answer Amplitude.

examples

example
Print a simple message to the transcript demonstrating the various types.

Class: **Pitch**

Environment: Siren
Superclass: Siren.Chroma
Category: Music-Models-Representation

Instances of classes whose species is Pitch are used for the pitch or frequency parameters of musical events.
Pitches come in several flavors, as in the classes HertzPitch, RatioPitch, MIDIPitch and SymbolicPitch.
The class Pitch is abstract, its class creation method answers an instance of a concrete Pitch class.

Examples:
Pitch value: 440.0 "create an instance with units of Hertz"
Pitch value: 77 "create an instance with units of MIDI key numbers"
Pitch value: #e4 "create a symbolic instance"
Pitch value: 'e4' "same as using a symbol"
Pitch value: 4/3 "create a ratio instance"

Note that new pitch representations such as music11-like pch (4.11 = 11th note in oct 4) or oct (4.1100 = oct4 + 1100 cts) notations can be added by overriding the float-to-Hz or float-to-midi conversions.

See also the class example.

MetaClass: Pitch class

instance creation

value: aValue
Answer a new instance of a member of my species.

class constants

color
Answer the default color to display instances with.

initialize
Set up the class inst var, a generality table.

initializeGenerality
Set up the class inst var, a generality table.

mostGeneral
Answer the most general-purpose duration--Hertz

propertyName
Answer the receiver class's default property name.

relativeMember
Answer the receiver class's species member used as a relative magnitude.

species
Answer Pitch.

examples

example
Print a simple message to the transcript demonstrating the various types.

exampleAdC
Pitch exampleAdC

Class: **Chronos**

Environment: Siren
Superclass: Siren.MusicModel
Category: Music-Models-Representation

*Class Chronos is the abstract representational class for the duration and meter species.
Instances of its subclass species are used to model times, durations and metronomes.*

MetaClass: Chronos class

Class: **Meter**

Environment: Siren

Superclass: Siren.Chronos
Category: Music-Models-Representation

Instances of the Meter species model the tempo or metronome used to map durations.
This class can be used as a concrete one (adding a few methods to fill it out), or like the other representational classes (making concrete classes of this species).
In the later case, the value instance variable could hold a number, process or block.

MetaClass: Meter class

Class: **Positus**

Environment: Siren
Superclass: Siren.MusicModel
Category: Music-Models-Representation

Class Positus is the abstract representational class for the position, space, and direction species.
Instances of its subclass species are used to model spatial and positional values.

MetaClass: Positus class

Class: **Spatialization**

Environment: Siren
Superclass: Siren.Positus
Category: Music-Models-Representation

Instances of the Spatialization species model the characteristics and configuration of room simulations in scores.
This class can be used as a concrete one (adding a few methods to fill it out), or like the other representational classes (making concrete classes of this species).
In the later case, the value instance variable would hold a record with the geometry or the room, the positions of default sources, and the listener's position and features.

MetaClass: Spatialization class

Class: **Position**

Environment: Siren
Superclass: Siren.Positus
Category: Music-Models-Representation

Instances of the Position species model the position of sound sources in room simulations or scores.
This class can be used as a concrete one (adding a few methods to fill it out), or like the other representational classes (making concrete classes of this species).
In the later case, the value instance variable could hold a number or point.

MetaClass: Position class

Class: **Directionality**

Environment: Siren
Superclass: Siren.Positus
Category: Music-Models-Representation

Instances of the Directionality species model the radiation characteristics of sound sources in room simulations.
This class can be used as a concrete one (adding a few methods to fill it out), or like the other representational classes (making concrete classes of this species).
In the later case, the value instance variable would hold a 1- or 2-dimensional position as a number or point.

MetaClass: Directionality class

Class: **ModeMember**

Environment: Siren
Superclass: Siren.Chroma
Category: Music-Models-Representation

Instances of the ModeMember species model pitches as elements of a mode (e.g., minor) or gamut (e.g., pentatonic on F).
This class can be used as a concrete one (adding a few methods to fill it out), or like the other representational classes (making concrete classes of this species).
In the later case, the value instance variable could hold a number or pitch, and the mode or gamut could be shared.

MetaClass: ModeMember class

Class: **Duration**

Environment: Siren
Superclass: Siren.Chronos
Category: Music-Models-Representation

Instances of classes whose species is Duration are used for the duration parameters of musical events. Durations come in several flavors, as in the classes RatioDuration, MSecondDuration and ConditionalDuration.
The class Duration is abstract, its class creation method answers an instance of a concrete Duration class.

Examples:
Duration value: 0.77 "create an instance with seconds as the unit"
Duration value: 770 "create an instance with milliseconds as the unit"
Duration value: 1/4 "create an instance with beats as the unit"
Duration value: [:x | x > 4] "create an instance for: 'until x > 4'"

See also the class example.

MetaClass: Duration class

instance creation

value: aValue
Answer a new instance of a member of my species.

class constants

color
Answer the default color to display instances with.

initialize
Set up the class inst var, a generality table.

initializeGeneralities
Set up the class inst var, a generality table.

mostGeneral
Answer the most general-purpose duration--seconds

propertyName
Answer the receiver class's default property name.

relativeMember
Answer the receiver class's species member used as a relative magnitude.

species
Answer Duration.

examples

example
Print a simple message to the transcript demonstrating the various types.

Music-Models-Implementation

Class: **ConditionalDuration**

Environment: Siren
Superclass: Siren.MusicMagnitude
Category: Music-Models-Implementation

Instances of ConditionalDuration are duration times where the value is a block.
The accessing protocol allows them to be spawned as co-processes in schedulers.
The valueAt: and waitUntil: methods allow flexible conditional scheduling and tests have shown that several dozen conditionals can be managed in real-time on modest hardware.

See the class examples.

accessing

valueAt: anArg
Answer the result of passing the argument to the receiver's block.

wait
Cycle the receiver until the argument fulfills the receiver's block.

waitUntil: anArg
Cycle the receiver until the argument fulfills the receiver's block.

private

species
Answer Duration.

converting

mostGeneral
It is an error to try this here--we implement what's ok for CDs

arithmetic

*** aValue**
Answer that it is an error to attempt arithmetic with this magnitude.

+ aDuration
Answer the sum of the receiver and the argument--the composition of two blocks

- aValue
Answer that it is an error to attempt arithmetic with this magnitude.

/ aValue
Answer that it is an error to attempt arithmetic with this magnitude.

MetaClass: ConditionalDuration class

instance creation

randomBetween: lo and: hi
Answer a new conditional duration whose value is between lo and hi (given in seconds)

examples

example
Print a simple message to the transcript demonstrating the various types.

exampleWithRands
Demonstrate the random duration

Class: **OrdinalMagnitude**

Environment: Siren
Superclass: Siren.MusicMagnitude
Category: Music-Models-Implementation

Instance variables: table

Class instance variables: Table

Instances of the OrdinalMagnitude classes are order-only magnitudes.
They use the instance or class instance tables for holding comparative relationships among instances (e.g., mag1 might know that it's > mag2).
The relation-setting (i.e., order assignment) messages are: ==, >>, <<, =<, and =>.
The query messages are: =?, > < <=, and >=.

Each subclass may decide whether instances or the class will hold the table of relationships.
The decision should be made on the basis of the expected number of magnitude instances and the sparseness of their relationships.

See the subclass' class examples.

Instance Variable:
table Symbol> instance rel. table

Class Instance Variable:
Table Symbol> class rel. table of all instances

accessing

hash
Answer a SmallInteger unique to the receiver. Essential. See Object documentation whatIsAPrimitive.

table
Answer the receiver's loop-up table--its or the class'.

value
Signal an error.

value: theValue
Signal an error.

initialize-release

release
Release the receiver's table.

printing

printOn: aStream
Print the receiver as an ordinal magnitude.

converting

mostGeneral
Answer that it is an error to attempt coercion with ordinal magnitudes.

ordering

< anotherOMag
Answer whether the receiver is less than the argument.

<< anotherOMag
Specify that the receiver is less than the argument.

<= anotherOMag
Answer whether the receiver is less than or equal to the argument.

=< anotherOMag
Specify that the receiver is less than or equal to the argument.

== anotherOMag
Specify that the receiver is equal to the argument.

=> anotherOMag
Specify that the receiver is greater than or equal to the argument.

=? anotherOMag
Answer whether the receiver is equal to the argument.

> anotherOMag
Answer whether the receiver is greater than the argument.

>= anotherOMag
Answer whether the receiver is greater than or equal to the argument.

>> anotherOMag
Specify that the receiver is greater than the argument.

arithmetic

*** aValue**
Answer that it is an error to attempt arithmetic with ordinal magnitudes.

+ aValue
Answer that it is an error to attempt arithmetic with ordinal magnitudes.

- aValue
Answer that it is an error to attempt arithmetic with ordinal magnitudes.

/ **aValue**
Answer that it is an error to attempt arithmetic with ordinal magnitudes.

= aValue
Answer whether the receiver and the argument are equivalent.

MetaClass: OrdinalMagnitude class

Instance variables: Table

table access

table
Answer the class' instance look-up table.

values
Answer the sorted values.

instance creation

new
Answer a new instance and, if it's in use, add it to the table.

value: aValue
Answer an instance

class initialization

flush
Release the shared class table and all instances.

useTable
Set up a shared class table for all instances.

Class: **Length**

Environment: Siren
Superclass: Siren.OrdinalMagnitude
Category: Music-Models-Implementation

*Instances of Length represent subjective length (~ duration * loudness) values.*
The instance variable tables are used for the name -> relation symbol map.

See the class examples.

private

species
Answer Duration.

MetaClass: Length class

examples

example
Demonstrate the use of an OrdinalMagnitude with a scale of length.

Class: **NominalMagnitude**

Environment: Siren
Superclass: Siren.MusicMagnitude
Category: Music-Models-Implementation
Class instance variables: NameMap

Instances of the NominalMagnitude classes are named (symbolic) properties where a symbol <--> value map is well-established for a given range and domain, e.g., 0.0 to 1.0 or 0 to 127.
Examples are pitch (#d4) or dynamic (#mp) names.

Class Instance Variable:
NameMap Number or Interval)> the class' look-up table

MetaClass: NominalMagnitude class

Instance variables: NameMap

class instance variables

nameMap
Answer the class instance variable NameMap.

Class: **SymbolicPitch**

Environment: Siren
Superclass: Siren.NominalMagnitude
Category: Music-Models-Implementation
Instance variables: fracPitch

Instances of SymbolicPitch represent well-tempered note names relative to a4=440Hz.
The range is c0 to g#9 and the values are symbols.
Note the confusion between the sharp sign (always placed after the note name) and Smalltalk's symbol key #.
One often writes ('c#3' asSymbol) to be safe.

Instance Variables:
fracPitch the remainder for microtonal tunings

The class instance variable NameMap is used for the name <--> MIDI key number mapping array.

printing

printOn0: aStream
Print the receiver as a symbolic pitch.

units
Answer the units string of the receiver.

arithmetic

+ aValue
Answer the sum of the receiver and the argument-handle adding Integers as a special case.

- aValue
Answer the sum of the receiver and the argument-handle adding Integers as a special case.

transposeBy: aValue
transpose a SymbolicPitch by aValue in fractional halfsteps

private

species
Answer Pitch.

converting

asFracMIDI
Assuming value is a symbolic note name, answer a key number.

asHertz
assuming value is a symbolic note name, return a frequency

asHz
assuming value is a symbolic note name, return a frequency

asMIDI
Assuming value is a symbolic note name, answer a key number.

asSymbol
Answer a SymbolicPitch.

mostGeneral
Answer the most numerically meaningful version of the receiver.

accessing

accidental
Answer the receiver's accidental, if any.

fracPitch
return microtonal offset as fractions of a halfstep

fracPitch: aValue
set microtonal offset as fractions of a halfstep

MetaClass: SymbolicPitch class

class initialization

initialize
Set up the class name mapping array.

instance creation

fromFracMIDI: aValue
Assuming value is a key number, answer a symbolic pitch name

fromMIDI: aValue
Assuming value is a key number, answer a symbolic pitch name

coercion

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

Class: **SymbolicLoudness**

Environment: Siren
Superclass: Siren.NominalMagnitude
Category: Music-Models-Implementation

Instances of SymbolicLoudness are dynamic names in the range #ppp to #fff.

The class instance variable NameMap is used for the name <--> MIDI velocity range (0 to 127) mapping dictionary.

printing

printOn0: aStream
Print the receiver as a symbolic amplitude.

units
Answer the units string of the receiver.

private

species
Answer Amplitude.

converting

asDB
Answer a dB loudness; ratio 1 = 0dB, ratio 0.5 = -6dB, etc.

asMIDI
Answer a MIDIVelocity.

asRatio
Answer a RatioLoudness.

asSymbol
Answer a SymbolicLoudness.

mostGeneral
Answer the most numerically meaningful version of the receiver.

MetaClass: SymbolicLoudness class

class initialization

initialize
Set up the class dynamic mapping dictionary

instance creation

fromMIDI: aValue
Assuming value is a key velocity, answer a symbolic loudness name

coercion

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

Class: **NumericalMagnitude**

Environment: Siren
Superclass: Siren.MusicMagnitude
Category: Music-Models-Implementation

Instances of the subclasses of NumericalMagnitude are music magnitudes with numerical values.

When the values are floating-point numbers, the range is often 0.0 to 1.0.
Integer ranges such as 0 to 127 can also be used in subclasses.

One may want to add range-checking for these cases.

Note that the class NumericalMagnitude is vacuous at present, and exists solely for representational modeling.

double dispatching

adaptInteger: val

adaptToInteger

MetaClass: NumericalMagnitude class

Class: **SecondDuration**

Environment: Siren
Superclass: Siren.NumericalMagnitude
Category: Music-Models-Implementation

Instances of SecondDuration are duration times in floating-point seconds.
This is among the most general duration time representations.

printing

printOn: aStream
Print the receiver on the argument as a MM declaration.

units
Answer the units string of the receiver.

private

mostGeneral
Answer the receiver in seconds.

species
Answer Duration.

converting

adaptToFloat
Answer a float of seconds.

asBeat
Answer a RatioDuration.

asMS
Answer a MSecondDuration.

asMsec

Answer a MSecondDuration.

asMseconds
Answer a MSecondDuration.

asRatio
Answer a RatioDuration.

asSec
Answer a float of seconds.

asSeconds
Answer a float of seconds.

asUsec
Answer a USecondDuration.

asUSeconds
Answer an int of micro seconds.

MetaClass: SecondDuration class

-- all --

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

Class: **USecondDuration**

Environment: Siren
Superclass: Siren.SecondDuration
Category: Music-Models-Implementation

Instances of USecondDuration are duration times in micro-seconds.
This is the default time representation (most general duration) and is usually used for keys in event lists.

printing

units
Answer the units string of the receiver.

converting

asMseconds
Answer a MSecondDuration.

asSeconds
Answer a float of seconds.

asUSeconds
Answer a float of micro seconds.

MetaClass: USecondDuration class

Class: **MSecondDuration**

Environment: Siren
Superclass: Siren.SecondDuration
Category: Music-Models-Implementation

Instances of MSecondDuration are duration times in milli-seconds.

printing

units
Answer the units string of the receiver.

converting

asMseconds
Answer a MSecondDuration.

asSeconds
Answer a float of seconds.

asUSeconds
Answer a float of micro seconds.

MetaClass: MSecondDuration class

coercion

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

Class: **HertzPitch**

Environment: Siren
Superclass: Siren.NumericalMagnitude
Category: Music-Models-Implementation

Instances of HertzPitch are frequency values in Hertz.
This is the most general Pitch representation.
The value is assumed to be a floating-point number.

printing

printOn0: aStream
Print the receiver as a pitch string in Hertz.

units
Answer the units string of the receiver.

private

species
Answer Pitch.

converting

asFracMIDI
Assuming value is a frequency, Answer a fractional key number

asFracSymbol
Assuming value is a frequency, answer a symbolic note name

asHertz
Answer a HertzPitch.

asHz
Answer a HertzPitch.

asMIDI
Assuming value is a frequency, Answer a key number

asSymbol
Assuming value is a frequency, answer a symbolic note name

MetaClass: HertzPitch class

coercion

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

Class: **RatioMagnitude**

Environment: Siren
Superclass: Siren.NumericalMagnitude
Category: Music-Models-Implementation
Instance variables: relative

Instances of the RatioMagnitude subclasses are fractional values relative to some context-defined reference value--their relative. The relative may be provided, or it may be assumed if uninitialized (e.g., for a default whole note's duration or default reference pitch).

Instance Variable:
relative the reference value

accessing

realValue
Answer the receiver's value mapped to the receiver.

value
Answer the receiver's value mapped to the receiver.

converting

asRatio
Answer self.

map
Apply the receivers reference.

relativeTo: aRelative
Set the receivers reference.

MetaClass: RatioMagnitude class

instance creation

value: aValue relative: aMMagnitude
Answer and instance with the argument as its value.

Class: **RatioDuration**

Environment: Siren
Superclass: Siren.RatioMagnitude
Category: Music-Models-Implementation

Instances of RatioDuration are 'beat' fractions.
They can be expanded into msec. relative to some given event (a whole note), or use the default tempo of 1 sec.

printing

printOn0: aStream
Print the receiver as a fractional duration.

units
Answer the units string of the receiver.

private

species
Answer Duration.

double dispatching

quotientFromInteger: numerator
Answer a MM whose value is the argument over the receiver's value.

converting

asMS
Answer a MSecondDuration.

asMsec
Answer a MSecondDuration.

asSec
Answer a SecondDuration.

mostGeneral
Answer the most numerically meaningful version of the receiver.

MetaClass: RatioDuration class

coercion

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

Class: **RatioLoudness**

Environment: Siren
Superclass: Siren.RatioMagnitude
Category: Music-Models-Implementation

Instances of RatioLoudness are fractional amplitude values normally relative to the range 0 to 1.

private

species
Answer Amplitude.

printing

printOn0: aStream
Print the receiver as a ratio.

converting

asDB
Answer a dB loudness; ratio 1 = 0dB, ratio 0.5 = -6dB, etc.

asMIDI
Answer a MIDI key velocity (0 to 127)

asSymbol
Answer a symbolic dynamic.

MetaClass: RatioLoudness class

coercion

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

Class: **IntervalMagnitude**

Environment: Siren
Superclass: Siren.NumericalMagnitude

Category: Music-Models-Implementation

Class instance variables: range

Instances of subclasses of the abstract class IntervalMagnitude are NumericalMagnitudes where a specific interval is defined within which values are possible. Examples are MIDI values in the range of 0 .. 127. the actual range is a class inst var.

accessing

value: theValue
Accept theValue, as the new instance variable 'value' -- perform range-checking

MetaClass: IntervalMagnitude class

Instance variables: range

class inst var access

range
Answer the class' range.

range: anInterval
Set the class' range.

Class: **MIDIVelocity**

Environment: Siren

Superclass: Siren.IntervalMagnitude

Category: Music-Models-Implementation

Instances of MIDIVelocity are key velocities (approximately proportional to loudness) in the range 0 to 127.

The SymbolicLoudness class maps symbolic dynamic names onto this range on an approximately logarithmic scale.

accessing

value: theValue
Truncate

printing

printOn0: aStream
Print the receiver as a MIDI velocity.

units
Answer the units string of the receiver.

private

species
Answer Amplitude.

converting

asDB
Answer a dB loudness; ratio 1 = 0dB, ratio 0.5 = -6dB, etc.

asMIDI
Answer a MIDIVelocity.

asRatio
Answer a RatioLoudness.

asSymbol
Answer a symbolic loudness.

mostGeneral
Answer the most numerically meaningful version of the receiver.

MetaClass: MIDIVelocity class

class initialization

initialize
Initialize the class instance variable.

coercion

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

Class: **MIDIPitch**

Environment: Siren
Superclass: Siren.IntervalMagnitude
Category: Music-Models-Implementation

Instances of MIDIPitch are well-tempered piano key numbers where 60 is middle-c (c3) according to the MIDI standard.

The translation key <--> Hz. is done with the logarithm or 27.5 Hz. (very low A = 440/16).

Note that AdC's additions make fractional MIDI pitches possible, whereby the first 2 digits to the right of the decimal point signify pitch cents. There are coercion methods such as asFracMIDI for handling microtonal MIDI pitches.

printing

units
Answer the units string of the receiver.

private

species
Answer Pitch.

converting

asFracMIDI
Answer a fractionalMIDIPitch (or integer if value is no fraction).

asFracSymbol
Assuming value is a key number, answer a symbolic pitch name

asHertz
Assuming value is a key number, answer a frequency

asHz
Assuming value is a key number, answer a frequency

asMIDI
Answer a MIDIPitch.

asSymbol
Assuming value is a key number, answer a symbolic pitch name

mostGeneral
Answer the most numerically meaningful version of the receiver.

MetaClass: MIDIPitch class

class initialization

initialize
Initialize the class instance variable.

coercion

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

Class: **RatioPitch**

Environment: Siren
Superclass: Siren.RatioMagnitude
Category: Music-Models-Implementation

Instances of RatioPitch are fractional pitch values normally relative to c=261.623Hz.
They can be transformed into other values (e.g., note names or key numbers), but often at a loss of accuracy if they are not well-tempered.

private

species
Answer Pitch.

printing

printOn0: aStream
Print the receiver as a ratio.

converting

asFracMIDI
return a RatioPitch as a fractional MIDINote.

asFracSymbol
Assuming value is a ratio, answer a symbolic note name

asHertz
Answer a HertzPitch.

asMIDI
return a RatioPitch as a MIDINote (rounded).

asSymbol
Assuming value is a ratio, answer a symbolic note name

mostGeneral
Answer the most numerically meaningful version of the receiver.

MetaClass: RatioPitch class

coercion

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

Class: **DBLoudness**

Environment: Siren
Superclass: Siren.RatioLoudness
Category: Music-Models-Implementation

Instances of DBLoudness represent deciBel loudness values.

printing

units
Answer the units string of the receiver.

double dispatch

differenceFromDynamic: aDyn
Subtract the argument from the receiver after turning both into db.

differenceFromVelocity: aDyn
Subtract the argument from the receiver after turning both into db.

productFromDynamic: aDyn
Multiply the argument by the receiver after turning both into db.

productFromVelocity: aDyn
Multiply the argument by the receiver after turning both into db.

quotientFromDynamic: aDyn

Divide the argument by the receiver after turning both into db.

quotientFromVelocity: aDyn
Divide the argument by the receiver after turning both into db.

sumFromDynamic: aDyn
Add the argument to the receiver after turning both into db.

private

species
Answer Amplitude.

converting

asMIDI
Answer a MIDI key velocity (0 to 127)

asRatio
Answer the receiver as a ratio between 0 and 1.

positiveDB
Answer the zero-based positive dB value of the receiver.

MetaClass: DBLoudness class

coercion

ddMsgName
Answer the selector to be used in coercing double-dispatching messages.

Class: **MeasureDuration**

Environment: Siren
Superclass: Siren.NumericalMagnitude
Category: Music-Models-Implementation
Instance variables: timeSignature

A MeasureDuration reprsents time as beats and measures.

It is partially implemented and unused.

Instance Variables:
timeSignature my time signature

printing

units
Answer the units string of the receiver.

private

species
Answer Duration.

MetaClass: MeasureDuration class

Class: **Sharpness**

Environment: Siren
Superclass: Siren.OrdinalMagnitude
Category: Music-Models-Implementation

*Instances of Sharpness represent subjective sharpness (~ pitch * loudness) values.*
The instance variable tables are used for the name -> relation symbol map.

See the class examples.

private

species
Answer Pitch.

MetaClass: Sharpness class

examples

example
Demonstrate the use of an OrdinalMagnitude with a scale of sharpness.

Music-Events

Class: **AbstractEvent**

Environment: Siren
Superclass: Object
Category: Music-Events
Instance variables: properties

AbstractEvent is the base class in the event/event-list hierarchy (although it can be instantiated). Instances of AbstractEvent are objects that can be used as dictionaries or property lists. Their properties can be accessed either with at: and at:put: or by using the new property names themselves as selectors (e.g., (anAEventInstance at: #color put: #green) or (anAEventInstance color: #green)).

Instance variables:
properties property list dictionary

The global dictionary called Events can be used for sharing event instances.

accessing

, anArgument
Add the argument as a property of the receiver.

date
Answer the receiver's 'date'.

date: obj
Set the receiver's 'date'.

inspect
Inspect the receiver--Use a special inspector for Event types

name
Answer the receiver's 'name'.

species
Answer AEvent--all subclasses look like me

version
Answer the receiver's 'version'.

version: obj
Set the receiver's 'version'.

printing

asExplorerString: showHide

display: showHide field: filter on: stream

displayField: filter on: stream

printOn: aStream
Format and print the receiver on the argument.

storeOn: aStream
Format and store the source the receiver on the argument.

templateFields
Answer the field names for the instances of the receiver class.

scheduling

scheduleOn: aChannel
Perform or interpret the receiver on the argument; override in subclasses.

properties

at: aProp
Answer a value from the property list dictionary (or an instVar).

at: aProp ifAbsent: otherCase
Answer a value from the property list dictionary or the value of the given block.

at: aProp put: aVal
Set a value in the receiver's property list dictionary (or instVar).

doesNotUnderstand: aMessage
Handle doesNotUnderstand: to try to access the property dictionary.
If this is unsuccessful, announce that the receiver does not understand the argument.

hasProperty: aSymbol
Answer whether or not the receiver's property list dictionary includes the symbol as a key.

properties
Answer the receiver's property list dictionary.

respondsTo: aSymbol
Answer whether the method dictionary of the receiver's class contains aSymbol as a message selector OR if the selector is unary and is a key in the receiver's property dictionary.

initialize-release

initialize
Set up the default state of the receiver--add props. dict.

release
Flush the receiver.

private

propCheck
Make sure the receiver has a property list dictionary.

testing

isEvent
Answer true.

isSound
Answer false.

comparing

= anObject
Answer whether the receiver and the argument represent the same values.

MetaClass: AbstractEvent class

examples

eventInspectExample
Demonstrate the creation of an AEvent.

example
Demonstrate the creation of an AEvent.

Class: **DurationEvent**

Environment: Siren
Superclass: Siren.AbstractEvent
Category: Music-Events
Instance variables: duration index startedAt realTime

Instances of DurationEvent are events that have special slots for their duration and voice properties.

Instance variables:
duration duration--a relative time

accessing

dur
Answer the receiver's duration.

dur: newValue
Set the receiver's duration.

duration
Answer the receiver's duration.

duration: newValue
Set the receiver's duration.

index: aNumber
Set the receiver's event index.

order
Answer the receiver's order.

order: anOrder
Set the receiver's order.

voice
Answer the receiver's voice or some reasonable default.

voice: aValue
Set the receiver's voice to the argument.

printing

printOn: aStream
Format and print the receiver on the argument.

storeOn: aStream
Format and store the source the receiver on the argument.

scheduling

nextTime: ignored
Answer whether to reschedule the receiver

play
Play the receiver by scheduling it.

playAt: aTime
Play the receiver on its voice then.

playOn: aVoice at: aTime
This is a no-op in the abstract class

reset
Reset the receiver's index.

scheduleAt: aTime
Play the receiver on its voice then.

comparing

= anObject
Answer whether the receiver and the argument represent the same values.

initialize-release

initialize
Set up the default state of the receiver--add props. dict.

MetaClass: DurationEvent class

examples

eventInspectExample
Demonstrate the creation of a DEvent.

example
Demonstrate the creation of a DEvent.

instance creation

dur: aD voice: aVoice
Answer a DurationEvent instance initialized with the arguments.

Class: **MusicEvent**

Environment: Siren
Superclass: Siren.DurationEvent
Category: Music-Events
Instance variables: pitch loudness voice

Instances of class MusicEvent are concrete musical note event objects used for eventLists and eventGenerators.

Instance variables:
pitch the pitch/frequency
loudness the loudness/amplitude
voice voice--a voice or key

accessing

ampl
Answer the receiver's loudness

ampl: aValue
Set the receiver's loudness to the argument.

amplitude
Answer the receiver's loudness

amplitude: aValue
Set the receiver's loudness to the argument.

loudness
Answer the receiver's loudness

loudness: aValue
Set the receiver's loudness to the argument.

pitch
Answer the receiver's pitch

pitch: aValue
Set the receiver's pitch to the argument.

voice
Answer the receiver's voice

voice: aValue
Set the receiver's voice to the argument.

processing

transposeBy: aStep
Add the given step to the receiver's pitch.

printing

printOn: aStream
Format and print the receiver on the argument.

printTerseOn: aStream
Format and print the receiver on the argument as tersely as possible.

printVerboseOn: aStream
Format and print the receiver on the argument.

readDataFrom: aDataStream size: size
Read a new event from the given stream using the compact format.

storeDataOn: aDataStream
Store myself on a DataStream. Answer self.

storeOn: aStream
Format and store the source the receiver on the argument.

comparing

= anObject
Answer whether the receiver and the argument represent the same values.

scheduling

playOn: aVoice at: aTime
Play the receiver on the voice then.

MetaClass: MusicEvent class

instance creation

ampl: anA voice: aVoice
Answer a MusicEvent instance initialized with the arguments.

dur: aD ampl: anA voice: aVoice
Answer a MusicEvent instance initialized with the arguments.

dur: aD pitch: aP
Answer a MusicEvent instance initialized with the arguments.

dur: aD pitch: aP ampl: anA
Answer a MusicEvent instance initialized with the arguments.

dur: aD pitch: aP ampl: anA voice: aVoice
Answer a MusicEvent instance initialized with the arguments.

dur: aD pitch: aP voice: aVoice
Answer a MusicEvent instance initialized with the arguments.

dur: aD voice: aVoice ampl: anA
Answer a MusicEvent instance initialized with the arguments.

duration: aD pitch: aP
Answer a MusicEvent instance initialized with the arguments.

duration: aD pitch: aP ampl: anA
Answer a MusicEvent instance initialized with the arguments.

pitch: aP
Answer a MusicEvent instance initialized with the argument.

pitch: aP ampl: anA voice: aVoice
Answer a MusicEvent instance initialized with the arguments.

class initialization

initialize
Initialize the global dictionary of Events (optional).

initializeEventDictionary
Initialize the global dictionary of Events (optional).

examples

eventInspectExample
Demonstrate the terse format of event description.

example
Demonstrate the terse format of event description.

Class: **ActionEvent**

Environment: Siren
Superclass: Siren.DurationEvent
Category: Music-Events
Instance variables: action
Indexed variables: objects

An instance of ActionEvent evaluates a Smalltalk block when scheduled.

Instance variables:
action Something to do

scheduling

play
Play the receiver by executing its action block.

playAt: aTime
Play the receiver by executing its action block.

accessing

action
Answer the receiver's 'action'.

action: anObject
Set the receiver's instance variable 'action' to be anObject.

MetaClass: ActionEvent class

examples

eventExample
Demonstrate the creation of an ActionEvent.

listExample
Demonstrate the creation of an ActionEvent by making a list of events that draw rectangles on the top window.

playExample
Demonstrate the performance of a list of ActionEvents.

instance creation

dur: aD action: actionBlock
Answer an ActionEvent instance initialized with the arguments.

dur: aD block: actionBlock
Answer an ActionEvent instance initialized with the arguments.

Class: **EventList**

Environment: Siren
Superclass: Siren.MusicEvent
Category: Music-Events
Instance variables: events

Instances of class EventList (subclass of DurationEvent), are used for holding onto multiple timed events. EventLists are events themselves and can have arbitrary properties as well as a collection of (relative start time -> event) associations.

Class EventList implements much collection-style protocol for event processing, as well as having special block application methods. All EventList algorithms are recurrsive in order to handle arbitrarily-deep

hierarchical EventLists (possible because EventList is a subclass of DurationEvent). The class also has numerous powerful and flexible instance creation methods.

Instance Variables:
events MEvent)>
the list's events, time/event associations sorted by relative start times
index the current position (used in scheduling)
startedAt the clock value when I sterted

Standard properties are #tempoScale for the tempo scaling factor (used by the CMN editors as the MS/whole note scale), and #code (#duration, #delta, or #startStop) to support different event coding schemes. The class also supports typed links to other event lists, e.g., "eList1 isTonicAnswerOf: eList2" or "eList1 hasPreviousVersion: eList2", which can be very useful properties in editors and browsers.

The global dictionary EventLists holds onto instances that can be referenced with their name symbols; these can have gensym-like names such as #temp.23. Creating a named EventList automatically adds it to the global dictionary. There are tools for browsing and maintaining this dictionary.

accessing

addModifier: aModifier
Add the given event modifier to the receiver's special modifier collection

code
Answer a symbol about the receiver's events' coding, usually #durations or #noteOnOff

code: aSymbol
Set the code about the receiver's events, usually #durations or #noteOnOff

deepCopy
Answer a careful deep copy.

defaultEventClass
Answer the default note event class.

name
Answer the receiver's name.

name: aName
Set the receiver's name.

next
Answer the next event in the list.

removeModifier: aModifier
Remove the given event modifier from the receiver's collection.

shallowCopy
Answer a shallow copy of the receiver's events.

size
Answer the number of events in the receiver.

tempo: someValue
Set the tempo to scale the events by.

voices
Answer the collection of voices used by thereceiver event list.

code conversion

continueProperties

Fill in missing properties with values that are continued from previous ones.

findNoteOff: anAssociation
Locate the noteOff event that corresponds to anAssociation; answer the sum of the delta times between (the duration)

setDurations
Take a delta-time encoded event list and determine the durations.

setDurations0
Take a delta-time encoded event list and determine the durations.

tracks
Used on MIDIScores.

collecting

collect: aBlock
Iterate over the receiver's events with the given block

collectAll: aBlock
Iterate over the receiver's events with the given block

do: aBlock
Iterate over the receiver's event associations with the given block

eventsDo: aBlock
Iterate over the receiver's events with the given block

excerptFrom: start to: end
Answer a hierarchical list delineated by the given times.

expanded
Expand all sub-event lists and answer one large flat list.

expandedFrom: start to: end
Expand all sub-event lists and answer one flat list with events between the given times.

group: selection
Group the argument's events as a sub-event list in the receiver's list.

includes: anAss
Answer whether the given association is in the receiver's event list.

keysAndValuesDo: aBlock

remove: anA
Remove the given event from the receiver's collection

select: aBlock
Iterate over the receiver's events with the given block

selectAll: aBlock
Iterate over the receiver's events with the given block

selectKey: aBlock
Iterate over the receiver's events with the given block

selectValue: aBlock
Iterate over the receiver's events with the given block

timesDo: aBlock

Iterate over the receiver's events' relative start times with the given block

printing

asExplorerString

asExplorerString: ignored

printOn: aStream
Print out the receiver's events on the argument.

readDataFrom: aDataStream size: size
Read a new event list from the given stream using the compact format.

storeDataOn: aDataStream
Store myself on a DataStream. Answer self.

storeOn: aStream
Store source code for the receiver's events on the argument.

storeOnFile: aName
Store out the receiver's events on the file named by the argument.

templateFields
Answer the field names for the instances of the receiver class.

processing

addValue: theValue toProp: theSelector
Add theValue to the aspect refered to by theSelector of all events.

applyBlock: theBlock toProp: theSelector
Apply the given block to the aspect refered to by theSelector of all events.

applyFunction: aFunction to: aSelector
Apply the given function to the range of the given selector

applyFunction: aFunction to: aSelector startingAt: sTime
Apply the given function to the range of the given selector starting at sTime.

chooseRange: theRange toProp: theSelector
Select a value from the given range for the aspect refered to by theSelector of all events.

edit
Open your favorite graphical editor on the receiver (left-shift-sensitive)

scale: theSelector by: theValue
Multiply the aspect refered to by theSelector of all events by theValue.

scaleStartTimesBy: theValue
Multiply the start times all events by theValue.

scaleValue: theValue toProp: theSelector
Multiply the aspect refered to by theSelector of all events by theValue.

setValue: theValue toProp: theSelector
Set the aspect refered to by theSelector of all events to theValue.

spreadValue: theRandomPercent toProp: theSelector
Spread the given property of all events by the given random range (%+-).

testing

hasItems
Answer whether or not the receiver has items or components (true).

isEmpty
Answer whether the receiver is an event list (true).

isEventList
Answer whether the receiver is an event list (true).

species
Answer EventList.

scheduling

loop
Play the event list by passing it off to the event scheduler.

nextTime: now
Answer the time of the next appointment

nextTimeFor: ass
Answer the time delay between the given event association and the one that follows it.

play
Play the event list by passing it off to the event scheduler.

playOn: aVoice
Play the event list on the voice by expanding its events

playOn: aVoice at: startTime
Play the event list on the voice by expanding its events

scheduleAt: aTime
Expand an eventList for the appointment scheduler.

stop
Play the event list by passing it off to the event scheduler.

initialize-release

empty
Throw away the receiver's events.

initializeAnonymous
Initialize an un-named instance with default state.

initializeNamed: aName
Initialize a named instance with default state.

release
Flush the receiver.

event accessing

, anEorA
Add a new event or association to the receiver.

add: anEorA

Add a new event or association to the receiver.

add: anEvent at: aTime
Add a new event to the receiver at the given relative start time.

addAll: anEventList
Add the given event list's events to the receiver.

eventKeys
Answer the keys (durations) of the receiver's eventDictionary.

eventNear: aPoint
Answer an event within a reasonable distance (100 msec.) of the given point's x.

eventNearestTime: aTime
Answer an event within a reasonable distance (32 msec.) of the given point.

eventNearTime: aTime
Answer an event within a reasonable distance (32 msec.) of the given point.

events
Answer the receiver's eventDictionary.

events: anEL
Set the receiver's eventDictionary.

eventValues
Answer the values (events) of the receiver's eventDictionary.

recomputeDuration
Recompute the total duration of the receiver.

private

map: anAssociation
Map the receiver's special properties and/or eventModifiers onto the given event.

map: anAssociation at: startTime
Map the receiver's special properties and/or eventModifiers onto the given event.

comparing

= anObject
Answer whether the receiver and the argument represent the same events.

MetaClass: EventList class

examples

creationExamples
Select the expressions below one-at-a-time and inspect the results.

randomExample
Create an event list with random data.

randomExample: length
Create an event list with random data.

randomExample: length from: data
Create an event list with random data.

randomSWSSExample
Create an event list with random data appropriate for software sound synthesis.

randomSWSSExample2
Create an event list with random data appropriate for software sound synthesis.

scaleExample2
Answer a scale where the event property types are mixed.

scaleExampleFrom: start to: stop in: dur
Answer an event list with a scale.

sentenceExample
Create an event list for a beautiful sentence.

wordExample
Create an event list for a long word.

class initialization

flush
Erase the shared EventList dictionary and try to clean up all events.

initialize
Initialize the shared EventList dictionary.

storeAll
Write out all event lists.

named constants

named: aName
Answer the named instance from the dictionary or a new EL.

named: aName ifAbsent: theBlock
Answer the named instance from the dictionary or the result of the block.

namedLists
Answer the dictionary of named event lists.

newNamed: aName
Create and answer a new named (and stored) instance.

nextName
Answer a gensym-type of event list name for which no EL exists in the dictionary.

instance creation

basicNew: ignored
Create and answer a new default-named instance of me

fromFile: theName
Create and answer a new default un-named (and therefore un-persistent) instance of me

named: myName fromPitches: pitchCollection
Answer an event list made with the given collection of pitches.

named: myName fromSelectors: selArray values: valueArrayArray
Answer an event list made with the given collection of values applied to the given array of selectors.

named: myName fromStartTimes: timeCollection

Answer an event list with the given collection of relative start times.

new
Create and answer a new default-named instance of me

new: ignored
Create and answer a new default-named instance of me

newAnonymous
Create and answer a new default un-named (and therefore un-persistent) instance of me

---

Music-EventGenerators

Class: **EventGenerator**

Environment: Siren
Superclass: Siren.EventList
Category: Music-EventGenerators

This class is the parent of most of the simple music-structure classes in this category.
An EventGenerator can create an EventList using its given parameters and a function (block) to use to create Event sequences. See the subclasses for examples.

playing

edit
edit me

eventList
Answer an eventList for the receiver.

play
Play the receiver's event list.

MetaClass: EventGenerator class

accessing instances

all
Find the and return instances of me or any subclasses of me.

allPlaying
Find all playing instances of me or my subclasses.

named: aName
Find the named instance of me or a subclass of me.

stopAll
Find all playing instances of me or my subclasses and terminate them.

examples

randomExample
Play random examples from the set of examples in my subclasses

Class: **Ostinato**

Environment: Siren
Superclass: Siren.EventGenerator
Category: Music-EventGenerators
Instance variables: list playing process

Ostinati can repeat their basic eventList without end, if necessary.

Instance variables:
list my riff
playing am i on?
process the process that's playing me

initializing

initializeAnonymous
set up a new Ostinato.

initializeNamed: aName
Set up a new Ostinato.

accessing

events

list

list: anEL

process: anBlockOrProc

playing

play: numberOfTimes
return an event list from the given number of times playing

playAt: firstTime
Play an ostinato until turned off.

playOn: aVoice at: firstTime
play an ostinato until turned off.

stop
stop a playing Ostinato.

printing

printOn: aStream

MetaClass: Ostinato class

instance creation

named: aSymbol onList: anEL
create an Ostinato on the given EventList.

named: aSymbol onList: anEL dur: theDur
create an Ostinato on the given EventList.

newNamed: aSymbol onList: anEL
create an Ostinato on the given EventList.

onList: anEL
create an Ostinato on the given EventList.

onList: anEL dur: theDur
create an Ostinato on the given EventList.

examples

ostinatoExamples
Create and edit some ostinati.

Class: **Cloud**

Environment: Siren
Superclass: Siren.EventGenerator
Category: Music-EventGenerators
Instance variables: density

Clouds are eventLists that are describe in terms of their contours or selection criteria.
This can be (e.g.,) POD-style specification of ranges for random selection, or selection from a given set of pitches.
See the class examples of my subclasses for description examples.

Instance variables:
density number of notes per second

accessing

density: theDensity
set the density instance variable

playing

eventList
Make the cloud's events; assume that the receiver's pitch, loudness and durations are actually intervals

eventList0
Make the cloud's events; assume that the receiver's pitch, loudness and durations are actually intervals

MetaClass: Cloud class

examples

dictionaryExample
Answer an initialized Cloud generated from the given property map dictionary.

example1
Create and edit a low 6 second stochastic cloud with 5 events per second

randomExample
Play a stochastic cloud with random properties.

randomExample2
Play a stochastic cloud with random properties.

instance creation

dur: aD pitch: aP ampl: anA
return an initialized Cloud

dur: aD pitch: aP ampl: anA voice: aVoice
return an initialized Cloud

dur: aD pitch: aP ampl: anA voice: aVoice density: aDe
return an initialized Cloud

dur: aD voice: aVoice density: aDe
return an initialized Cloud

fromDictionary: aDictionary
Answer an initialized Cloud generated from the given property map dictionary.

Class: **DynamicCloud**

Environment: Siren
Superclass: Siren.Cloud
Category: Music-EventGenerators

Dynamic clouds are clouds that have starting and ending conditions (ranges or selection sets) and interpolate between them over their duration.

playing

eventList
make the cloud's events--assume that the receiver's pitch, loudness and durations are actually ARRAYS of two intervals (for the start and end ranges)

MetaClass: DynamicCloud class

examples

crescendoExample
Create and play an 8-second cloud that goes from soft to loud on the same pitch.

example1
Edit a 6-second cloud that goes from low to high and soft to loud.

example4
Edit a 6 second cloud that focuses on its center

focusExample
Create a 6-second cloud that focuses on its center.

randomExample
Create a dynamic second cloud with random properties.

Class: **Peal**

Environment: Siren
Superclass: SequenceableCollection
Category: Music-EventGenerators
Instance variables: base directions position finished
Indexed variables: objects

Peals are repetitive note-generating sequences.
This implementation was written by Mark Lentczner in 1986.

Instance variables:
base the pitches i use
directions the direction of the current sub-sequence
position the current index
finished an i done?

playing

eventList

play

playOn0: out durations: dur meter: mtr at: start
Play the receiver

playOn: vox

playOn: out durations: dur meter: mtr at: start
Play the receiver

accessing

at: index
Answer the element in the base collection currently mapped into the index position.

at: index put: object

currentChange
Answer the base as mapped by the current change.

finished
Answer is the peal has completed it's last change

setBase: baseSet
Initialize the base set and everything else.

size
Redone here because SequencableCollection overrides it.

private

changeOrder: n bounds: range
Produce the next change in the peal by moving the element n within the range. If the element wants to move outside the range, alter it's direction and move the next element (recursively call this).

indexOfElement: n
Answer the index the anElement item (i.e. the n-th item, not the item = to n) within the receiver.

swap: index1 with: index2
We need to redefine this, since we redefined at: & at:put: to map trhrough the base collection

changing

change
Produce the next change in the peal

MetaClass: Peal class

examples

pealExample1
Play a simple bell peal

instance creation

new: size
Create a new peal on the given size with 1..size as the base set.

upon: baseSet
Create a new peal on the base set given

Class: **SelectionCloud**

Environment: Siren
Superclass: Siren.Cloud
Category: Music-EventGenerators

Selection clouds are created with a set of pitch and amplitude values and return eventLists selected from these values.

playing

eventList
Make the cloud's events

eventListWithDensityFunction
make the cloud's events

MetaClass: SelectionCloud class

examples

example1
Create a low 4 second cloud selecting pitch, amp and voice from value arrays.

exampleRand
Edit a selection cloud with random properties.

randomExample
Create a selection cloud with random properties.

Class: **DynamicSelectionCloud**

Environment: Siren
Superclass: Siren.SelectionCloud
Category: Music-EventGenerators

Dynamic selection clouds are described by their beginning and ending pitch and amplitude sets and interpolate between them.

playing

eventList
make the dynamic selection cloud's events

MetaClass: DynamicSelectionCloud class

examples

example1
Create a selection cloud that focuses onto a trill.

example2
Create a selection cloud that makes a transition from one triad to another

randomExample
Answer a dynamic selection cloud with random properties.

Class: **ExtDynamicSelectionCloud**

Environment: Siren
Superclass: Siren.DynamicSelectionCloud
Category: Music-EventGenerators
Instance variables: list

playing

eventList
make the dynamic selection cloud's events

accessing

list: aList
Set the receiver's list

MetaClass: ExtDynamicSelectionCloud class

examples

chordExample
Answer a dynamic selection cloud that plays chords from a scale.

Class: **Cluster**

Environment: Siren
Superclass: Siren.EventGenerator
Category: Music-EventGenerators

Cluster is an abstraction of a set of simultaneous (or same-pitch) events.
A cluster need only have a set of pitches, or a rhythm.

playing

eventList
make the cluster's events

MetaClass: Cluster class

instance creation

dur: aD list: anEL ampl: anA
return an initialized Cluster with the given list as pitches

dur: aD list: anEL ampl: anA voice: aV
return an initialized Cluster with the given list as pitches

dur: aD pitchSet: aColl ampl: anA voice: aV
return an initialized Cluster with the given list as pitches

examples

example1
Cluster example1

example2
Cluster example2

Class: **Roll**

Environment: Siren
Superclass: Siren.Cluster
Category: Music-EventGenerators
Instance variables: number delta noteDuration

Rolls repeat their single events.
It will eventually be possible to apply pitch, amplitude or duration envelopes to them as well.

Instance variables:
number how many notes to play
delta delta time between events
noteDuration duration of events

initialize

length: aLength rhythm: aDuration note: aNote
make a new roll of the specified length...

number: aNumber rhythm: aDuration note: aNote
make a new roll with the specified number of notes...

playing

eventList
return an eventList for me

accessing

duration
computer the duration

MetaClass: Roll class

instance creation

length: aNumber rhythm: aDuration note: aNote
return a new roll of the specified length...

number: aNumber rhythm: aDuration note: aNote
return a new roll with the specified number of events...

examples

rollExample
Create and edit/play a few rolls.

Class: **Trill**

Environment: Siren

Superclass: Siren.Roll
Category: Music-EventGenerators

A trill is like a roll, except that it can be given an eventList for repetition.

playing

eventList
return an eventList for me

MetaClass: Trill class

instance creation

length: aNumber rhythm: aDuration note: aNote
return a new roll of the specified length...

length: aNumber rhythm: aDuration notes: aNote
return a new trill of the specified length...

Class: **Chord**

Environment: Siren
Superclass: Siren.Cluster
Category: Music-EventGenerators
Instance variables: root inversion type arity

Instances of Chord are eventLists that can be created by giving them a root and inversion. They can return eventLists.

Instance variables:
root the root of the chord
inversion the inversion level (unused at present)
type #major, #minor, etc.
arity number of notes of the chord

accessing

arity: aNumber
set the number of notes of the receiver chord

duration: aDuration
set the durations of my notes

eventList
Answer the receiver's events

inversion: number
set the inversion of the receiver chord

root: tonic
set the root of the receiver chord

type: aSymbol
set the type symbol of the receiver chord

generating events

majorTetrad
return a three-note major chord on the given tonic in the given inversion

majorTriad
return a three-note major chord on the given tonic in the given inversion

minorTetrad
return a three-note minor chord on the given tonic in the given inversion

minorTriad
return a three-note minor chord on the given tonic in the given inversion

MetaClass: Chord class

instance creation

majorTetradOn: tonic inversion: inversion
return a three-note major chord on the given tonic in the given inversion

majorTriadOn: tonic inversion: inversion
return a three-note major chord on the given tonic in the given inversion

minorTetradOn: tonic inversion: inversion
return a three-note minor chord on the given tonic in the given inversion

minorTriadOn: tonic inversion: inversion
return a three-note minor chord on the given tonic in the given inversion

examples

example
Create and edit some chords.

Class: **Arpeggio**

Environment: Siren
Superclass: Siren.Chord
Category: Music-EventGenerators
Instance variables: delay

Arpeggii can be created on Chords or other event lists and can step through their events (assumed to be simultaneous at the start) with the given delay time.

Instance variables:
delay the delay betyween the onsets of my events

playing

edit

edit me

eventList
return my event list

play
don't expand me

private

setDelays
set the start times of my notes

accessing

delay: aValue
set the delay between the onset of my notes

MetaClass: Arpeggio class

instance creation

on: aChordOrList delay: aDelay
create an Arpeggio on the given chord or event list

Music-EventModifiers

Class: **EventModifier**

Environment: Siren
Superclass: Object
Category: Music-EventModifiers
Instance variables: selector function scale start stop index

EventModifier is the abstract superclass of the classes whose instances operate on event lists. There are operations that can be done eagerly (at definition time) or lazily (at run time)
Instance Variables:
selector What aspect of the event list do I modify
function What function do I aply to the aspect
scale Do I apply a scalar scale?
start When do I start in the event list?
stop When do I stop in the event list?
index Used internally to count through events

accessing

function
Answer the receiver's 'function'.

function: anObject
Set the receiver's instance variable 'function' to be anObject.

index
Answer the receiver's 'index'.

index: anObject
Set the receiver's instance variable 'index' to be anObject.

scale
Answer the receiver's 'scale'.

scale: anObject
Set the receiver's instance variable 'scale' to be anObject.

selector
Answer the receiver's 'selector'.

selector: anObject
Set the receiver's instance variable 'selector' to be anObject.

start
Answer the receiver's 'start'.

start: anObject
Set the receiver's instance variable 'start' to be anObject.

stop
Answer the receiver's 'stop'.

stop: anObject
Set the receiver's instance variable 'stop' to be anObject.

application

applyTo: evtList

valueIn: evtList at: time

MetaClass: EventModifier class

instance creation

new
Create a new modifier and initialize it

Class: **Swell**

Environment: Siren
Superclass: Siren.EventModifier
Category: Music-EventModifiers

A swell applies a fnuction to the amplitudes of events in an event list.

initialize-release

initialize

MetaClass: Swell class

as yet unclassified

example
Swell example

Class: **Rubato**

Environment: Siren
Superclass: Siren.EventModifier
Category: Music-EventModifiers

Rubato allows you to apply a function of time to the start times of events.

initialize-release

initialize

MetaClass: Rubato class

examples

example
Rubato example

Music-Functions

Class: **FunctionEvent**

Environment: Siren
Superclass: Siren.MusicEvent
Category: Music-Functions
Instance variables: function interval delta
Indexed variables: objects

accessing

delta

delta: aValue

function

function: aFcn

interval

interval: aValue

value
Answer the receiver's data value

events

playOn: aVoice at: aTime
Play the receiver on the voice then.

scheduling

nextTime: now
Answer the next time to reschedule me

play
Play the event list by passing it off to the event scheduler.

scheduleAt: aTime
Expand an eventList for the appointment scheduler.

MetaClass: FunctionEvent class

instance creation

new
Create and answer a new instance of me

Class: **Function**

Environment: Siren
Superclass: Siren.DurationEvent
Category: Music-Functions
Instance variables: data range domain
Indexed variables: objects

Instances of of Function and its subclasses represent abstractions of 1- or n-dimensional functions of 1 variable (e.g., time).
Class Function is concrete and represents functions that are described by a array of data points assumed to lie equally-spaced in the unit interval.
Functions are normally created from an array of values over the unit interval; x varies from 0.0 to 1.0 and y is free over that range.
One can address them within the unit interval with atX: or one can address them with integer indeces up to the data set's size with atIndex: (can be dangerous).

Examples are a ramp from 0 to 1 such as:
[(LinearFunction from: #((0 0) (1 2))) at: 0.33]
or a spline that traces a sine-like path centered around 1:
[(SplineFunction from: #((0 1) (0.33 2) (0.67 0) (1 1))) at: 0.35]

InstanceVariables:
data
the data values or breakpoints of the function
domain the x-range
range the y-range

ui

edit
Open a function view on the receiver.

updateSelector

processing

averagedTo: size
Answer an averaged version of the receiver of the given size.

freeData
Release the receiver's 'data'.

maxTo: size win: wsize
Answer a version of the receiver of the given size taking the maximum value of each window.

sampledTo: size
Answer a down-sampled version of the receiver of the given size.

smoothed
Answer a version of the receiver smoothed to about 32 points.

updateRange

accessing

add: aValue
Add the argument to the receiver's point collection.

at: anIndex
Answer the value at the given index (between 0 and 1 -or- 1 and data size).

at: anIndex put: aValue
Put the given value at the given index (between 0 and 1).

atX: anIndex
Answer the value at the given index (between 0 and 1).

data
Answer the receiver's 'data'.

data: anObject
Set the receiver's instance variable 'data' to be anObject.

dataClass
Answer the class of the elements in the receiver's data collection.

domain
Answer the receiver's 'domain'.

domain: anObject
Set the receiver's instance variable 'domain' to be anObject.

duration
Answer the domain of the receiver's collection of breakpoints.

nextXMoreThan: delta from: thisIndex
Answer the next X value after index whose Y value is more than delta from the value at thisIndex

nextXMoreThan: delta from: thisIndex step: step
Answer the next X value after thisIndex whose Y value is more than delta from the value at thisIndex

pointAt: index
Answer the given value in the receiver's breakpoint collection.

pointAt: index put: value
Assign the given values in the receiver's breakpoint collection.

points
Answer the receiver's 'data'.

range
Answer the receiver's 'range'.

range: anObject
Set the receiver's instance variable 'range' to be anObject.

realPointAt: index
Answer the given value in the receiver's breakpoint collection (this is not overridden in ExpSeg).

sampleAt: anIndex
Answer the value at the given index (between 0 and 1).

scale
Answer the receiver's 'scale'.

selection

size
Answer the size of the receiver's collection of breakpoints.

printing

printOn: aStream
Format and print the receiver on the argument.

storeOn: aStream
Format and print the receiver on the argument.

enumerating

detect: aBlock ifNone: exceptionBlock
Evaluate aBlock with each of the receiver's elements as the argument.
Answer the first element for which aBlock evaluates to true.

do: aBlock
Evaluate aBlock with each of the receiver's elements as the argument.

isEmpty

initialize-release

initialize: size
Initialize the receiver for the given size.

geometry

hasPointNear: anXValue
Answer whether or not the receiver has a function breakpoint near the given x value.

indexOfPointNearestX: anXValue
Answer the receiver`s point nearest the given x value.

MetaClass: Function class

standard functions

exponentialADSR1
Answer a exponential attack/decay/sustain/release envelope.

exponentialADSR2
Answer a exponential attack/decay/sustain/release envelope.

linearADSR1
Answer a linear attact/decat/sustain/release envelope.

linearADSR2
Answer a linear attact/decat/sustain/release envelope.

spline
Answer a generic spline curve.

sumOfSines
Answer a simple Fourier summation.

instance creation

default
Answer a default instance of the receiver class.

from: anArray
Answer a function with the given array of collections, points, or data values.

fromFile: fName
Load 1 or more functions from a text file.

new
Answer an instance of the receiver class.

ofSize: size
Answer an instance of the receiver class of the requested size.

randomOfSize: size from: low to: high
Answer a function with the given number of data points in the given range.

randomWalkSize: size from: low to: high
Answer a function with the given number of data points in the given range.

readFloatsFrom: filename
Answer a function with the given points.

examples

averagedFunctionFileExample
Function usage example; read a function from a binary file and view it.

fileExample
Function usage example; read a function from a binary file and view it.

functionFileExample
Function usage example; read a function from a binary file and view it.

functionPlayExample
Function usage example; make a roll-type eventList and apply a crescendo/decrescendo to it

functionViewExample
Function usage example; make a z-z function and view it.

maxedFunctionFileExample
Function usage example; read a function from a binary file and view it.

randomViewExample
Function usage example; make a random walk fcn and view it.

randomViewExample2
Function usage example; make a random walk fcn and view it.

class constants

defaultSize
Answer the default size for the instances' storage array.

Class: **FourierSummation**

Environment: Siren
Superclass: Siren.Function
Category: Music-Functions
Instance variables: myForm myArray lazy
Indexed variables: objects

Instances of FourierSummation are functions that interpret their points as 3-element arrays (harmonic, amplitude, phase), and sum sine waves into their data array based on the fourier summation of these components.

Instance variables:
myForm

my function plot--default size = 1024@180
myArray my value array--default length = 1024
lazy do I cache my values (eager) or cmopute them on the fly (lazy)?

Example:
| fcn |
fcn := FourierSummation from: #((1 1 0) (3 0.3 0) (5 0.2 0)

(7 0.15 0) (9 0.11 0) (11 0.09 0)).
Transcript show: (fcn at: 0.14) printString; cr.

initialize-release

initialize: size
Initialize the receiver for the given size.

computing

computeCurve
Compute the block by sine summation--set the values in the cached array.

computeValueAt: anIndex
Compute the answer by sine summation.

accessing

add: a3DPoint
Add the argument to the receiver's point collection.

at: theIndex
Answer the value from my array--assume an index in the range 0 to 1

MetaClass: FourierSummation class

instance creation

default
Answer a default instance of the receiver class.

from: anArrayOfZPoints
Answer a sum-of-sines function with the given points.

examples

fourierExample
Make a Sine summation that approaches a square wave

fourierViewExample
Make a Sine summation that approaches a square wave and open a view on it.

Class: **Spectrum**

Environment: Siren
Superclass: Siren.Function
Category: Music-Functions
Instance variables: sound window windowSize stepSize magReal
Indexed variables: objects

Instances of Spectrum represent 3-D data functions derived from sound analysis.

Instance Variables:

sound the receiver's sound
window the window type id
windowSize the window size
stepSize the step between windows
fft the receiver's FFT analyzer
magReal The spectrum type: mag, real, polar, or complex

accessing

D
Answer the receiver's decimation factor.

D: factor
Set the receiver's decimation factor.

N
Answer the receiver's 'windowSize'.

N: aNum
Set the receiver's instance variable 'windowSize' to be aNum.

sound
Answer the receiver's 'sound'.

sound: anObject
Set the receiver's instance variable 'sound' to be anObject.

stepSize
Answer the receiver's 'stepSize'.

stepSize: anObject
Set the receiver's instance variable 'stepSize' to be anObject.

windowSize
Answer the receiver's 'windowSize'.

windowSize: aNum
Set the receiver's instance variable 'windowSize' to be aNum.

computing

compute
Do the FFT and store the results into the receiver's data.

compute0
Do the FFT and store the results into the receiver's data.

setData
Create the receiver's data array for the right number of frames.

updateRange
Iterate over the receier's data getting the RMS data range

window: which
Answer the real data array for the given window of the receiver's sound.

modes

complex
Set the receiver's instance variable 'magReal' to be #complex.

real
Set the receiver's instance variable 'magReal' to be #real.

setWindow: aSymbol
Set the receiver's 'window'.

window
Answer the receiver's 'window'.

time warp

interpolateFrame: ind
Answer a new frame interpolated for the given (floating-point) index.

timewarpBy: fact
Interpolate/decimate the receiver by the given (float or function) time factor.

printing

display
Display the receiver

printOn: aStream
Format and print the receiver on the argument.

initialize-release

initialize
Set up the receiver.

frames

at: anIndex
Answer the value at the given index (between 1 and self size).

at: anIndex put: aValue
Answer the value at the given index (between 0 and 1).

frame: anIndex
Answer the frame at the given index (between 1 and self size).

frame: anIndex do: aBlock
Iterate over a single frame with the given block.

frame: frm imagAt: frq
Answer the given imag value in the given frame.

frame: frm imagAt: frq put: val
Set the given imag value in the given frame.

frame: anIndex put: aValue
Set the value at the given index (between 0 and 1).

frame: frm realAt: frq
Answer the requested real value from the given frame.

frame: frm realAt: frq put: val
Set the given real value in the given frame.

MetaClass: Spectrum class

examples

fileExample
Read a sound from disk and take its fft.

sweepExample
Create a swept sine wave and take its fft.

Class: **LinearFunction**

Environment: Siren
Superclass: Siren.Function
Category: Music-Functions
Indexed variables: objects

Instances of LinearFunction are line-segment functions of one free variable.

Example:
LinearFunction from: #((0 0) (0.1 1) (0.2 0.6) (0.9 0.4) (1 0))

processing

normalize
Normalize the receiver to the range and domain of 0-1, inclusive.

normalize1
Normalize the receiver to the range and domain of 0-1, inclusive.

scaleBy: scalePt
Scale the receiver's points by the given scale.

updateRange

accessing

add: anItem
Add the argument to the receiver's point collection.

at: anIndex
Answer the value at the given index in my range--do linear interpolation.

size
Answer the size of the receiver's collection of breakpoints.

MetaClass: LinearFunction class

instance creation

a: att d: dec s: sus r: rel
Answer an ADSR envelope.

default
Answer a default instance of the receiver class - an ADSR shape.

fromLorisData: data size: siz duration: dur
Create a linear envelope from the given Loris linear envelope.

examples

exampleEnvelope
Answer a LinSeg of a typical envelope function.

linearExample
Make a LinSeg and answer a value.

linearViewExample
Make an ADSR-shaped LinSeg and open a view on its form.

Class: **ExponentialFunction**

Environment: Siren
Superclass: Siren.LinearFunction
Category: Music-Functions
Indexed variables: objects

Instances of ExponentialFunction are exponential-segment functions of one free variable. They use ZPoints for x/y/exponent. The exponent determines the speed of the exponential/logarithmic transition between breakpoint values. A value of 0 leads to linear interpolation.

Example:
ExponentialFunction from: #((0 0 -5) (0.2 1 -3) (0.8 0.5 -2) (1 0))

accessing

at: anIndex
Answer the value at the given index in my range--do exponential interpolation such that if f(x, i) is the i-th function value in the transition from breakpoint v[J] to v[J+1], then:
f(x) = v[J] + (v[J+1] - v[J])(1 - exp(i*x[J]/(N-1)))/(1 - exp(x[J]))*
for 0 <= i < N, where N is the number of function points between t[J] and the next horizontal value, and x is the exponential weight whereby x = 0 will yield a straight line, x < 0 will yield an exponential transition, and x > 0 will yield a logarithmic transition.

pointAt: index
Answer the given value in the receiver's breakpoint collection as a 2-D point.

printing

printOn: aStream
Format and print the receiver on the argument.

MetaClass: ExponentialFunction class

instance creation

a: att d: dec s: sus r: rel
Answer an ADSR envelope.

default
Answer a default instance of the receiver class.

from: anArrayOfPoints
Answer a function with the given points.

examples

expADSRViewExample
Make an exp seg and open a view on its form.

expASRViewExample
Make an exp seg and open a view on its form.

expsegViewExample
Make an exp seg and open a view on its form.

Class: **SplineFunction**

Environment: Siren
Superclass: Siren.LinearFunction
Category: Music-Functions
Instance variables: linSeg
Indexed variables: objects

Instances of SplineSeg are cubic splines betwen their points.

Instance Variable:
linSeg my linear twin

computing

computeCurve
Compute the receiver by cubic interpolation; use the Geometric spline.

accessing

at: anIndex
Answer the value at the given index in my range--take it out of my computed form.

MetaClass: SplineFunction class

examples

splineExample
Make a SplineSeg.

splineViewExample
Make a SplineSeg and open a view on its form.

instance creation

default

Answer a default instance of the receiver class.

Music-PitchClasses

Class: **PitchClass**

Environment: Siren
Superclass: Object
Category: Music-PitchClasses
Class variables: A AllNatural AllNotes B C D E English F G

Instances of (subclasses of) this class represent pitch-classes.
A pitch class is an octave-independent note.
There are 35 (sub) instances of this class.
Octave-dependent notes are represented by instances of class OctaveDependentNote.
Ideally PitchClass should be a metaclass, so that its instances be classes, and octave-dependent notes could then be instances of pitchClasses !!
Unfortunately this is not possible straightforwadly in Smaltalk, so we use aggregation instead to represent octave-dependent notes

private intervals

descendingNumberOfSemiTonesBetween: aNote
Important method in the theory. It is a 3-stage computation of the interval between 2 notes

descendingSemiTonesToNatural

intervalBetween: aNote
returns the interval between the two notes

intervalTypeBetween: aNote
returns the interval between the two notes

nthFollowing: i
returns the i th natural note following self

nthPreceding: i
returns the i th natural note preceding self

numberOfSemiTonesBetween: aNote
Important method in the theory. It is a 3-stage computation of the interval between 2 notes

semiToneCount

smallestIntervalBetween: aNote
returns the descending interval between the two notes

public intervals

alterate: note toReach: i

alterateBelow: note toReach: i

augmentedEleventh

augmentedFifth

augmentedFourth

augmentedNinth

augmentedSecond

augmentedUnison

diminishedFifth

diminishedNinth

diminishedSecond

diminishedSeventh

diminishedThirteenth

eleventh

fifth

flatFifth

flatNinth

flatThirteenth

fourth

majorNinth

majorSecond

majorSeventh

majorSixth

majorThird

majorThirteenth

minorNinth

minorSecond

minorSeventh

minorSixth

minorThird

minorThirteenth

ninth

octave

perfectEleventh

perfectFifth

perfectFourth

second

seventh

sixth

thirteenth

unison

testing

isDoubleFlat

isDoubleSharp

isFlat

isNatural

isSharp

pitchEqual: aNote

chord creation

chordFromTokens: st

majorTriad

minorTriad

mutations

@ o

chordFromString: st

inTessitura: qualTessitura
N C pitchInTessitura: QTessitura high

octave: o

scales

chromaticScale

harmonicMinorScale

majorScale

melodicMinorScale

minorScale

pseudoMinorScale

making octave-dep notes

downAndUpOctaves: odNote

returns two octave dependent notes of self who are repectively lower and higher than the given odNote

pitchBetween: n1 and: n2
return, if it exists, a pitch (octave dependent note) between the two given notes

the: nb octavesBeginningFrom: initialOctave
N D the: 3 octavesBeginningFrom: 0

access

doubleFlat

doubleSharp

natural

pitchClass

saving

representBinaryOn: s
to ensure uniqueness, pitch classes save themselves as messages sent to the appropriate class, so that no duplicate are created

printing

printOn: s

storeOn: s
N C storeString

transpose

transposeOf: anInterval
three cases: integer (+/-), method name (= ascending interval), or interval object

constraining

intervalTypeModuloOctaveBetween: n

Paleo

copy

nameInScale: aScale

MetaClass: PitchClass class

initialization

englishOrFrench

initialize
There are 35 pitch classes. That's too much for Squeak, so the method was split into two methods

initializeAllNaturalNotes

initializeClass
There are 35 pitch classes. That's too much for Squeak, so the method was split into two methods

initializeDoubleFlat

initializeFrenchNames

initializeGlobals

global access

A

allNotesButDoubles

allPlausibleRootNotes

allPlausibleRootsForMajorScales
self allPlausibleRootsForMajorScales

allPlausibleRootsForMinorScales
self allPlausibleRootsForMinorScales

B

C

D

do

E

F

fa

flatNotes

fromSemiTones: n
arbitrary method, used for transposing pitch classes (a strange notion...)

G

la

mi

naturalNotes

noteNamed: c
takes the case of flat into account. Sharps are naturally parsed out from the note name by the smalltalk parser.
Since the algorithm proceeds from the left, it accepts any number of sharps and flats (using the common algebra) :
N noteNamed: 'C#b#b#b#' -> C#

re

sharpNotes

si

sol

vocal ranges

altoRange
retourne la collection des notes de l'alto

altoRangeInScale: aScale
PitchClass sopranoRangeInScale: (N do sharp minorScale)

baseRange
retourne la collection des notes de la basse

baseRangeInScale: aScale
PitchClass sopranoRangeInScale: (N do sharp minorScale)

sopranoRange
retourne la collection des notes du soprano

sopranoRangeInScale: aScale
PitchClass sopranoRangeInScale: (N do sharp minorScale)

tenorRange
retourne la collection des notes du tenor

tenorRangeInScale: aScale
PitchClass sopranoRangeInScale: (N do sharp minorScale)

ordering

flatOrdering

nFirstFlats: n
PitchClass nFirstFlats: 3

nFirstSharps: n
PitchClass nFirstSharps: 3

sharpOrdering

examples

chordExamples
N C sharp chordFromString: '' -> [C#]

(N C sharp chordFromString: 'min 7 dim5') notes -> OrderedCollection (C# E G B)

closestEnharmonic

N D sharp closestEnharmonic Eb
N E flat closestEnharmonic D#

N B sharp closestEnharmonic C
N C flat closestEnharmonic D#

N D sharp sharp closestEnharmonic E
N E flat flat closestEnharmonic D

(N D sharp @ 2) closestEnharmonic Eb2
(N E flat @ 2) closestEnharmonic D#2

(N B sharp @ 2) closestEnharmonic C3
(N C flat @ 3) closestEnharmonic B2

majorScaleExample
PitchClass B majorScale notes- > #(B C# D# E F# G# A#)

melodicMinorScaleExample

minorScaleExample

pitchInTessituraExamples

N C pitchInTessitura: QTessitura high -> C5
N A pitchInTessitura: QTessitura high -> A4
N G pitchInTessitura: QTessitura high -> G4
N F pitchInTessitura: QTessitura high -> F5

N A pitchInTessitura: QTessitura low -> A2

sharpflatAlgebraExample

sharp and flat's algebra
N C sharp ->Do#
N C sharp sharp ->Do##
N C sharp sharp sharp ->error

N C flat sharp -> Do
N re sharp sharp natural -> re

Intervals computation :
N C diminishedFifth -> Solb
N C augmentedFourth -> Fa#
N C diminishedThirteenth -> Lab
N C flat minorSeventh -> Sibb
N C majorThird majorThird -> Sol#

Notes equivalence : pitchEgal methode
N C sharp pitchEqual: N re flat -> true
N C augmentedFourth pitchEqual: N C diminishedFifth -> true
N C diminishedFifth pitchEqual: N F minorSecond -> true

Class: **ChordNameEditor**

Environment: Siren
Superclass: ApplicationModel
Category: Music-PitchClasses
Instance variables: pitchClass pitchClassIndex structure seventh eleventh root ninth fifth third thirteenth noteNames

change

pitchClassIndexChanged

structureChanged

initialize

initialize

aspects

chord

eleventh
This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method.

fifth
This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method.

ninth
This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method.

noteNames
This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method.

pitchClass
This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method.

pitchClassIndex
This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method.

root
This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method.

seventh
This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method.

structure
This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method.

third
This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method.

thirteenth
This method was generated by UIDefiner. The initialization provided below may have been preempted by an initialize method.

actions

doHalfDim7

doMaj

doMaj7

doMin7

MetaClass: ChordNameEditor class

interface specs

windowSpec
UIPainter new openOnClass: self andSelector: #windowSpec

resources

eleventhMenu
UIMenuEditor new openOnClass: self andSelector: #eleventhMenu

fifthMenu
UIMenuEditor new openOnClass: self andSelector: #fifthMenu

ninthMenu
UIMenuEditor new openOnClass: self andSelector: #ninthMenu

rootMenu
UIMenuEditor new openOnClass: self andSelector: #rootMenu

seventhMenu
UIMenuEditor new openOnClass: self andSelector: #seventhMenu

thirdMenu
UIMenuEditor new openOnClass: self andSelector: #thirdMenu

thirteenthMenu
UIMenuEditor new openOnClass: self andSelector: #thirteenthMenu

---

Class: **PitchClassAltered**

Environment: Siren
Superclass: Siren.PitchClass
Category: Music-PitchClasses
Instance variables: natural

accessing

following

name
yields a symbol

natural

natural: value

nom
yields a symbol

preceding

testing

isNatural

MetaClass: PitchClassAltered class

Class: **PitchClassFlat**

Environment: Siren

Superclass: Siren.PitchClassAltered
Category: Music-PitchClasses
Instance variables: flat

intervals

alterate: note toReach: i

alterateBelow: note toReach: i

closestEnharmonic

semiTonesToNatural

saving

representBinaryOn: s

printing

printOn: s

storeOn: s
N C flat storeString

testing

isFlat

accessing

flat

flat: value

semiToneCount

sharp

MetaClass: PitchClassFlat class

Class: **Signature**

Environment: Siren
Superclass: Object
Category: Music-PitchClasses
Instance variables: sharps flats

Notion of signature is related to the notion of "legal" scales, i.e.
scales having only sharps OR flats in their signature.
So method tonality here works only for legal scales.
It should possible to compute the tonality for arbitrary scales though,
but I am not sure how

printing

printOn: s

storeOn: s

testing

flatsInRightOrder

isEmpty

isLegal

sharpsInRightOrder

tonality

tonality

access

alterate: aNaturalNote

flats

initialize

nbFlats: n

nbSharps: n

reOrderSharpsAndFlats
attempts to reorder flats and sharps according to the natural order as found in

PitchClass

sharps

sharps: s flats: f
attempts to reorder flats and sharps according to the natural order as found in PitchClass

MetaClass: Signature class

examples

example

example2
Signature new tonality -> C MajorScale
(Signature new nbSharps: 4) tonality -> E MajorScale
(Signature new nbFlats: 3) tonality -> Eb MajorScale

example3
Signature fromTonality: N D majorScale
Signature fromTonality: N E flat majorScale

example4
Signatures may be created for illegal scales. However, method tonality yields an error for these scales:

Signature fromTonality: N D minorScale -> 1 sharps; 1 flats
Signature sharps: (Array with: N F sharp) flats: (Array with: (N E flat)) -> 1 sharps; 1 flats
(Signature fromTonality: N D minorScale) tonality -> error

creation

choseSignature
Signature choseSignature

fromTonality: aScale
Signature fromTonality: N E flat majorScale

new

sharps: s flats: f

Class: **PitchClassDoubleSharp**

Environment: Siren
Superclass: Siren.PitchClassAltered
Category: Music-PitchClasses

saving

representBinaryOn: s

intervals

alterate: note toReach: i

alterateBelow: note toReach: i

closestEnharmonic

semiTonesToNatural

access

flat
la methode diese n'est pas definie et provoque donc une erreur

semiToneCount

sharp
Sorry, I have to do that (Cf. Bluesette) otherwise I can't compute possibleTonalities properly ...

printing

printOn: s

storeOn: s
N C sharp sharp storeString

testing

isDoubleSharp

MetaClass: PitchClassDoubleSharp class

Class: **MusicalInterval**

Environment: Siren
Superclass: Object
Category: Music-PitchClasses
Instance variables: type semiTones

MusicalInterval commentStamp: '' prior: 0!
Intervalle represente un intervalle entre deux notes.
type represente l'ecart entre les deux notes extremites. Cet ecart ne tient compte que des notes naturelles. Par exemple, les tierces (majeures ou mineures) ont comme type 3,

les quarte 4 etc.
demisTons est le nombre de demis tons entre les deux notes. Par exemple une tierce majeure a comme type 3 et demisTons = 4.

computing notes

topIfBottomIs: aNote
yields the note making the interval self with aNote

printing

isPrintable

printOn: s

printPrintableOn: s
Watch out : all Intervals cant print themselves.
You can convert a musicalInterval into a ChromaticInterval.
E.g. : Interval between Cbb and C ## (super super augmented unison!!).
There are here 40 printable intervals. Cf. method MusicalInterval allPrintableIntervals (put in protocol constants)

storeOn: s

automatic access

orientedSemiTones

semiTones

semiTones: aValue

type

type: aValue

arithmetics

+ anInterval
(MusicalInterval majorSecond) + (MusicalInterval perfectFifth)

testing

isA: anInterval

isAscending

isDescending

comparing

< anInterval

<= anInterval

= anInterval

> anInterval

>= anInterval

hasSameDirectionAs: anInterval

mutations

ascending

asChromaticInterval

descending

MetaClass: MusicalInterval class

examples

allPrintableIntervals
MusicalInterval allPrintableIntervals

example

MusicalInterval majorThird inverse

MusicalInterval perfectFourth topIfBottomIs: N C sharp -> Fa#
MusicalInterval perfectFourth bottomIfTopIs: N F sharp -> Do#

N do intervalBetween: N re -> Major second
N do flat intervalBetween: N sol -> Augmented fifth
N do flat intervalBetween: N do sharp -> superAugmented unisson
N do flat intervalBetween: N do sharp sharp -> non printable interval
(N do flat intervalBetween: N do sharp sharp) asChromaticInterval -> Chromatic 3
(N do intervalBetween: N sol) = (N re intervalBetween: N la) -> true
(N do intervalBetween: N fa sharp) = (N do intervalBetween: N sol flat) -> false
MusicalInterval allIntervalsType: 3 -> #(Major third Minor third)
MusicalInterval allIntervalsType: 2 ->
#(Augmented second Diminished second Major second Minor second)
MusicalInterval allIntervalsType: 1 -> #(Augmented unisson Diminished unisson superAugmented unisson superDiminished unisson unisson)

exampleTranspositions
three ways of of transposing pitch classes and od notes

creation

allIntervalsType: t
MusicalInterval allIntervalsType: 2

type: aType semiTones: d

all-printable

augmentedEleventh

augmentedFifth
8 semitones

augmentedFourth
6 semitones

augmentedNinth
15 semitones

augmentedOctave
3 semitones

augmentedSecond
3 semitones

augmentedTenth
17 semitones

augmentedTwelvth
20 semitones

augmentedUnison

diminishedFifth
6 semitones

diminishedFourth
5 semitones

diminishedNinth
1 octave + 1 semitones

diminishedOctave
11 semitones

diminishedSecond
0 semitones

diminishedSeventh
9 semitones

diminishedTenth
15 semitones

diminishedThirteenth

diminishedTwelvth
18 semitones

diminishedUnison
MusicalInterval diminishedUnison topIfBottomIs: N C

majorNinth
14 semitones

majorSecond
2 semitones

majorSeventh
11 semitones

majorSixth
9 semitones

majorTenth
14 semitones

majorThird
4 semitones

majorThirteenth

majorTwelvth
19 semitones

minorNinth
1 octave + 1 semitones

minorSecond
1 semitone

minorSeventh
10 semitones

minorSixth
8 semitones

minorThird
3 semitones

minorThirteenth

octave

perfectEleventh
5 semitones

perfectFifth
7 semitones

perfectFourth
5 semitones

superAugmentedUnison
MusicalInterval superAugmentedUnison topIfBottomIs: N C

superDiminishedUnison
MusicalInterval superDiminishedUnison topIfBottomIs: N C

unison

constants nick-names

aug11

aug4

aug5

aug9
15 semitones

dim13

dim5
6 semitones

dim7
9 semitones

dim9

eleventh

fifth

flatFifth

flatNinth
1 octave + 1 semitones

flatThirteenth

fourth

ninth

second
2 semitones

seventh

sixth

thirteenth

Class: **DiatonicInterval**

Environment: Siren
Superclass: Siren.MusicalInterval
Category: Music-PitchClasses
Instance variables: ascending

private

ascending: t

computing notes

bottomIfTopIs: aNote
yields the note making the interval self with aNote

topIfBottomIs: aNote
yields the note making the interval self with aNote

printing

printOn: s

initialize

initialize

MetaClass: DiatonicInterval class

creation

ascending

descending

new

Class: **ChromaticInterval**

Environment: Siren
Superclass: Siren.MusicalInterval

Category: Music-PitchClasses

ChromaticInterval commentStamp: '' prior: 0!
used to produce intervals with no names, measured only in terms of a number of semie tones (for non-tonal music)

printing

printOn: s

MetaClass: ChromaticInterval class

examples

examples
(N C octave: 3) transposeBySemiTones: 4 -> E3

creation

semiTones: x
anInterval without type, only chromatic

Class: **PitchClassDoubleFlat**

Environment: Siren
Superclass: Siren.PitchClassAltered
Category: Music-PitchClasses

saving

representBinaryOn: s

intervals

alterate: note toReach: i

alterateBelow: note toReach: i

closestEnharmonic

semiTonesToNatural

access

flat
Sorry, I have to do that (Cf. Bluesette) otherwise I can't compute possibleTonalities properly ...

semiToneCount

sharp
Methode flat is not defined.
Exemple C flat diminishedSeventh..

printing

printOn: s

storeOn: s
N C flat flat storeString

testing

isDoubleFlat

MetaClass: PitchClassDoubleFlat class

Class: **PitchClassChord**

Environment: Siren
Superclass: Object
Category: Music-PitchClasses
Instance variables: root structure notes possibleTonalities

automatic access

root

root: aValue

rootPitchClass

structure

structure: aValue

printing

name
produces a printable name (without the sharps before symbols)

notesString

printOn: s

quotedPrintOn: s
as printOn but without the brackets [] and with quotes

storeOn: s
returns a String representation of the receiver from which the receiver can be reconstructed

structureString

note testing

allNotesInScale: aScale
The previous version couldnt deal with situations like
(FPChord newFromString: 'C 7 aug9') possibleTonalitiesInScaleClass: MajorScale

chordNotesAmong: list
select among given notes those that are chord notes

containsNote: n

nextIn: aCS

nonChordNotesAmong: list
select among given notes those that are chord notes

notes: aListOfNotes contains: anArrayOfIntervals

creation

format
format the structure. Assumes root is not nil

fromNotes: l
assumes the first note is the root

fromNotes: aList root: r
l isa list of pitch-classes.
This method computes the correct (and canonical) name for the chord

fromString: l
l is a list whose first element is a note and the rest a structure, in our standardized chord syntax

structureFromNotes: aList root: r
l isa list of pitch-classes.
This method computes the correct (and canonical) name for the chord

notes computation

computeAllNotes
computes the list of notes from the structure. The job is the opposite of what is written is method fromListOfNotes.
Assumes root is not nil.
| c |

c := FPChord newFromString: 'A halfDim7'.
c computeAllNotes.
c notes

computeDiminished

computeEleventh

computeFifth

computeNinth

computeRoot

computeSeventh

computeSixth

computeThird

computeThirteenth

notes

notesWithinOctave
accordingly, return the notes as in the case of a four note chord
with no higher dissonances

compute possible tonalities

= c

analyseIn: aScale
en majeure, mineure harm et min mel

computePossibleTonalities
self allInstancesDo: [:x| x computePossibleTonalities]

matchWith: c
((Chord new fromString: 'D min 7') computeAllNotes) format
matchAvec: ((N re majorScale genereAccordsPoly: 4) at: 3).

possibleTonalitiesInScaleClass: s
possible tonalities in scale class s.
(PitchClassChord newFromString: 'C ') possibleTonalitiesInScaleClass: MajorScale AnalysisList ({I de C MajorScale} {IV de G MajorScale} {V de F MajorScale})

There is a problem here, related to invalid scales. If you ask for
instance the possible tonalities of 'C# min', it will try to say that V of G# major is a
possible tonality. But the creation of G# major raises an error since it
is an invalid scale. For the moment I leave it as is because I would
like to have a model of reasoning in an object-oriented setting first

standardPossibleTonalities
possible tonalities in scale class major, minorMel and minorHarm
(Chord new fromString: 'C min') standardPossibleTonalities
ListeDAnalyses ({II de Sib MajorScale} {III de Lab MajorScale} {VI de Mib MajorScale} {I de Do HarmonicMinorScale} {IV de Sol HarmonicMinorScale} {I de Do MelodicMinorScale} {II de Sib MelodicMinorScale})

tonalitesCommunesAvec: unAccord
(FPChord newFromString: 'C min') tonalitesCommunesAvec: (FPChord newFromString: 'D b ')

tonalitesCommunesAvec: unAccord et: autreAccord
(FPChord newFromString: 'C min') tonalitesCommunesAvec: (FPChord newFromString: 'D b ')

transposing

beTransposedBy: int

transposeOf: interval

FP

natural

accessing

possibleScales

possibleTonalities

initializations

initAnalyse

initialize

resetNotes
reset notes to an empty collection when an important change occurred

accessing by intervals

theFifth

theFourth

theOctave

theSeventh

theThird

theTonic

testing ok

commonTonalitiesWith: aChord

hasA: anInterval
(FPChord newFromString: 'C min 7') hasA: #majorThird

hasADiminishedFifth

hasADiminishedSeventh

hasAMajorSeventh

hasAMinorSeventh

hasAnAugmentedFifth

hasNoFifth

hasNoSeventh

isAlterate

isDiminished

isDominant

isHalfDiminished

isMajor

isMinor

isSuspended

structureHasEitherOf: aList

structureHasNeitherOf: aList

mutations

@ o

asPlayableObject

defaultOctave

fondamental

inTessitura: qt

lapse: l

octave: n

startBeat: s duration: d
a default octave is used

startTime: s duration: d

tessitura

rootInTessitura: qt

other computations

intervalBetweenRoots: aChord
remark: not that this method answers the interval between
*the root *pitch classes*!!*

private-pact

bestTonalityInIsolatedCase
answer the most plausible tonality when the chor is isolated.
Later, I will rewrite it to bypass possibleTonalities for sake of efficience

MetaClass: PitchClassChord class

creation

new

newFromNotes: l
computes the name from the list of notes, according to our syntax

newFromNotesNames: aStringOfNoteNames

computes the name from the list of notes, according to our syntax
PitchClassChord newFromNotesNames: 'A C# E G' [La 7]

newFromString: l
computes the list of notes fom the name acording to our syntax

root: r structure: s

editing

openEditor

examples

allChordsFromListOfNotes: aList
self allChordsFromListOfNotes: (Array with: N do with: N mi with: N sol)
#(Do Mi #min #no5 #no11 #no9 #no7 #dim13 Sol #sus4 #no5 #sixth)
self allChordsFromListOfNotes: (Array with: N do with: N mi with: N sol with: N si)

commonTonalities

(PitchClassChord new fromString: 'C maj7')
possibleTonalities AnalysisList ({I of C MajorScale} {IV of G MajorScale} {VI of E HarmonicMinorScale})

(PitchClassChord new fromString: 'F maj7')
possibleTonalities AnalysisList ({I of F MajorScale} {IV of C MajorScale} {VI of A HarmonicMinorScale})

(PitchClassChord new fromString: 'E min 7')
possibleTonalities AnalysisList ({II of D MajorScale} {III of C MajorScale} {VI of G MajorScale} {IV of B HarmonicMinorScale} {II of D MelodicMinorScale})
—
(PitchClassChord new fromString: 'C maj7')
tonalitesCommunesAvec: (PitchClassChord new fromString: 'F maj7')
et: (PitchClassChord new fromString: 'E min 7') Set (C MajorScale)

examples

(PitchClassChord new fromString: 'Re maj7') notes OrderedCollection (Re Fa# La Do#)
(PitchClassChord new fromString: 'Re# maj7') notes OrderedCollection (Re# Fa## La# Do##)
(PitchClassChord new fromString: 'C') notes OrderedCollection (Do Mi Sol)
(PitchClassChord new fromString: 'D min 7 dim5') notes OrderedCollection (Re Fa Lab Do)
(PitchClassChord new fromString: 'C aug9') notes OrderedCollection (Do Mi Sol Sib Re#)
(PitchClassChord new fromString: 'C aug9 dim5') notes OrderedCollection (Do Mi Solb Sib Re#)
(PitchClassChord new fromString: 'C 13') notes OrderedCollection (Do Mi Sol Sib Re Fa La)
(PitchClassChord new fromString: 'C 13 aug9') notes OrderedCollection (Do Mi Sol Sib Re# Fa La)
(PitchClassChord new fromString: 'C 13 aug9 no7') notes OrderedCollection (Do Mi Sol Re# Fa La)
(PitchClassChord new fromString: 'C halfDim7') notes OrderedCollection (Do Mib Solb Sib)

'F min 7' asChord notes > OrderedCollection (F Ab C Eb)
'F min7' asChord notes > OrderedCollection (F A C)
'F m 7' asChord notes > OrderedCollection (F A C Eb)
'Fm 7' asChord notes > error
'F maj 7' asChord notes > error
'F m7' asChord notes > OrderedCollection (F A C)
'F maj 7' asChord notes > OrderedCollection (F A C Eb)
'F maj7' asChord notes > OrderedCollection (F A C E)
'F halfDim7' asChord notes > OrderedCollection (F Ab Cb Eb)
'F min dim5 7' asChord notes OrderedCollection (F Ab Cb Eb)
'F min b5 7' asChord notes OrderedCollection (F Ab Cb Eb)
'F dim7' asChord notes OrderedCollection (F A C Ebb)

PitchClassChord new fromNotes: (Array with: N do with: N mi with: N sol) Do
PitchClassChord new fromNotes: (Array with: N do with: N mi with: N sol sharp) Do #aug5
PitchClassChord new fromNotes: (Array with: N do with: N fa with: N sol) Do #sus4
PitchClassChord new fromNotes: (Array with: N do with: N mi with: N sol with: N la) Do #sixth
PitchClassChord new fromNotes: (Array with: N do with: N mi with: N la) Do #no5 #sixth
PitchClassChord new fromNotes: (Array with: N do with: N la) Do #no3 #no5 #sixth
PitchClassChord new fromNotes: (Array with: N do with: N mi with: N sol sharp with: N si) Do #aug5 #maj7
PitchClassChord new fromNotes: (Array with: N do with: N mi with: N sol flat with: N si flat) Do #dim5 7
PitchClassChord new fromNotes: (Array with: N do with: N mi flat with: N sol flat with: N si flat) Do #halfDim7
PitchClassChord new fromNotes: (Array with: N do with: N mi flat with: N sol flat with: N si flat flat) Do #dim7

PitchClassChord new fromNotes: (Array with: N do with: N mi flat) Do #min #no5
PitchClassChord new fromNotes: (Array with: N do with: N mi with: N fa with: N fa sharp) Do #no5 #no9 #no7 11 #aug11
PitchClassChord new fromNotes: (Array with: N do with: N sol flat with: N sol sharp) Do #no3 #dim5 #aug5

A chord of A. holdsworth
PitchClassChord new fromNotes: (Array with: N re sharp with: N fa sharp sharp with: N la with: N do sharp sharp) Re# #dim5 #maj7

(PitchClassChord new fromString: 'C min') standardPossibleTonalities OrderedCollection ({II de Sib MajorScale} {III de Lab MajorScale} {VI de Mib MajorScale} {I de Do HarmonicMinorScale} {IV de Sol HarmonicMinorScale} {I de Do MelodicMinorScale} {II de Sib MelodicMinorScale})

(PitchClassChord new fromString: 'Do') possibleTonalities
ListeDAnalyses ({V de Fa HungarianMinor} {VI de Mi HungarianMinor} {I de Do MajorScale} {IV de Sol MajorScale} {V de Fa MajorScale} {I de Do DoubleHarmonic} {II de Si DoubleHarmonic} {IV de Sol MelodicMinorScale} {V de Fa MelodicMinorScale} {IV de Sol Oriental} {IV de Sol NeapolitanMajor} {V de Fa HarmonicMinorScale} {VI de Mi HarmonicMinorScale} {II de Si NeapolitanMinor} {VI de Mi NeapolitanMinor})

(PitchClassChord new fromString: 'D min ') possibleTonalities
ListeDAnalyses ({I de Re HungarianMinor} {VII de Mib HungarianMinor} {II de Do MajorScale} {III de Sib MajorScale} {VI de Fa MajorScale} {III de Sib DoubleHarmonic} {IV de La DoubleHarmonic} {I de Re MelodicMinorScale} {II de Do MelodicMinorScale} {I de Re NeapolitanMajor} {I de Re HarmonicMinorScale} {IV de La HarmonicMinorScale} {I de Re NeapolitanMinor} {IV de La NeapolitanMinor})

exampleShort
'C min 7 dim5' asChord.
('C min 7 dim5 9' asChordOct: 4) notes
OrderedCollection (C4 Eb4 Gb4 Bb4 D5)

holdsworth

reallyAllChordsFromListOfNotes: aList
self reallyAllChordsFromListOfNotes: (Array with: N do with: N mi with: N sol)
OrderedCollection ([La #noRoot #min 7] [Si #noRoot #sus4 #no5 #no7 #dim9 #dim13] [Do] [Re #noRoot #sus4 #no5 7 9] [Mi #min #no5 #no11 #no9 #no7 #dim13] [Fa #noRoot #no3 #maj7 9] [Sol #sus4 #no5 #sixth] [La# #noRoot #no3 #dim5] [Do# #noRoot #min #dim5] [Re# #noRoot #no3 #no5 #dim9] [Fa# #noRoot #no3 #dim5 7 #dim9] [Sol# #noRoot #no3 #no5 #no11 #no9 #no7 #dim13] [Lab #noRoot #aug5 #maj7] [Sib #noRoot #no3 #no5 #no7 9 #aug11 #sixth] [Reb #noRoot #no3 #no5 #maj7 #aug9 #aug11] [Mib #noRoot #no5 #sixth] [Solb #noRoot #no3 #no5 #no9 #no7 #aug11])

Class: **Triad**

Environment: Siren
Superclass: Siren.PitchClassChord
Category: Music-PitchClasses

Instance variables: type

type

type

type: t

notes

notes: x

MetaClass: Triad class

Class: **OctaveDependentChord**

Environment: Siren
Superclass: Siren.PitchClassChord
Category: Music-PitchClasses
Instance variables: notesByInterval

*notesBytInterval provides a fast access to the chord notes
by the names of their intervals with the root. The calculation
is done once forever. On the contrary, for FPChord the
calculation is done all the times a note is required since
its rarely required.
Note that for all request (e.g. theFifth) the copy of the note is
answer). it is necessary in order to avoid some confusions. For instance,
the pitches (ODNotes) of an simple arpeggio on a chord ch may be
obtained by:*

```
o := OredredCollection new.
o add: ch theTonic; add: ch theThird; ch theFifth; theTonic.
```

*the first and last must be different objects in the case we will assign
them todifferents PlayableNotes.*

saving

save

storeOn: s

mutations

asMelody

asPlayableObject

lapse: l

startBeat: s duration: d

startTime: s duration: d

accessing

addNote: c

notes: value

removeNote: c

alteration

octave: o

octaveFromNote: n
set octave from new root or chord note

copy

copy

private-pact

closestDownTriadNoteTo: n
((FPChord newFromString: 'C maj7') @ 3) closestTriadNoteTo: N B flat @ 2

closestTriadNoteTo: n
((FPChord newFromString: 'C maj7') @ 3) closestTriadNoteTo: N B flat @ 2

closestUpTriadNoteTo: n
((PitchClassChord newFromString: 'C maj7') @ 3) closestTriadNoteTo: N B flat @ 2

containsLegalNote: n

downTriadNotes
return all triad notes within 2 octaves in a particular order

triadDownNoteLeadingTo: n
for the conflicting cases uses the priority given by the ordering of upAndDownTriadNotes

triadNoteLeadingTo: n
for the conflicting cases uses two priority criteria: the proximity and the ordering given by triadNotesLeadingTo:

triadNotesLeadingTo: n
answer the chord 'triad' notes (extend to 7?) within an octave,

that are leading notes to the given note n

triadUpNoteLeadingTo: n
for the conflicting cases uses the priority given by the ordering of upAndDownTriadNotes

upAndDownTriadNotes
return all triad notes within 2 octaves in a particular order

upTriadNotes
return all triad notes within 2 octaves in a particular order

initialize

initialize

resetNotes
extend super class method for notesByIntevals

notes by intervals

computeNotesByInterval
only for triad notes

notesByInterval

reallyTheFifth

reallyTheLowFifth

reallyTheLowOctave

reallyTheLowThird
attention: do not cofound with 'root downThird interval

reallyTheOctave

reallyTheThird

reallyTheTonic

theFifth

theLowFifth

theLowOctave

theLowThird

theOctave

theThird

theTonic

As yet unclassified

theLowSeventh
WARN: the calculatin of the seven is done in every request

FP

allPitchClasses

transposing

beTransposedOneStepDown

beTransposedOneStepUp

MetaClass: OctaveDependentChord class

cation

newWithValues: anArray

readFromFile

readFromFile: fileName

root: r notes: n

examples

example

example1
self example1

example2

Class: **OctaveDependentNote**

Environment: Siren

Superclass: Object

Category: Music-PitchClasses

Instance variables: oct pc midiPitch natural

Instances of this class represent octave dependent notes.
pc : the pitch class of the note (takes enharmonic spelling into account)
octave: integer from 0 to .., represent the octave.
Middle C is therefore represented by
pc : C
octave: 4
Defines methods to compute intervals, midiPitches, and lots
of thing that PitchClass implements too.
So why not make this class and PitchClass have
a common superclass, to factor out common computations ?
Ontological questions :
how to compute an interval between two octave-dependent notes,
since their ambitus may be arbitrarily large (no question of computing 25th intervals).
So intervals should be reduced to thirteenth, i.e. :
octaves should be taken into account only in a 0/1 manner

access

oct

oct: o

octave: o

pitchClass

pitchClass: aPc

alterations

flat

isDoubleFlat

isDoubleSharp

isFlat

isNatural

isSharp

natural

sharp

printing

printOn: s

storeOn: s
(N C sharp sharp octave: 3) storeString = '(N C sharp sharp @ 3)'

intervals

closestEnharmonic

diatonicStepsTo: aNote
returns the number of diatonic steps to aNote

following
the next diatonic note of self's natural note

newdiatonicStepsTo: aNote
returns the number of diatonic steps to aNote

nthFollowing: i
yields the nth diatonic note following self.
Takes octave shifts into account

nthPreceding: i
yields the nth diatonic note preceding self.
Takes octave shifts into account

olddiatonicStepsTo: aNote
returns the number of diatonic steps to aNote

oldEquals: x
important pour l'integrite des operations sur les melodies

preceding
the next diatonic note of self's natural note

semiToneCount
related to OctaveDependentNote fromMidiPitch

semiTonesWith: aNote

interval testing

intervalBetween: aNote
returns the interval between the two octaveDependentNotes.
Assume that aNote is really a OctaveDependentNote

intervalTypeBetween: aNote
returns the type of the interval between the two notes modulo one octave
(N C octave: 6) intervalTypeBetween: (N re octave: 3)

isLessThanA: interval from: od
ascending or descending direction

realIntervalTypeBetween: aNote
returns the type of the interval between the two notes
(N C octave: 6) realIntervalTypeBetween: (N C octave: 3)

testIntervalsFrom: pitch1 to: pitch2
(N C sharp @ 3) testIntervalsFrom: (N A @ 1) to: (N A @ 4)

public intervals

alterate: note toReach: i
Join semitons to note in order to obtain the required number of semitones with self.
Redefined here because interval computation must take octave into account ...

alterateBelow: note toReach: i
Join semitons to note in order to obtain the required number of semitones with self.
Redefined here because interval computation must take octave into account ...

augmentedEleventh

augmentedFifth

augmentedFourth

augmentedNinth

augmentedOctave

augmentedSecond

diminishedFifth

diminishedNinth

diminishedOctave

diminishedSeventh

diminishedThirteenth

eleventh

fifth

flatFifth

flatNinth

flatThirteenth

fourth

majorNinth

majorSecond

majorSeventh

majorSixth

majorThird

majorThirteenth

minorNinth

minorSecond

minorSeventh

minorSixth

minorThird

minorThirteenth

ninth

octave

perfectEleventh

perfectFifth

perfectFourth

second

seventh

sixth

thirteenth

unison

transpose

beDownAnOctave

beTransposedBy: i

beTransposedOneStepDown
used by the score editor as diatonic step!

beTransposedOneStepUp
used by the score editor as diatonic step!

beUpAnOctave

downAnOctave

setPitchTo: aPitch

switchTo: aNote

transposeOf: anInterval
three cases: integer (+/-), method name (= ascending interval), or interval object

upAnOctave

comparing

< aNote

<= aNote

= x

> aNote

>= aNote

hash

isbetween: n1 and: n2

isNearerTo: n1 than: n2

midiPitchEquals: aNote

pitchEqual: aNote

theNearestPitch: list
(N A flat @ 2) theNearestPitch: (Array with: (N C @ 2) with: (N C @ 3))

mutations

duration: d

leading

isLeadingToneTo: n

lowerLeadingToneInScale: s

upperLeadingToneInScale: s

private-pacts

respectsBassTessitura

modifying

inTessitura: qualTessitura
modify octave according to the given tessitura

mutations more

dottedEighth

dottedFull

dottedHalf

dottedQuarter

eighth

eighthInTriplet

full

half

quarter

quarterInTriplet

sixteenth

accessing delegation

downAndUpOctaves: odNote

constraining

intervalTypeModuloOctaveBetween: n

MetaClass: OctaveDependentNote class

creation

from: aNote to: n2

fromMidiPitch: n
yields the note that has n as midiPitch. Only natural and sharp notes are created as there is no possible way of knowing the intention

named: s
self named: 're#2'

octave: o pc: pc

examples

examples
(N do octave: 3) minorThird sharp sharp E#3

(N C octave: 3) semiTonesWith: (N la octave: 4) 21

(#(do re mi fa sol la si) collect: [:x | (N perform: x) octave: 3])
collect: [:y | y - #octave]

(N B octave: 3) intervalBetween: (N B flat octave: 4)
(N B octave: 3) intervalBetween: (N B flat octave: 2)

(N C octave: 3) intervalBetween: (N C octave: 2) descending octave

(N C octave: 2) intervalBetween: (N C octave: 3) octave

vocal ranges

altoRange
retourne la collection des notes de l'alto

altoRangeInScale: aScale
OctaveDependentNote sopranoRangeInScale: (N do sharp minorScale)

baseRange
retourne la collection des notes de la basse

baseRangeInScale: aScale
OctaveDependentNote sopranoRangeInScale: (N do sharp minorScale)

sopranoRange
retourne la collection des notes du soprano

sopranoRangeInScale: aScale
OctaveDependentNote sopranoRangeInScale: (N do sharp minorScale)

tenorRange
retourne la collection des notes du tenor

tenorRangeInScale: aScale
OctaveDependentNote sopranoRangeInScale: (N do sharp minorScale)

Class: **MusicalDescendingInterval**

Environment: Siren
Superclass: Siren.MusicalInterval
Category: Music-PitchClasses

La theorie des intervalles "descendants" n'est pas tres claire:
Que voudrait dire DescendingInterval topIfBottomIs: N C ?
(quelle difference avec ascending ?).
Introduit juste pour les besoins de la cause, mais un peu incoherent.
Herite tout de MusicalInterval, et donc se comporte presque pareil.
Utilisee uniquement en creation dans la methode intervalBetween: de OctaveDependentNote.
Il faudrait d'ailleurs modifier aussi la methode similaire de PitchClass

testing

isAscending

isDescending

computing notes

bottomIfTopIs: aNote
yields the note for which I am the interval self

topIfBottomIs: aNote
yields the note for which I am the interval self

mutations

ascending

descending

MetaClass: MusicalDescendingInterval class

examples

example
(N C octave: 3) intervalBetween: (N D octave: 2) descending Minor seventh
(N B flat octave: 3) intervalBetween: (N B flat flat octave: 2) descending augmented octave

---

Class: **PitchClassSharp**

Environment: Siren
Superclass: Siren.PitchClassAltered
Category: Music-PitchClasses
Instance variables: sharp

intervals

alterate: note toReach: i

alterateBelow: note toReach: i

closestEnharmonic

semiTonesToNatural

saving

representBinaryOn: s

printing

printOn: s

storeOn: s
N C sharp storeString

testing

isSharp

accessing

flat

semiToneCount

sharp

sharp: value

MetaClass: PitchClassSharp class

Class: **PitchClassNatural**

Environment: Siren
Superclass: Siren.PitchClass
Category: Music-PitchClasses
Instance variables: semiToneCount nom name following preceding sharp flat

intervals

alterate: note toReach: i
Join semitons to note in order to obtain the required number of semitones with self

alterateBelow: note toReach: i
Join semitons to note in order to obtain the required number of semitones with self

closestEnharmonic

semiTonesToNatural

semiTonesWithNaturalNote: aNote
number of semitones between self and aNote

semiTonesWithNaturalNoteBelow: aNote
number of semitones between aNote and self

accessing

flat

flat: value

name

name: x

natural

nom

nom: value

sharp

sharp: value

private-accessing

following

following: value

preceding

preceding: value

semiToneCount

semiToneCount: s

printing

nameIs: aSymbol

printOn: s

saving

representBinaryOn: s

MetaClass: PitchClassNatural class

Music-PitchScales

Class: **Scale**

Environment: Siren
Superclass: Object
Category: Music-PitchScales
Instance variables: root notes
Class variables: AllNotes

accessing

allNotes

noteAfter: aNote
aNote can be a pitchClass, a pitch (ODNote) or even a PlayableNote. coputation takes into accont circularity and is approximative wrt aNote alterations

noteBefore: aNote
aNote can be a pitchClass, a pitch (ODNote) or even a PlayableNote. coputation takes into accont circularity and is approximative wrt aNote alterations

notes

root

root: value

scale tone chords

arpeggioFrom: startNote poly: n interval: int
yields n notes from startNote by intervals of int

chordFromNotes: l

following: noteDepart by: x
rend la x ieme note a partir de noteDepart dans mes notes

generateChordsPoly: n
generate the list of 7 scale tone chords from my notes with polyphony n

generateChordsPoly: n inOctave: oct
generate the list of 7 scale tone chords from my notes with polyphony n

indexDe: uneNote

scaleToneTriads
N C majorScale scaleToneTriads
(NeapolitanMinor root: N D) scaleToneTriads

octave-dependent notes

ascendingFromOctave: o
first note is in octave o. others follow (and may shift octave!)

asPitchesInOctave: o

closestPitchClassTo: aPc

decendingFromOctave: o
first note is in octave o. others follow (and may shift octave!)

diatonicNotesFrom: dep to: arr
assumes a and b belong to the scale.
N re minorScale diatonicNotesFrom: (N do sharp octave: 3) to: (N re octave: 4)

octave: o

computing notes

computeNotes
intervalList depends on the type of the scale. It is redefined in each subclass of Scale

intervalList
yields the list of interval. Redefined in subclasses

leadingNote
N la minorScale leadingNote

reverseNotes
first is root, others following descending order

tonic
N re minorScale tonic

printing

printOn: s

storeOn: s

degres

degreDeAccord: unAccord

degreeOfNote: aNote

transpose

transposeOf: anInterval

testing

= g

containsAllOfTheseNotes: listOfNotes

containsNote: n

isMajor

isMinor

isValid

querying

flatNotesInSignature
Warning: doubles sharps are not sharps!

numberOfFlats
Attention les doubles bemols ne sont pas des bemols!

numberOfSharps
Attention les doubles dieses ne sont pas des dieses!

sharpNotesInSignature
Warning: doubles sharps are not sharps!

comparing

hash

Paleo

costForPitch: n

enharmonicScale

pitchClassFor: p
N C majorScale pitchClassFor: N A

MetaClass: Scale class

creation

askScale
self askScale

root: aNote

standardScales

warn: s

examples

example

N la flat majorScale notes -> #(Lab Sib Do Reb Mib Fa Sol)
N do harmonicMinorScale notes -> #(Do Re Mib Fa Sol Lab Si)

N re majorScale numberOfSharps -> 2

N do majorScale generateChordsPoly: 4 ->
OrderedCollection ([Do #maj7] [Re #min 7] [Mi #min 7] [Fa #maj7] [Sol 7] [La #min 7] [Si #halfDim7])

N do majorScale generateChordsPoly: 6
-> OrderedCollection (Do #maj7 9 11 Re #min 7 9 11 Mi #min 7 #dim9 11 Fa #maj7 9 #aug11 Sol 7 9 11 La #min 7 9 11 Si #min #dim5 7 #dim9 11)

N do majorScale generateChordsPoly: 7
-> OrderedCollection (Do #maj7 9 11 13 Re #min 7 9 11 13 Mi #min 7 #dim9 11 #dim13 Fa #maj7 9 #aug11 13 Sol 7 9 11 13 La #min 7 9 11 #dim13 Si #min #dim5 7 #dim9 11 #dim13)

N re HarmonicMinorScale generateChordsPoly: 3

N do majorScale degreDeAccord: (FPChord new fromString: 'Fa min 7')
-> 4

interval list

intervalList

Paleo class initialize

initialize

Class: **PseudoMinorScale**

Environment: Siren
Superclass: Siren.Scale
Category: Music-PitchScales

MetaClass: PseudoMinorScale class

interval list

intervalList

Class: **Oriental**

Environment: Siren
Superclass: Siren.Scale
Category: Music-PitchScales

MetaClass: Oriental class

interval list

intervalList
tire de McLaughlin et le Mahavishnu Orchestra

Class: **NeapolitanMajor**

Environment: Siren
Superclass: Siren.Scale
Category: Music-PitchScales

MetaClass: NeapolitanMajor class

interval list

intervalList
tire de McLaughlin et le Mahavishnu Orchestra

Class: **HungarianMinor**

Environment: Siren
Superclass: Siren.Scale
Category: Music-PitchScales

MetaClass: HungarianMinor class

examples

exemple

(HungarianMinor root: N do) notes -> #(Do Re Mib Fa# Sol Lab Si)

(HungarianMinor root: N do) generateChordsPoly: 4
-> OrderedCollection (Do #min #maj7 Re #dim5 7 Mib #aug5 #maj7 Fa# #dim5 #dim7 Sol #maj7 Lab #maj7 Si #min #dim7)

(FPChord newFromString: 'D min ') possibleTonalitiesInScaleClass: self ->
ListeDAnalyses ({I de Re HungarianMinor} {VII de Mib HungarianMinor})

interval list

intervalList
tire de McLaughlin et le Mahavischnu Orchestra

Class: **MajorScale**

Environment: Siren
Superclass: Siren.Scale
Category: Music-PitchScales

Paleo

costForPitch: n

enharmonicScale

MetaClass: MajorScale class

interval list

allIntervals

intervalList

Class: **NeapolitanMinor**

Environment: Siren
Superclass: Siren.Scale
Category: Music-PitchScales

MetaClass: NeapolitanMinor class

interval list

intervalList
tire de McLaughlin et le Mahavishnu Orchestra

Class: **MelodicMinorScale**

Environment: Siren
Superclass: Siren.Scale

Category: Music-PitchScales

Paleo

costForPitch: n

enharmonicScale

MetaClass: MelodicMinorScale class

interval list

allIntervals

intervalList

Class: **DoubleHarmonic**

Environment: Siren
Superclass: Siren.Scale
Category: Music-PitchScales

MetaClass: DoubleHarmonic class

interval list

intervalList
tire de McLaughlin et le Mahavishnu Orchestra

Class: **HarmonicMinorScale**

Environment: Siren
Superclass: Siren.Scale
Category: Music-PitchScales

Paleo

costForPitch: n

enharmonicScale

MetaClass: HarmonicMinorScale class

interval list

allIntervals

intervalList

Class: **ChromaticScale**

Environment: Siren
Superclass: Siren.Scale
Category: Music-PitchScales

MetaClass: ChromaticScale class

interval list

intervalList

Music-Sound

Class: **Sound**

Environment: Siren
Superclass: Siren.Function
Category: Music-Sound
Indexed variables: objects
Imports: private SoundConstants.*

Instances of the subclasses of Sound are used to represent sound objects.
The abstract class Sound is vacuous.
Sounds use their Function and DurationEvent behaviors actively.

testing

isSound
Answer true.

initialize-release

initialize
Set up a default sound.

cue accessing

addCueNamed: cName from: start to: stop
Add the given named cue region to the receiver's list.

cueList
Answer the receiver's 'cueList'.

cueList: CollectionOfCues
Set the receiver's 'cueList' to be the given OrderedCollection of (#name -> (start to: stop)) cues.

cueNamed: cName
Answer a sound derived from the receiver using the samples between the named cue points.

cueRegionNamed: cName
Answer a indices for the samples between the named cue points.

MetaClass: Sound class

instance accessing

named: aName
Answer the sound by the given name, or nil.

named: aName put: aSound
Put the given sound in the shared dictionary under the given name.

utilities

fromFile: nameString
Open the given file (AIFF, IRCAM, NeXT, or SPARC soundfile format)

playFile: theName
Play the sound file with the given name

Class: **SampledSound**

Environment: Siren
Superclass: Siren.Sound
Category: Music-Sound
Instance variables: name rate format channels samplesInMemory firstIndex
Indexed variables: objects
Class variables: MagicNumbers MaxIntSample MinIntSample
Imports: private SoundConstants.*

Instances of SampledSound represent digitally-sampled sound objects. Their "samples" are stored in the "data" instance variable inherited from Function. They can be read/written to/from files (using SoundFile objects), displayed (in SoundViews) and played (via SoundPorts).

The class SampledSound is concrete and assumes 16-bit linear encoding of samples; there are subclasses for floating-point and 8-bit Mu-law sample types. There are useful class methods for creating a number of standard sounds such as silence, impulses, swept sine waves, etc. Note that SampledSound uses the 'range' instance variable inherited from Function as its (integer) size.

Instance variables:
name the file name or object ID
rate the sample rate
format the sample format, e.g., #lin16Bit
channels the # of channels
samplesInMemory The number of samples held in memory (may be smaller than size for very large sounds)
firstIndex The sample index of the start of the in-memory samples (may be non-zero for very large "paged" sounds)

Other properties, such as the sound's (optional) file name, its sample rate, or the number of channels, are stored in the property list dictionary that is inherited by virtue of being an event subclass.

KNOWN BUGS:

Note that not all of this class is finished--there are methods that call non-existent user primitives and have no Smalltalk implementations (like the heavy DSP).

accessing

allocateData

channels
Answer the receiver's 'channels'.

channels: anObject
Set the receiver's instance variable 'channels' to be anObject.

copy
Answer a 'clean' copy of the receiver.

copyAllButSamples
Answer a 'clean' copy of the receiver, but don't write in the samples yet.

copyFrom: start to: stop
Answer a copy of the receiver for the given sample range.

cueNamed: cName
Answer a sound derived from the receiver using the samples between the named cue points.

envelope
Answer the receiver's envelope.

envelope: anEnv
Set the receiver's envelope property.

file
Answer the sound file for the receiver (if present).

firstIndex
Answer the receiver's 'firstIndex'.

firstIndex: anObject
Set the receiver's instance variable 'firstIndex' to be anObject.

floatsFrom: aStart to: aStop
Answer a copy of the receiver's data within the given sample range.

format
Answer the receiver's 'format'.

format: fSymbol

Set the receiver's instance variable 'format' to be fSymbol.

frameRate
Answer the receiver's 'rate'.

from: aStart to: aStop
Answer a copy of the receiver within the given time range.

fromSample: start toSample: finish
Answer a copy of the receiver within the given sample range.

hasFile
Answer whether the receiver is stored on a file of the same name.

longFormat
Answer the receiver's format as a descriptive string.

name
Answer the receiver's 'name'.

name: anObject
Set the receiver's instance variable 'name' to be anObject.

rate
Answer the receiver's 'rate'.

rate: anObject
Set the receiver's instance variable 'rate' to be anObject.

sampleRate
Answer the receiver's 'rate'.

samples
Answer the instance variable 'data'.

samples: theArray
Accept the argument, 'theArray', (a Byte, Word, or Float array, or an UninterpretedBytes) as the new instance variable 'samples'.

samplesInMemory
Answer the receiver's 'samplesInMemory'.

samplesInMemory: anObject
Set the receiver's instance variable 'samplesInMemory' to be anObject.

sampleSize
Answer the size in bytes of the receiver's samples (2).

size
Return the receiver's size in sample frames.

size: aNumber
Set the receiver's 'size' in samples.

sizeInBytes
Answer the size in bytes of the receiver.

sizeInFrames
Return the receiver's size in sample frames.

sample accessing

at: anIndex

Answer the value at the given index (between 0 and 1).

at: anIndex put: aValue
Stuff the given sample at the given index (after lots of checking and testing).

cutFrom: startSample to: stopSample
Cut the designated section from the receiver; answer a new sound (!).

floatSampleAt: index
Answer the sample at the given index as a normalized floating-point number.

floatSampleAt: index put: aValue
Stuff the given floating-point sample at the given index after scaling.

intSampleAt: index
Answer the sample at the given index as an integer.

intSampleAt: index put: aValue
Stuff the given integer sample at the given index.

paste: aSound from: startSample to: stopSample at: index
Paste the given sound into the receiver; answer a new sound (!).

sampleAt: index
Answer the sample at the given index, mapping and paging as necessary.

sampleAt: anIndex put: aValue
Stuff the given sample at the given index (after lots of checking and testing).

interpolating

autoCorrelationBetween: index1 and: index2 length: length
Answer the cummulative error between the portions of my waveform starting at the given two indices and extending for the given length. The larger this error, the greater the difference between the two waveforms.

errorBetween: sampleArray1 and: sampleArray2
Answer the cummulative error between the two sample arrays, which are assumed to be the same size.

fadeInOver: fTime
Apply a fade-in ramp to the receiver with the given time.

fadeOutOver: fTime
Apply a fade-in ramp to the receiver with the given time.

interpolatedWindowAt: anIndex width: nSamples
Return an array of N samples starting at the given index in my data.

sampledFrom: aStart to: aStop into: anArray
Place a down-sampled version of the receiver in the argument array.

sampledTo: anArray
Place a down-sampled version of the receiver in the argument array.

sampleMaxFrom: aStart to: aStop into: anArray
Place a version of the root-mean-square energy of the receiver in the argument array.

merging

extractChannel: cNumber into: aSound
Extract the given channel from the receiver into the argument sound.

mergeChannel: cNumber from: aSound
Merge the argument sound into the given channel of the receiver.

envelopes

average
Answer the average sample value of the receiver (normally the DC offset).

computeEnvelope: type
Compute a sample envelope for the receiver with 128 points per second.

computePeakEnvelope
Compute a windowed peak sample envelope for the receiver with 1024 points per 4 seconds.

computeRMSEnvelope
Compute a windowed root-mean-square sample envelope for the receiver.

edit
Edit the receiver.

max
Answer the maximum value of the samples.

offsetBy: aFactor
Answer a copy of the receiver offset by the given factor.

offsetFrom: aStart to: aStop by: anOffset
Answer a copy of the receiver offset by the given factor.

peakEnvelope
Answer the receiver's peak-detected envelope, or compute a new one.

peaksFrom: aStart to: aStop into: anArray
Place a version of the root-mean-square energy of the receiver in the argument array.

readEnvelope: type
Try to read the sample envelope for the receiver from a file named XXX.env.

rmsEnvelope
Answer the receiver's envelope, or compute a new one.

rmsFrom: aStart to: aStop into: anArray
Place a version of the root-mean-square energy of the receiver in the argument array.

rmsTo: anArray
Place a version of the root-mean-square energy of the receiver in the argument array.

scaleBy: aScaleFactor
Scale the receiver's samples by the given factor.

scaledBy: aScaleFactor
Answer a copy of the receiver scaled by the given factor.

scaledByEnvelopeArray: anArray
Answer a copy of the receiver scaled by the given envelope array.

scaledByFunction: aFunction
Answer a copy of the receiver scaled by the given breakpoint envelope expressed as points.

scaledFrom: aStart to: aStop by: aScaleFactor
Answer a copy of the receiver scaled by the given factor.

scaledFrom: aStart to: aStop byEnvelopeArray: anArray

Answer a copy of the receiver scaled by the given envelope array.

scaledFrom: aStart to: aStop byFunction: aFunction
Answer a copy of the receiver scaled by the given breakpoint envelope expressed as points.

scaleFrom: aStart to: aStop by: aScaleFactor
Scale the receiver's damples by the given factor.

writeEnvelope: type
Write the sample envelope for the receiver to a file named XXX.env or XXX.pk.

enumerating

do: aBlock
Evaluate aBlock with each of the receiver's elements as the argument.

printing

printOn: aStream
pretty-print the receiver.

printSamples
pretty-print a few samples.

printSamplesOn: aStream
*Pretty-print a few samples--32 by default, 1024 if shift-down, *all* of ctrl and shift down.*

storeOnFileNamed: aName
Save the receiver on the sound file named by the argument.

private

getData
Read samples in from the file if none in memory.

mapSampleIndex: index
Map the given sample index according to the 'page' (firstIndex) of the receiver.

privateSampleAt: anIndex
Answer the sample at the given index as an integer.

privateSampleAt: anIndex in: cPtr bigEndian: isBE
Answer the sample at the given index as an integer,
assuming the output is a cPointer; handle endianness.

privateSampleAt: anIndex in: cPtr put: value bigEndian: isBE
Answer the sample at the given index as an integer,
assuming the output is a cPointer; handle endianness.

privateSampleAt: anIndex put: aValue
Stuff the given sample at the given index (safely).

readSamplesFrom: index
Read in samples from disk starting a bit before the given index.

realSound
Answer the real sound for the receiver (overridden in virtual subclasses).

scaleSampleIndex: index
Scale the given sample index according to the sample size of the receiver.

play/record

nextBufferInto: outBuffer frames: bufferSize channels: outChannels format: sformat startingAt: startFrame
Copy data from the receiver into the given output buffer; answer the current sample index.

play
Play the receiver out over the default sound port.

testing

hasGaps
Answer whether the receiver has any gaps.

isComposite
Answer whether the receiver is a composite sound.

isEmpty
Answer whether the receiver is empty.

isVirtual
Answer whether the receiver is a virtual sound.

initialize-release

initialize
Set up a default sound.

MetaClass: SampledSound class

instance creation

default
Answer the default empty sound.

duration: aDur named: nameString rate: aRate channels: aChannels format: aFormat
Answer a new 16-bit linear sound with the given properties.

duration: aDur rate: aRate channels: aChannels
Answer a new sound with the given properties.

duration: aDur rate: aRate channels: aChannels format: aFormat
Answer a new sound with the given properties.

fromData: anArray named: nameString rate: aRate channels: aChannels format: aFormat
Answer a new sound from the given data.

fromData: anArray rate: aRate channels: aChannels format: aFormat
Answer a new 16-bit linear sound from the given data.

fromDblData: anArray rate: aRate channels: aChannels size: siz
Answer a new sound from the given data.

headerFromFile: nameString
Open the given file (EBICSF, NeXT, or SPARC soundfile format)

named: aName size: aSize format: formatSymbol
Answer a new Sound with the given name and storage size.

named: nameString size: size rate: rate channels: chans format: formatSymbol

Answer a new Sound with the given storage size (in sample frames), etc.

named: nameString size: size rate: rate channels: chans format: formatSymbol data: data
Answer a new Sound with the given storage size (in sample frames), etc.

size: size format: formatSymbol channels: chans
Answer a new Sound with the given storage size, etc.

size: size rate: aRate channels: chans
Answer a new Sound with the given storage size, etc.

size: size rate: rate channels: chans format: formatSymbol
Answer a new Sound with the given storage size, etc.

size: size rate: rate format: formatSymbol channels: chans
Answer a new Sound with the given storage size, etc.

examples

immediateInspect
Answer a sampled sound from immediate data; inspect it.

rmsViewExample
Read a sampled sound from the file, take the rms into 64 values, and edit that.

sweepExample
Open a sound view on a swept sine wave.

sweepView
Open a sound view on a swept sine wave.

standard wave forms

constantOfDur: dur value: value rate: rate chans: chans
Answer a sound with a constant value.

expSweepDur: dur rate: rate from: start to: stop chans: chans
Answer a SampledSound with a swept sine wave.

impulseOfDur: dur width: width rate: rate chans: chans
Create a sound with an impulse of the given characteristics.

linearSweepDur: dur rate: rate from: start to: stop chans: chans
Answer a StoredSound with a swept sine wave.

pulseTrainDur: dur rate: rate freq: freq width: width chans: chans
Answer a StoredSound with a pulse train.

pulseTrainDur: dur rate: rate freq: freq width: width chans: chans zero: zero
Answer a StoredSound with a pulse train.

rampDur: dur rate: rate chans: chans
Answer a StoredSound with a single ramp of samples.

sawtooth
Answer a StoredSound with raw sawtooth samples.

sawtoothDur: dur rate: rate freq: freq chans: chans
Answer a StoredSound with raw sawtooth samples.

sineDur: dur rate: rate freq: freq chans: chans
Answer a StoredSound with a pulse train.

squareDur: dur rate: rate freq: freq chans: chans
Answer a StoredSound with a square wave.

sweepDur: dur rate: rate from: start to: stop chans: chans
Answer a StoredSound with a swept sine wave.

class constants

formatSymbol
Answer the symbolic code used for the format of the receiver's instances.

maxSample
Answer the maximum value of the receiver class.

minSample
Answer the minimum value of the receiver class.

Class: **FloatSound**

Environment: Siren
Superclass: Siren.SampledSound
Category: Music-Sound
Indexed variables: objects

Instances of FloatSound are used for sounds with 32-bit floating-point numbers as samples. There are behaviors for mapping into other formats.

accessing

format
Return the receiver's format--a symbol constant.

sampleSize
Answer the size in bytes of the receiver's samples (4).

private

privateSampleAt: index
Answer the sample at the given index.

privateSampleAt: index put: aValue
Stuff the given sample into the data array at the given index (no checking).

sample accessing

intSampleAt: index
Answer the sample at the given index as a scaled 16-bit integer.

intSampleAt: index put: aValue
Put the given integer sample at the given index after scaling.

MetaClass: FloatSound class

class constants

formatSymbol
Answer the symbolic code used for the format of the receiver's instances.

maxSample
Answer the maximum value of the receiver class.

minSample
Answer the minimum value of the receiver class.

examples

fileExample
Answer a typical float sound read in from a file.

Class: **ComponentSound**

Environment: Siren
Superclass: Siren.Sound
Category: Music-Sound
Instance variables: start stop sound offset
Indexed variables: objects

A ComponentSound is used as a component (a "splice" element, if you will).
It represents another sound that "composes" the composite.

Instance Variables:
sound the "subject" sound
start the starting sample in the composite
stop the ending sample in the composite
offset the offset into the selection

accessing

offset
Answer the receiver's 'offset'.

offset: anObject
Set the receiver's 'offset' to be anObject.

sound
Answer the receiver's 'sound'.

sound: anObject
Set the receiver's 'sound' to be anObject.

start
Answer the receiver's 'start'.

start: anObject
Set the receiver's 'start' to be anObject.

stop
Answer the receiver's 'stop'.

stop: anObject
Set the receiver's 'stop' to be anObject.

printing

printOn: aStream
Pretty-print the receiver on the argument.

testing

includes: anIndex
Answer whether the argument is within the range of the receiver.

MetaClass: ComponentSound class

instance creation

on: sound from: start to: stop
Answer a new instance initialized with the arguments.

on: sound start: start stop: stop
Answer a new instance initialized with the arguments.

on: sound start: start stop: stop offset: offset
Answer a new instance initialized with the arguments.

Class: **VirtualSound**

Environment: Siren
Superclass: Siren.SampledSound
Category: Music-Sound
Instance variables: source
Indexed variables: objects

An instance of VirtualSound is a "reference" to another sound, typically by a named cue list entry.

Instance Variables:
source the sound to which the VirtualSound points

accessing

cue: cueName
Set the cue name of the receiver

source
Answer the receiver's source sound.

source: aSound
Set the receiver's source sound.

start: start
Set the argument as the starting sample of the receiver.

stop: stop
Set the argument as the ending sample of the receiver.

sample accessing

sampleAt: index put: aValue
*Stuff the given sample at the given index **after transforming into a 'real' sound**.*

private

getReal
Answer a 'real sound' based on the receiver.

mapSampleIndex: index
Map the given sample index according to the 'page' (firstIndex) of the source and the receiver's relative offset.

realSound
Answer the real sound for the receiver.

testing

isVirtual
Answer whether the receiver is a virtual sound.

MetaClass: VirtualSound class

instance creation

from: source cue: cName
Answer a VirtualSound derived from the argument and named cue region.

examples

exampleView
Open a sound view on a ramp with a chunk cut out of it.

Class: **GapSound**

Environment: Siren
Superclass: Siren.VirtualSound
Category: Music-Sound
Instance variables: cutList
Indexed variables: objects

An instance of GapSound can be used to represent a sound with samples deleted from it (i.e., a gap).
It uses its cut list instance variable to maintain sample ranges that have been deleted from the sound that comprises it (the source).
It responds to sampleAt: and sampleAt:put: just like other sounds, but stores only its cut list on files unless explicitly told to store samples.

Instance Variables:
cutList the list of deleted sections by sample index i.e., pointers denoting sections that have been

deleted. For example, having a cutList of (1000 2000) means that samples (1000, 2000] have been virtually cut.

private

mapSampleIndex: index
Map the given sample index according to the cut list (i.e., deleted sections) of the receiver.

testing

hasGaps
Answer whether the receiver has any gaps.

accessing

allocateData
no-op

cutFrom: startSample to: stopSample
Cut the designated section from the receiver.

samples

MetaClass: GapSound class

instance creation

on: source cutFrom: start to: stop
Answer a CompositeSound derived from the argument missing the given range.

examples

exampleView
Open a sound view on a ramp with a chunk cut out of it.

Class: **CompositeSound**

Environment: Siren
Superclass: Siren.VirtualSound
Category: Music-Sound
Instance variables: components
Indexed variables: objects

An instance of CompositeSound can be used to represent a sound constructed by "splicing together" sections of other sounds.
It uses its components collection to maintain sounds and sample ranges that have been pasted together. It responds to sampleAt: and sampleAt:put: just like other sounds, but stores only its components list on files unless explicitly told to store samples.

Instance Variables:
components the "sub-sounds" of the CompositeSound

sample accessing

sampleAt: index
Answer the sample at the given index, mapping and paging as necessary.

private

rippleUpFrom: start adding: insert
Shift all the cues above the given one up by the given insert count.

accessing

on: aSound
Set the receiver's source sound.

paste: sound from: start to: stop at: index
Paste the designated section into the receiver.

testing

isComposite
Answer whether the receiver is a composite sound.

MetaClass: CompositeSound class

instance creation

on: source paste: newSound from: start to: stop at: index
Answer a CompositeSound derived from the argument pasting in the given sound at the given index.

examples

exampleView
Open a sound view on a ramp with a chunk of a sine pasted into it.

Music-Support

Class: **Timer**

Environment: Siren
Superclass: Model
Category: Music-Support
Instance variables: startTime accumulator lastUpdate interval running name

accessing

accumulator

accumulator: aValue

interval

interval: aValue

lastUpdate

lastUpdate: aValue

name

name: aValue

running

running: aValue

startTime

startTime: aValue

control

play
Start a timer

reset
Start a timer

restart
Start a timer

scheduleAt: aTime
update a timer

start
Start a timer

stop
Start a timer

MetaClass: Timer class

Class: **SEventQueue**

Environment: Siren

Superclass: SequenceableCollection
Category: Music-Support
Instance variables: first last
Indexed variables: objects

An SEventQueue is a doubly-linked list that has methods for inserting elements in time-sorted order. These are used for real-time schedules because they can have faster search methods than using SortedCollections of OrderedCollections for the EventScheduler.
The current implementation uses a simple linear search. For really big schedules, this could be made faster with a binary search or tree-based schedule.

Instance Variables:
first The first event in the Q
last The last event in the Q

accessing

add: eventAss
Add the given eventAssociation to the receiver in the right place.

add: eventAss loop: aBool
Add the given eventAssociation to the receiver in the right place.

asOrderedCollection
Answer a copy of the receiver's items

detect: aBlock
Walk the list looking

do: aBlock
Refer to the comment in Collection|do:.

isEmpty

itemNamed: theName
Answer a copy of the receiver's items

ready: now
Answer whether there's an event ready within 5000 usec of the given time

removeAllSuchThat: aBlock
Remove each element for which aBlock evaluates to true.

removeFirst
Remove and answer the first item from the list.

size
Answer how many elements the receiver contains.

private

locateSlotFor: start
Find the proper slot for inserting a new event with the given start time.

printing

printOn: aStream

Append to the argument aStream a sequence of characters that identifies the receiver.

MetaClass: SEventQueue class

Class: **EditorModel**

Environment: Siren
Superclass: Model
Category: Music-Support
Instance variables: selections

MetaClass: EditorModel class

Class: **EventAssociation**

Environment: Siren
Superclass: Association
Category: Music-Support

An EventAssociation is an Association with some special semantics for Siren. It can be created with the '=>' message to a Number, meaning start-time assicoated with event.

accessing

, anArgument
Add the argument as a property of the receiver.

event
Answer the receiver's event (value).

start
Answer the receiver's start (key).

start: aTime
Set the receiver's start (key).

stop
Answer the stop time of the event association.

time
Answer the receiver's start (key).

printing

printOn: aStream
Append to the argument, aStream, the two elements of the EventAssociation separated by a double-right arrow (=>).

comparing

< anAssociation
Handle events, associations, and time/order sorting.

= anAssociation
If the argument's not an association, compare it to the receiver's value, otherwise answer whether the receiver's key and value are both equal to the argument's.

MetaClass: EventAssociation class

Class: **ScheduleRecord**

Environment: Siren
Superclass: Association
Category: Music-Support
Instance variables: next previous loop

A ScheduleRecord is used as an entry in the SEventQueue.
It's a doubly-linked record with a key (the start time) and value (the scheduled event).

Instance Variables
next the next record in the Q (nil for the last)
previous the previous record in the Q (nil for the first)

key (inherited from LookupKey) the record's start time in msec
value (inherited from Association) the record's event.

accessing

loop

loop: aBool

next
Answer the receiver's 'next'.

next: anObject
Set the receiver's instance variable 'next' to be anObject.

previous
Answer the receiver's 'previous'.

previous: anObject
Set the receiver's instance variable 'previous' to be anObject.

start

control

stop

MetaClass: ScheduleRecord class

Class: **PortModel**

Environment: Siren
Superclass: Model
Category: Music-Support
Instance variables: name status device in out
Class instance variables: blockSize devices in instance mutex out properties rate
useSingleton

initialize-release

initialize
no-op

mutex support

critical: aBlock
Execute the given block as a critical section

accessing

device

device: aValue

in

in: aValue

name

name: aValue

out

out: aValue

status

status: aValue

MetaClass: PortModel class

Instance variables: instance devices in out properties useSingleton rate
blockSize mutex

instance creation

concreteClass
Answer the class to use for ports

default
Answer a new instance, or the singleton

instance
Answer a new instance, or the singleton

new
Answer a new instance, or the singleton

release
Release the unique instance

class initialization

initialize
Set up the defaults for the class constants.

initializeData
Set up the defaults for the class constants.

class var accessing

blockSize

blockSize: value

devices

devices: anArr

in

in: value

instance: value

mutex

out

out: value

properties

properties: anArr

rate

rate: value

resetDevices
Flush the device list

sampleRate

sampleRate: value

Class: **EventScheduler**

Environment: Siren
Superclass: Model
Category: Music-Support
Instance variables: appointments timers threads running doWait startTime
Class variables: Schedule

An instance of EventScheduler is a simple driver for real-time message-passing among any classes that can return timed event association values. One uses Schedules by setting up clients which a master scheduler process continuously evaluates for their next timed activations. Clients can return appointments which are sorted into the scheduler's list of (time -> event) associations that is evaluated continuously when running. Changed messages are sent from the accessor messages to simplify EventScheduler browsers and inspectors.

Instance variables:
clients client objects
appointments the schedule
running running schedule?
doWait whether or not the scheduler should do real-time scheduling--set to false if there is a lower level of scheduling (e.g., primitive-level) or if output need not be real-time
startTime the clock time I was started at

Class variable:
Schedule shared instance accessed by the class message masterSchedule

See the class examples.

accessing

addClient: theApp

Add the argument, 'theApp', to the instance variable 'appointments'.

addClient: theApp at: start
Add the argument, 'theApp', to the instance variable 'appointments'.

addClient: theApp at: start loop: aBool
Add the argument, 'theApp', to the instance variable 'appointments'.

addClient: theApp in: start
Add the argument, 'theApp', to the instance variable 'appointments'.

addClient: theApp in: start loop: loopBool
Add the argument, 'theApp', to the instance variable 'appointments'.

addClient: theApp loop: aBool
Add the argument, 'theApp', to the instance variable 'appointments'.

addTimer: start
Add the argument, 'theApp', to the instance variable 'timers'.

addTimer: start interval: int
Add the argument, 'theApp', to the instance variable 'timers'.

clientNamed: aName
Answer a named client

clients
Get the instances clients

clock
Answer the instances clock (in usec)

dontWait
Specify that the scheduler should NOT do the waiting--i.e., there is a lower level of scheduling going on

isRunning
Answer the instance variable 'running'.

removeClient: aClient
Add the argument, 'theApp', to the instance variable 'timers'.

removeClientNamed: aClient
Add the argument, 'theApp', to the instance variable 'timers'.

removeTimer: aTimer
Add the argument, 'theApp', to the instance variable 'timers'.

removeTimerNamed: aTimer
Add the argument, 'theApp', to the instance variable 'timers'.

resetClock
Reset the instances clock (in usec)

timerNamed: aName
Answer a named timer

timers
Answer the instance's timers

wait
Specify that the scheduler should do the waiting--i.e., Smalltalk-level real-time.

updating

update: anAspect
Check if clients are waiting

running

callNextAppointment
Calls upon the next appointment to be made and then reschedules the next one

flush
Reset the appointment list.

interrupt
Stop a running scheduler.

processEvent: entry at: now
Handle an event or timer

run
Set up the first meetings and then run them all till no one wants a meeting anymore.

run: theBool
Accept the argument, 'theBool', as the instance variable 'running'.

initialize release

initialize
Set up the default Schedule

release
Clean up the schedule.

MetaClass: EventScheduler class

examples

scheduleExample
Play MIDI using the event scheduler.

scheduleExample2
Play MIDI using the event scheduler.

timerExample
Start some timers in the event scheduler.

class initialization

initialize
Set up the global Scheduler.

release
Clear away the global EventScheduler.

instance control

addClient: aT
Add a timer to the instance

addClient: aT at: theTime
Add a timer to the instance

addClient: aT in: theTime
Add a timer to the instance

addClient: aT in: theTime loop: loopBool
Add a timer to the instance

addTimer: aT
Add a timer to the instance

addTimer: aT interval: int
Add a timer to the instance

clients
Get the instance's clients

clock
Answer the instance's clock

flush
Reset all running schedules by brute force.

interrupt
Stop all running schedules by brute force.

isRunning
Ask the instance if it's on

resetClock
Answer the instance's clock

run
Turn the instance on

timers
Get the instance's timers

instance creation

default
EventScheduler instance

instance
EventScheduler instance

Class: **SirenUtility**

Environment: Siren
Superclass: ApplicationModel
Category: Music-Support
Class variables: DataDir ScoreDir SoundDir Verbosity

The class SirenUtility represents the package-level state of the Siren Framework. Its class methods answer a variety of Siren system variables such as default file directories.

Class Variables:
DataDir the default sound/score directory.
ScoreDir the score directory.
SoundDir the sound directory.

MetaClass: SirenUtility class

class initialization

initialize
Edit these to taste for your installation.

postLoad: aParcel
Load the missing pieces and put up a dialog after loading the parcel

class var accessing

dataDir
Answer the class's default data storage directory.

scoreDir
Answer the class's default score storage directory.

soundDir
Answer the class's default sound storage directory.

verbosity
Answer the class's default verbosity, 0/1/2.

verbosity: aNum
Set the class's default verbosity, 0/1/2.

utilities

categoryList
Answer the class categories for all of CSL

playSoundFile: nam
Play a sound file using UNIX shell and libsndfile's play program

writeSirenManual
Create the big book

writeSirenManualToHTML: folder
Create the doxygen-style manual in an HTML folder

example access

formNamed: key
Answer the given item from the global music constants.

musicConstants
Answer the global music constants (mostly icons).

file support

addDir: dir to: list
Try to locate the requested directory and, if found, add it to the given list

createS7: aFullPathName
Create an s7 folder and copy any files with the same name into it

findDir: dir
Try to locate the requested directory either locally or globally

findDir: dir tryHard: aggressive
Try to locate the requested directory either locally or globally

findFile: fil
Try to locate the requested file in the data directories, being smart about s7 files

findFile: fil in: theDir
Try to locate the requested file in the given directory, being smart about s7 files

findFiles: ext
Answer all of the files in the user's data folders with the given filename extension

findFiles: ext in: theDir into: coll
Add the files with the given name extension to the given collection

listS7: aName
List the contents of the requested s7 file

nextName: aName type: extension
Answer the next free name with the given extension in the given s7 folder

---

Class: **SirenSession**

Environment: Siren
Superclass: ApplicationModel
Category: Music-Support
Instance variables: defaultVoiceClass inChannels blockSize sampleRate outChannels interfaceParams transport eventList sound voice timer scheduleList soundPort midiPort loadedSounds loadedLists oscAddress verbosity clock startedAt
Class instance variables: instance useSingleton
Class variables: EventLists Sounds Voices

The class SirenSession represents the user session state of the Siren Framework. The class is also the implementor of the Siren configuration panel, see the interface window spec, and the instance methods.

Instance Variables (used by the pop-up utility window)
sampleRate - the sample rate
inChannels outChannels - # of snd I/O channels
oscIP oscPort - OSC IP & port
defaultVoiceClass - voice class to use
blockSize - snd IO block size
interfaceParams - snd IO properties
transport - value used by GUI
eventList - value used by GUI
sound - value used by GUI
voice - value used by GUI
timer - value used by GUI
scheduleList - value used by GUI
soundPort - value used by GUI

midiPort - value used by GUI

Class Variables:
EventLists Event lists
Sounds Sounds
Voices Voices

actions

allOff
Turn off anything that's playing

cleanUp

configureMIDI

freeLists

inspectSession

loadAll
SirenSession initialize. SirenSession loadDemoData

openSirenUtility

openTransport
Open a transport control

resetSound
Reset the parameters of the sound port

stopMIDI
Turn off MIDI

stopSound
Stop the sound player if it's active

aspects

blockSize

clock

defaultVoiceClass

eventList
This method was generated by UIDefiner. Any edits made here may be lost whenever methods are automatically defined. The initialization provided below may have been preempted by an initialize method.

inChannels

midiPort
This method was generated by UIDefiner. Any edits made here may be lost whenever methods are automatically defined. The initialization provided below may have been preempted by an initialize method.

oscAddress
Answer the formatted OSC IP address + port

outChannels

sampleRate

scheduleList
Answer the scheduler's list

sound

soundPort

timer
Answer the timer

updateClock
Answer the clock display

updateScheduleList
Answer the scheduler's list

updateTimer
Answer the timer display

verbosity
Answer the verbosity level 0, 1, or 2

verbosityLabel
Answer the verbosity level 0, 1, or 2

voice

interface opening

postOpenWith: aBuilder
This message is sent by the builder after it has opened a completed window.

initialize release

initialize
Set up the defaults values.

release

actions--change

chBlockSize

Respond to a selection in the block size menu

chInChans
Respond to a selection in the input channel menu

chListSelection
Respond to a selection in the main list menu

chOSCAddr
Respond to a selection in the OSC address field

chOutChans
Respond to a selection in the output channel menu

chSRate
Respond to a selection in the sampleRate menu

chVerbosity
Respond to a selection in the verbosity menu

chVoiceClass
Respond to a selection in the voice class menu

newEventList
Respond to a selection in the sound port menu

newMIDIPort
Respond to a selection in the sound port menu

newSound
Respond to a selection in the sound port menu

newSoundPort
Respond to a selection in the sound port menu

newTimer
Respond to a selection in the timer menu

newVoice
Respond to a selection in the sound port menu

accessing

selectedObject
Remove an item from the list

transport
it's me

resources

addItemsFrom: aDict to: aMenu path: aPath
Add items frmo the given dictionary to a hierarchical menu

listMenu
Answer the event list menu

soundMenu
Answer the hierarchical sound menu

timerMenu
Answer the timer list menu

voiceMenu
Tools.MenuEditor new openOnClass: self andSelector: #voiceMenu

actions--menu

deleteItem
Remove an item from the list

playItem
Remove an item from the list

spawnItem
Remove an item from the list

zeroTimer

actions--transport

forward
This stub method was generated by UIDefiner

play
Play the selection

record
This stub method was generated by UIDefiner

rewind
This stub method was generated by UIDefiner

stop
Stop the selection

actions--tests

testMIDI

testMIDIin

testMIDIout

testOSCout

testSndFileIn
This stub method was generated by UIDefiner

testSoundIn
This stub method was generated by UIDefiner

testSoundOut
Play a sound

updating

executeAutoSave

update: anAspectSymbol with: aParameter from: aSender
Catch an update

MetaClass: SirenSession class

Instance variables: instance useSingleton

class initialization

flushTempEventLists
Flush the class's temporary event list dictionary.

initialize
Edit these to taste for your installation.

initializeEventLists
Flush the class's event list dictionary.

initializeSounds
Empty the class's sound dictionary.

initializeVoices
SirenSession initializeVoices

update: anAspect with: arguments from: anObject
You can put code here to be done before or after a snapshot
(i.e., do you want to store events lists and sounds in the image or not)

class var accessing

atVoice: nam put: vox
Add to the class's voice dictionary.

eventList: aName put: anEventList
Store the give n event list in the shared dictionary; handle hierarchical names

eventListNamed: aName
Get the event list at the given (possibly hierarchical) name in the shared dictionary

eventLists
Answer the class's event list dictionary.

release
Make sure there's only ever one of me.

schedule
Answer the 'global' scheduler.

sound: aName put: aSound
Store the give n event list in the shared dictionary; handle hierarchical names

soundNamed: aName
Get the event list at the given (possibly hierarchical) name in the shared dictionary

sounds
Answer the class's sound dictionary.

voice: aName put: vox

Add to the class's voice dictionary.

voiceNamed: aName
Add to the class's voice dictionary.

voices
Answer the class's voice dictionary.

resources

blocksizeMenu
Tools.MenuEditor new openOnClass: self andSelector: #blocksizeMenu

chMenu
Tools.MenuEditor new openOnClass: self andSelector: #chMenu

clockMenu
Tools.MenuEditor new openOnClass: self andSelector: #clockMenu

forwardButton
UIMaskEditor new openOnClass: self andSelector: #forwardButton

midiPortMenu
Answer the midi port menu

playButton
UIMaskEditor new openOnClass: self andSelector: #playButton

rateMenu
Tools.MenuEditor new openOnClass: self andSelector: #rateMenu

recordButton
UIMaskEditor new openOnClass: self andSelector: #recordButton

rewindButton
UIMaskEditor new openOnClass: self andSelector: #rewindButton

scheduleMenu
Tools.MenuEditor new openOnClass: self andSelector: #scheduleMenu

soundPortMenu
Answer the sound port menu

stopButton
UIMaskEditor new openOnClass: self andSelector: #stopButton

timerMenu
Tools.MenuEditor new openOnClass: self andSelector: #timerMenu

verbosityMenu
Tools.MenuEditor new openOnClass: self andSelector: #verbosityMenu

data load/store

data: aName in: theDict put: anItem
Store the give n event list in the shared dictionary; handle hierarchical names

dataAt: aName in: theDict
Get the event list at the given (possibly hierarchical) name in the shared dictionary

loadDemoData
Load a few scores and sounds for demos

interface opening

open
Open the ApplicationModel's user interface.

openTransport
Open the ApplicationModel's user interface.

openUtility
Open the ApplicationModel's user interface.

interface specs

transportButtonSpec
Tools.UIPainter new openOnClass: self andSelector: #transportButtonSpec

transportWindowSpec
Tools.UIPainter new openOnClass: self andSelector: #transportWindowSpec

utilWindowSpec
Tools.UIPainter new openOnClass: self andSelector: #utilWindowSpec

instance creation

instance
Answer the 'global' session.

new
Make sure there's only ever one of me.

refresh
Recreate the instance

Class: **DeviceModel**

Environment: Siren
Superclass: Model
Category: Music-Support
Instance variables: name port stream debug in out rate

accessing

debug

debug: aValue

in

in: aValue

name

name: aValue

out

out: aValue

port

port: aValue

rate

rate: aValue

stream

stream: aValue

printing

printOn: aStream
Append to the argument aStream a sequence of characters that identifies the collection.

MetaClass: DeviceModel class

MusicIO-Voices

Class: **Voice**

Environment: Siren
Superclass: Model
Category: MusicIO-Voices
Instance variables: name instrument stream
Class variables: DefaultVoiceClass

Subclasses of Voice implement the mappings between symbolic event parameters (like timbre descriptions and pitch names) and concrete output event data.
They can be used for generating sound compiler notelists or (real-time output or file dumps of) MIDI data.

The instance variables are:
instrument my instrument
name my handle or name

stream *my (optional) I/O stream*

The global object Voices is a dictionary that note events can refer to with integer or symbolic keys.

accessing

instrument
return my instrument.

instrument: newValue
set my instrument.

stream
answer the stream of the receiver voice

stream: someStream
plug in a stream to the receiver voice

events

eventList
Answer an event list read from the receiver.

play: anEventList
Expand the list and play it now.

play: anEventOrList at: start
Expand the list and play it at the given time.

playEvent: anEvent
Play the argument now.

playEvent: anEvent at: aTime
make sure my subclasses implement this

readOnto: eventList
Read events from the receiver into the argument.

printing

printOn: aStream
Print the receivcer on the argument.

testing

isVoice
Answer true

initialize-release

close
Close the stream, devise, or whatever.

initialize
Set up a Voice.

initializeNamed: aName
Set up a named Voice.

MetaClass: Voice class

class var accessing

class initialization

initialize
set up the shared dictionary for caching voices

reset
Reset all sub-instances of Voice.

setDefault: className
Set the default class to use for new voices

instance creation

default
Answer a voice.

named: aName
Answer the shared dictionary's voice item at the given name (or the name itself if it's a voice)

on: someStream
Answer a new voice instance on the given stream.

Class: **SoundVoice**

Environment: Siren
Superclass: Siren.Voice
Category: MusicIO-Voices

SoundVoice is unfinished; it is intended for putting sound objects in scores.

events

playEvent: t1 at: t2

MetaClass: SoundVoice class

constant access

default

Class: **MIDIFileVoice**

Environment: Siren
Superclass: Siren.Voice

Category: MusicIO-Voices

Instance variables: fileType tracks ppq tempo

A MIDIFileVoice can read version 2 MIDI files.

accessing

defaultEventClass

tempo

tempo: tem

read/write

eventList
load a MIDI file into anEventList.

readMThd
Read the header chunk from MIDI binary file.

readMTrk
Read a track chunk from MIDI binary file. Answer its length.

readOnto: eventList
Read a MIDI file

readTrackOnto: anEventList
Read a track of MIDI events from my stream into anEventList

private

getVarLen
Combine 1 or more input bytes from my stream into a variable length item

initialize-release

close
Close the stream, device, or whatever.

MetaClass: MIDIFileVoice class

instance creation

newOn: fnam

examples

midiFileExample
MIDIFileVoice midiFileExample

Class: **NotelistVoice**

Environment: Siren
Superclass: Siren.Voice
Category: MusicIO-Voices
Instance variables: parameterMap lastTime header

Instances of the subclasses of NotelistVoice can read or write note list file streams in various formats for software sound synthesis packages such as cmusic, csound, or cmix.

Note that we are output-only at present.

Instance Variables:
parameterMap
the object's P:=map, used to print out or parse note commands.

accessing

addMap: anItem
Add the argument (a symbol or block), to the parameterMap of the receiver

header: aString
Set the file header string of the receiver.

parameterMap
Answer the parameterMap of the receiver.

parameterMap: anOrderedCollection
Set the parameterMap of the receiver.

events

dump: aStringOrValue
The case I handle is strings. Subclasses may override.

endLine
Put the proper command terminator on the receiver's stream.

mapProperty: item of: ass within: list
Write the given event association to the receiver's stream in the appropriate format.

play: anEL
Perform the argument.

play: anEL at: start
Perform the argument.

playAssociation: ass in: list
Write the given event association to the receiver's stream in the appropriate format.

space
Put the proper field separator on the receiver's stream.

writeFooter
Write a notelist file footer on the receiver's stream.

writeHeader
Write a notelist file header on the receiver's stream.

initialize-release

close
Close the receiver's output stream (if it's a file).

initialize
Initialize the receiver.

initializeNamed: aName onStream: theStream
Answer a new named NotelistVoice on the given stream.

initializeOnStream: theStream
Answer a new named NotelistVoice on the given stream.

MetaClass: NotelistVoice class

instance creation

defaultPMap
Answer the default parameterMap of the receiver class's instances.

newNamed: aName onFile: fName
Set up a NotelistVoice on the given output file

newNamed: aName onStream: aStream
Set up a NotelistVoice on the given output stream

onFileNamed: fName
Set up a NotelistVoice on the given output file

Class: **CmusicVoice**

Environment: Siren
Superclass: Siren.NotelistVoice
Category: MusicIO-Voices

A CmusicVoice can write an event list to a sc format file for use by Cmusic.

events

endLine

writeFooter

writeHeader
Write the cmusic score file header.

MetaClass: CmusicVoice class

examples

randomExampleToFileAndEdit

CmusicVoice randomExampleToFileAndEdit

randomExampleToFileNamed: fnam
Store the given score to a file

randomExampleToTranscript
CmusicVoice randomExampleToTranscript

instance creation

defaultPMap
Answer the default cmusic p-map

Class: **SuperColliderVoice**

Environment: Siren
Superclass: Siren.NotelistVoice
Category: MusicIO-Voices

A SuperColliderVoice can write an event list to a sc format file for use by SuperCollider.

events

endLine
Put the proper command terminator on the receiver's stream.

space
Put the proper field separator on the receiver's stream.

writeFooter
Write a notelist file footer on the receiver's stream.

writeHeader
Write a SuperCollider notelist file header on the receiver's stream.

MetaClass: SuperColliderVoice class

examples

randomExampleToFileAndEdit
SuperColliderVoice randomExampleToFileAndEdit

randomExampleToFileNamed: fnam
Create a random event list and store it on a file.

instance creation

defaultPMap
Answer up the parameter map for the default instance.

pMapForPanner
Answer up the parameter map for the default instance.

Class: **CsoundVoice**

Environment: Siren
Superclass: Siren.NotelistVoice
Category: MusicIO-Voices

A CsoundVoice can write an event list to a sco format file for use by Csound.

events

writeFooter

writeHeader

MetaClass: CsoundVoice class

examples

randomExampleToFileAndEdit
CsoundVoice randomExampleToFileAndEdit

randomExampleToFileNamed: fnam
Create a random event list and store it on a file.

instance creation

defaultPMap

Class: **CmixVoice**

Environment: Siren
Superclass: Siren.NotelistVoice
Category: MusicIO-Voices

A CmixVoice can write an event list to a MINC format file for use by cmix.

events

endLine

playEvent: evt at: start
Play the given event

space

writeHeader
Write out a CMix score file header.

MetaClass: CmixVoice class

examples

randomExampleToFileAndEdit
CmixVoice randomExampleToFileAndEdit

randomExampleToFileNamed: fnam
Create a random event list and store it on a file.

instance creation

default

defaultPMap
CmixVoice randomExampleToFileAndEdit

MusicIO-MIDI

Class: **MIDIPacket**

Environment: Siren
Superclass: Object
Category: MusicIO-MIDI
Instance variables: length time flags duration data
Imports: private MIDICommands.*

A MIDIPacket represents a simple MIDI datagram with a time-stamp and a data array.

Instance Variables
data The MIDI data bytes (may include a MIDI command and running status messages)
length The number of meaningful bytes in the data
time The msec timestamp when the packet was created or received.
flags Any 'flags' received from the MIDI driver.
duration my event's length

Pool dictionaries:
MIDICommands octet)> e.g., (#noteOn -> 16r90)
(It is initialized by class MIDIPort.)

See the class comment in MIDIPort for details.

accessing

ampl
Answer the receiver's MIDI velocity.

ampl: a
Set the receiver's MIDI velocity.

channel

Answer the receiver's MIDI channel.

command
Answer the receiver's MIDI command nibble.

data
Answer the receiver's 'data' byte array.

data: anObject
Set the receiver's instance variable 'data' to be anObject.

duration
Answer the receiver's duration.

duration: d
Set the receiver's duration.

key
Answer the receiver's MIDI key number.

key: k
Set the receiver's MIDI key number.

length
Answer the receiver's 'length'.

length: anObject
Set the receiver's instance variable 'length' to be anObject.

pitch
Answer the receiver's MIDI key number.

pitch: aValue
Set the receiver's MIDI key number.

second: aValue
Set the receiver's MIDI key number.

setPitch: p dur: d loudness: vol
Answer an instance of the class with the given parameters.

status: s
Set the receiver's MIDI command.

third: aValue
Set the receiver's MIDI key velocity.

time
Answer the receiver's 'time'.

time: anObject
Set the receiver's instance variable 'time' to be anObject.

vel
Answer the receiver's MIDI velocity.

vel: v
Set the receiver's MIDI velocity.

voice
Answer the receiver's MIDI channel.

voice: v
Set the receiver's MIDI channel.

printing

print: x on: aStream
Store a two-digit hexadecimal version of the 8-bit argument x on the stream.

printOn: aStream
Store a readable version of the receiver on the argument.

initialize release

initialize
Set the receiver's time stamp

MetaClass: MIDIPacket class

instance creation

bytesFor14BitValue: val
Answer a two-element array with the 14-bit values that correspond to the given value.

fromBytes: aByteArray
Answer a new instance of a MIDI packet instantiated from the given byte array.

fromInt: anInt at: aTime
Answer a new instance of a MIDI packet instantiated from the given 3-byte long value.

new
Answer a new instance.

new: size
Answer a new instance with a data array of the given size.

setPitch: p dur: d loudness: vol
Answer an instance of the class with the given parameters.

Class: **MIDIDump**

Environment: Siren
Superclass: Model
Category: MusicIO-MIDI
Instance variables: eventList live notesOn startedAt
Imports: private MIDICommands.*

An instance of MIDIDump is used as a dependent of the MIDIPort to demonstrate the use of dependency for MIDI input. See the class example.

Instance Variables:
verbose should I log events to the Transcript?
live should I match noteOn/Off commands on the fly?
eventList the event list I'm capturing input to
notesOn the current playing notes
startedAt the clock time I was started at

updating

update: aspect with: aParameter from: aSender
The model (port) changed; add the event to our event list.

initialize release

initialize
Setup the receiver's instance variables.

accessing

eventList

eventList: aValue

live: aValue

MetaClass: MIDIDump class

instance creation

new

examples

example
Set up a MIDI dump object as a dependent of the input port. Dump for 10 seconds, then turn off. The default update: method just dumps the MIDI packet into the transcript; customize this by writing your own update: method.

exampleEditor
Set up a MIDI dump object as a dependent of the input port. Capture it to an event list and update the view in real time BROKEN.

exampleList
Set up a MIDI dump object as a dependent of the input port. Capture it to an event list.

Class: **MIDIVoice**

Environment: Siren
Superclass: Siren.Voice
Category: MusicIO-MIDI
Instance variables: currentTime
Class instance variables: default
Class variables: Instance UseSingleton

A MidiVoice is used to play note events onto one or more output devices.

I store a MidiDevice (with a MIDIPort) in my instrument variable and play events onto it when asked to.

Instance variables:
instrument my instrument, a MidiDevice

initialize-release

close
Close the stream, device, or whatever.

initialize
Set up a MIDIVoice's time counter.

reset
Reset my time.

events

playEvent: anEvent at: aTime
Send the given Event to the instrument.

accessing

number
Answer my channel or icon number

MetaClass: MIDIVoice class

Instance variables: default

examples

randomExample
Play some random notes on a voice on a device on a port.

scaleExample
Play a scale on a voice on a device on a port to the Apple MIDI Mgr.

voiceInspect
MIDIVoice voiceInspect

instance creation

default
Answer the default MIDIVoice.

named: aName onDevice: aMidiDevice channel: aChannel
set up a MidiVoice on the given device and channel

new
Cache the latest instance.

on: aMidiDevice
Answer up a MidiVoice on the given device.

on: aMidiDevice channel: aChannel
set up a MidiVoice on the given device and channel

class initialization

defaultOutInterface
Answer the class's default intertface # (use external device list to get the right value).

flushDefault
Reset the class's default instance.

initialize
Reset the class's default instance.

Class: **MIDIPort**

Environment: Siren
Superclass: Siren.PortModel
Category: MusicIO-MIDI
Instance variables: inputData readProcess
Class variables: Streams
Imports: private MIDICommands.* private MIDIioctls.* private GeneralMIDIMap.* private GeneralMIDIDrums.*

An instance of a subclass of MIDIPort is used for the interface betweeen Siren and external MIDI drivers and devices. It implements both note-oriented (e.g., play: pitch at: aDelay dur: aDur amp: anAmp voice: voice), and data-oriented (e.g., put: data at: delay length: size) behaviors for MIDI I/O. There is an extensive test suite and demo in the class examples method and in the Siren outline view.

There is typically only one instance of MIDIPort. The messages new, default, and instance all answer the sole instance. MIDIPorts use dependency to signal input data, objects wishing to receive input should register themselves as dependents of a port. In the default Siren implementation, the scheduler is all in Smalltalk, and only the simplest MIDI driver is assumed.

Instance Variables:
readProcess The loop process to read input data.
inputData ByteArray)> The available data.
status #open or #closed
device my IO device's index

Class Variables:
Instance The sole instance, or nil.
Debug Debug mode prints all I/O to the Transcript.
UseSingleton whether to use a singleton instance (not necessary)
DefaultInputDevice the index in the driver of the default input device
DefaultOutputDevice see above

MIDI Commands Supported:
0x9x pp vv -- note-on (x=channel, pp=pitch, vv=velocity)
0x8x pp vv -- note-off (x=channel, pp=pitch, vv=velocity)
0xCx cc -- program-change (x=channel, pp=pitch)
0xEx ll hh -- program-change (x=channel, ll=low 7 bits, hh=high 7 bits)
0xBx cc dd -- control change (x=channel, cc=controller, dd=data)

accessing

input

Answer the receiver's Q of input data.

isActive
Answer whether the receiver is active.

resetInput
Reset the receiver's Q of input data.

control commands

allNotesOff
Turn all MIDI notes off using the channel message 123.

allNotesOffVerbose
Turn all MIDI notes off (the verbose way).

controlChange: chan controller: controller to: value
Send out a control-change command now.

pitchBend: chan to: value
Send out a pitch-bend command at the given time.

programChange: chan to: value
Send out a program-change command at the given time.

sysex: command
Send out a MIDI system exclusive data packet at the given time.

open/close

close
Close MIDI.

open
Open the MIDI driver -- start the lower-level driver up.

openInput
Open the MIDI driver -- start the lower-level driver up.

openInput: which
Open the MIDI driver -- start the lower-level driver up.

openOutput
Open the MIDI driver -- start the lower-level driver up.

openOutput: which
Open the MIDI driver -- start the lower-level driver up.

reset
Reset the port.

ioctl

eventsAvailable
Answer the number of events in the input Q.

hasBuffer
Answer whether the MIDI driver has a time-stamped output buffer.

hasClock
Answer whether the MIDI driver has its own clock.

hasControllerCache
Answer whether the MIDI driver supports a controller data buffer.

hasDurs
Answer whether the MIDI driver supports a 1-call note-on/off command.

readLoop
polling or waiting loop

startControllerCaching
Start caching MIDI controller in the driver.

startMIDIEcho
Start echoing MIDI input from the driver.

startMIDIInput
Start the polling loop (or semaphore waiter) MIDI input.

stopControllerCaching
Stop caching MIDI controller in the driver.

stopMIDIEcho
Stop echoing MIDI input from the driver.

stopMIDIInput
Stop signalling the read semaphore on MIDI input.

initialize release

initialize
Setup the receiver's instance variables.

release
Release--break and dependencies on the error value.

read/write

get: packet
Read the data from the receiver into the argument (a MIDIPacket). Answer the number of data bytes read.

nextEventInto: anEventAssociation
Record via the receiver into the argument.

nextMessage
Answer the first (length -> bytes) association from the input data collection.

play: aPitch dur: aDur amp: anAmp voice: aVoice
Play a note (on/off message pair) with the given parameters on the receiver.

play: streamID pitch: aPitch dur: aDur amp: anAmp voice: aVoice
Play a note (on/off message pair) with the given parameters on the receiver.

playOff: streamID pitch: aPitch amp: anAmp voice: aVoice
Play a note-on command with the given parameters on the receiver.

playOn: streamID pitch: aPitch amp: anAmp voice: aVoice
Play a note-on command with the given parameters on the receiver.

put: data length: size
Send the argument data to the receiver now.

readController: index
Read the given controller value.

readControllersFrom: lo to: hi into: array
Read a range of controllers

MetaClass: MIDIPort class

message tests

functionExample
Demonstrate control commands by playing a note and making a crescendo with the volume pedal.

testBend
Demonstrate pitch-bend by playing two notes and bending them.

testControlContinuous
Demonstrate control commands by playing a note and making a crescendo with the volume pedal.

testProgramChange
Demonstrate program change by setting up an organ instrument to play on.

testSysex
Demonstrate system exclusive commands by loading the Santur scale and playing a scale.

driver performance tests

testRandomPlayHighLevel: num dur: dur
Play 'num' random pitches (molto legato) spaced 'dur' msec apart.

testRandomPlayLowLevel: num dur: dur
Play 'num' random pitches spaced 'dur' msec apart.

testRollLowLevel: num dur: dur
Play a roll of 'num' notes spaced 'dur' msec apart.

output tests

testAllNotesOff
Try to open and close the MIDI port.

testANote
Open MIDI, play a note.

testARandomNote
Open MIDI, play a note.

testAScale
Open MIDI, play a fast scale.

testInspect
Inspect a MIDI port.

testMouseMIDI
Open MIDI, play notes based on the mouse position.

testNoteOnOff
Open MIDI, play a note, and close it.

testOpenClose
Try to open and close the MIDI port.

testOutput
Open MIDI, play some random notes, and close it.

input tests

dumpExample
Set up a MIDI dump object as a dependent of the input port. Dump for 10 seconds, then turn off. The default update: method just dumps the MIDI packet into the transcript; customize this by writing your own update: method.

testInput
Open MIDI, try to read something--dump it to the transcript.

testInputStop
Execute this to end the input test

controller tests

testControllerCaching
Set up uncached controller reading--make a loop that reads and prints controller 48 twice a second (until you press the shift button).

testControllerCaching2
Set up uncached controller reading--read controllers 48-52 as an array and print it; stop on mouse press.

testControllerCachingFrom: lo to: hi
Set up uncached controller reading--read controllers from lo to hi (inclusive) as an array and print it; stop on press.

testControllerRecording
Set up uncached controller reading--make a loop that reads and prints controller 48 40 times a second for 5 seconds.

utilities

allNotesOff
MIDIPort allNotesOff

cleanUp
Close down and clean up all MIDI, sound IO, event lists, etc.; then garbage collect.

showInput
Open MIDI, wait to read something, then dump it to the transcript.

general MIDI patches

setAllInstrumentsTo: iname
Set instruments 0-15 to the General MIDI name iname

setEnsemble: orch
Down-load a general MIDI patch for the given ensemble (a collection of [chan -> name] associations).

setEnsembleInOrder: orch
Down-load a general MIDI patch for the given ensemble (a collection of symbolic keys into the General MIDI voice map) mapping the first element to MIDI channel 1, etc.

setupDefaultGeneralMIDI
Down-load a general MIDI patch for a 16-voice percussion ensemble.

setupOrgan
Down-load a general MIDI patch for a 4-voice organ.

setupTunedPercussion
Down-load a general MIDI patch for a 16-voice percussion ensemble.

setupWindOrchestra
Down-load a general MIDI patch for a 16-voice wind ensemble.

examples

examples
Select and execute the following for usage examples.

scaleExampleFrom: lo to: hi in: dur
Answer array of (start dur pitch amp)

instance creation

concreteClass
Answer the class to use for MIDI

class initialization

initialize
Set up the dictionaries of commands, ioctl primitive selectors and arguments, and general MIDI maps.

initializeMIDITables
MIDIPort initializeMIDITables

class var accessing

Class: **PortMIDIPort**

Environment: Siren
Superclass: Siren.MIDIPort
Category: MusicIO-MIDI
Instance variables: driver
Imports: private MIDICommands.* private MIDIioctls.* private GeneralMIDIMap.* private GeneralMIDIDrums.*

An instance of PortMIDIPort is the interface to the external driver that talks to the PortMIDI interface.

Instance Variables:
driver my I/O interface driver

Shared Variables:
Devices unised
Streams the map between Siren devices and interface ports

open/close

close
Close MIDI.

close: stream
Close MIDI.

open
Open the MIDI driver -- start the lower-level driver up.

openInput: dev
Open the MIDI driver -- start the lower-level driver up.

openOutput: dev
Open the MIDI driver -- start the lower-level driver up.

terminate
Close MIDI.

ioctl

readLoop
The MIDI driver input process loop.

startControllerCaching
Start caching MIDI controller in the driver.

stopControllerCaching
Stop caching MIDI controller in the driver.

initialize release

initialize
Setup the receiver's instance variables.

release
Release--break and dependencies on the error value.

read/write

playOff: aPitch amp: anAmp voice: aVoice
Play the argument on the receiver--no duration-->no note-off.

playOff: streamID pitch: pitch amp: amp voice: voice
Play the argument on the receiver.

playOn: aPitch amp: anAmp voice: aVoice
Play the argument on the receiver--no duration-->no note-off.

playOn: streamID pitch: pitch amp: amp voice: voice
Play the argument on the receiver--no duration-->no note-off.

put: streamID data: data length: size
Send the argument (a ByteArray for historical reasons) to the receiver now.

put: data length: size
Send the argument (a ByteArray for historical reasons) to the receiver now.

readControllersFrom: lo to: hi into: array

Read a range of controllers

accessing

isActive
Answer whether the receiver is active.

MetaClass: PortMIDIPort class

examples

testANote
Open MIDI, play a note.

class initialization

initialize
Set up the dictionaries of commands, ioctl primitive selectors and arguments, and general MIDI maps.

Class: **MIDIDevice**

Environment: Siren
Superclass: Siren.DeviceModel
Category: MusicIO-MIDI
Class variables: MStream

MIDIDevice is a subclass of Model whose instances are used to model MIDI input/output hardware devices.
The abstract class MIDIDevice implements the generic MIDI note on/off type events.
Subclasses of MIDIDevice exist for specific models of devices and implement the device-specific (system exclusive) commands.
MIDIDevice are passed Events and channel numbers by their voices and generate commands as ByteArrays that they pass to MIDIPorts.
Several devices may share one port (if there are several MIDI-capable devices on one cable), and one voice may point to several device/channel pairs on one or more devices.

Instance variables:
port the MIDIPort I use
stream logging stream
debug verbosity flag for Transcript dumping

Class variable:
MStream the stream I dump by bytes on if my port is nil (good for debugging new subclasses)

Standard MIDI Commands:
Note On = 9n kk vv
Note Off = 8n kk vv
Key Pressure = An kk vv
Pitch Wheel = En ll ll
After Touch Channel Pressure = Dn vv
Control Change = Bn cc vv
Program change = Cn pp

Relevant Constants for description of MIDI commands:
n = 4-bit channel number

kk = 7-bit key number
vv = 7-bit key velocity
ll = 7-bit low-order value
hh = 7-bit high-order value
cc = 7-bit control number
pp = 7-bit program number

accessing

port
Answer the receiver's I/O port.

port: aPort
Set the receiver's I/O port.

stream
Answer the receiver's MIDI stream ID.

stream: aNumber
Set the receiver's MIDI stream ID.

initialization

checkPort
Ensure that the receiver's output port is initialized.

close
Release the receiver's hold on the port.

initialize
Set the receiver up on the default output port.

initialize: thePort
Set the receiver up on the given output port.

play: anAss on: channelNumber
play the given event on my port as a default MIDI noteOn/noteOff pair

note events

play: anEvent at: aTime
Play the given event on my port as a default MIDI noteOn/noteOff pair.

parameter mapping

mapAmplitude: aVal
Map a numerical or symbolic amplitude to a MIDI-compatible volume number (key velocity)

mapDuration: aVal
Map a numerical or symbolic duration value to a MIDI-compatible duration in msec.

mapPitch: aVal
Map a numerical or symbolic pitch to a MIDI-compatible note number.

mapVoice: aVal
Map a numerical or symbolic pitch to a MIDI-compatible note number.

MetaClass: MIDIDevice class

examples

playOnDevice
Play a note out MIDI from the device level.

scheduleExample
Play MIDI using the event scheduler.

instance creation

new
Open a generic MidiDevice on the default Port

on: port
Open a generic MidiDevice on the given Port

class initialization

initialize
Set up the one class variable

Class: **MIDIFB01**

Environment: Siren
Superclass: Siren.MIDIDevice
Category: MusicIO-MIDI

MidiFB01 is a type of MidiDevice for the system exclusive commands of the Yamaha FB-01 synthesiser. It implements the special commands available here (like note commands with fractional pitch and given duration).

The standard FB01 note command is like:
2n k f v d1 d2 = cmd:channel, noteNum, frac, vel, durLow, durHigh

note events

play: anEvent on: aChannel
send FB01-specific code for note with fraction and duration
2n k f v d1 d2 = cmd:channel, noteNum, frac, vel, durLow, durHigh

MetaClass: MIDIFB01 class

Class: **MIDIPF70**

Environment: Siren
Superclass: Siren.MIDIDevice
Category: MusicIO-MIDI

MidiPF70 is a type of MidiDevice for the system exclusive commands of the Yamaha PF-70 electric piano. It implements the special commands available here (like note commands with fractional pitch and given duration).

The standard PF70 note command is like:
2n k f v d1 d2 = cmd:channel, noteNum, frac, vel, durLow, durHigh

MetaClass: MIDIPF70 class

MusicIO-OSC

Class: **OSCVoice**

Environment: Siren
Superclass: Siren.NotelistVoice
Category: MusicIO-OSC
Instance variables: port
Indexed variables: objects

An OSCVoice can play events out to an OSC server using a predefined parameter mapping block to generate the OSC message.

Instance Variables:
port my output port

events

oscMessageFrom: event
Answer a message for the given event by passing it to my parameter-mapper block

play: anEL
Expand the list and play it now.

playAssociation: ass in: list
Play the association...

playEvent: event at: start
Expand the list and play it at the given time.

waitTill: time

utilities

send: command args: args
Send an OSC command with the given arguments

accessing

port
Return the OSCPort

port: aPort
Set the OSCPort

MetaClass: OSCVoice class

examples

fmExample1
Play a few random notes on the CSL FM instrument

fmExample2
Play molto legato notes on 4 CSL FM instruments and loop until interrupted

fmExample3
Play a long CSL FM note and apply some real-time control functions to it

fmExample4
*Play an FM bell using CSL; fork a block that plays a note and waits a bit;
stop the scheduler to stop the block.*

functionExample
Play a function out to OSC

midiScaleExample
OSCVoice midiScaleExample

sndExample1
Play random notes on the CSL sound file instruments

sndExample2
Play random notes on the CSL sound file instruments; loop until interrupted

instance creation

default
Answer an OSC voice for use with OSC-to-MIDI by default.

localhost
Answer an instance on the local host

map: mapSelector
Answer a default instance that uses the given parameter map name

onPort: pt
Answer an instance on the given port

scHost
Answer the default voice for SC over OSC

parameter maps

defaultPMap
Answer the default parameterMap for use with OSC.

pMapForCSLFM
Answer the default parameterMap for use with the CSL FM example instrument.

pMapForCSLSimpleFM
Answer the default parameterMap for use with the CSL FM example instrument.

pMapForCSLSnd
Answer the default parameterMap for use with the CSL FM example instrument.

pMapForMIDI
Answer the default parameterMap for use with OSC-to-MIDI.
In this class, we create and return a OSCMessage with data taken from the given event.

pMapForMIDItoFM
Answer the default parameterMap for use with the CSL FM example instrument.

pMapWithBundle
Answer the default parameterMap for use with OSC-to-MIDI. In this class, we create and return a TypedOSCMessage with data taken from the given event. This allows us to have other versions that create OSC bundles.

Class: **AbstractOSCPacket**

Environment: Siren
Superclass: Object
Category: MusicIO-OSC
Instance variables: oscBytes

AbstractOSCPacket is the parent of the concrete OSC packet classes.

Subclasses must implement the following messages:
osc>>toOSCBytes:

Instance Variables:
oscBytes my packet contents

osc

toOSCBytes: converter

accessing

oscBytes

oscSize

initialize-release

init

MetaClass: AbstractOSCPacket class

Class: **OSCMessage**

Environment: Siren

Superclass: Siren.AbstractOSCPacket

Category: MusicIO-OSC

Instance variables: address arguments

OSCMessages are concrete old-format (untyped) OSC messages

Instance Variables:
address my OSC cmomand address
arguments the arguments

osc

addressToOSCBytes: convertor
Convert the address to the OSC format

argumentsToOSCBytes: convertor
Add the type tags, then the arguments

toOSCBytes: converter
Convert myself to OSC format

typesToOSC: converter

printing

printOn: aStream

initialize-release

initAddress: t1 arguments: t2

MetaClass: OSCMessage class

instance-creation

for: addr

for: addr with: args
Answer a TOM with the given address and arguments

examples

example1
OSCMessage example1

example2
OSCMessage example2

example3
OSCMessage example3

Class: **OSCBundle**

Environment: Siren
Superclass: Siren.AbstractOSCPacket
Category: MusicIO-OSC
Instance variables: messages time
Class variables: MillisecScale MillisecToNTP SecsInAYear

OSCBundle represents a bundle of OSC messages

Instance Variables:
messages my contents
time my action time

Shared Variables:
MillisecScale msec/year (?)
MillisecToNTP msec scale for NTP
SecsInAYear sec/year

osc

timeTagToOSCBytes: aConvertor
Store a time thingy into the given convertor

toOSCBytes: aConvertor
^self

initialize-release

init: aCollection

init: aCollection time: aTime
Initialize the receiver.

MetaClass: OSCBundle class

instance creation

with: aCollection

with: aCollection at: aTimestamp

support

asNTPMilliseconds: aTimestamp
Convert aTimestamp to the OSC representation of the same

asSecondsSince1900: aTimestamp
Convert aTimestamp to seconds since 1900

timestampToOSC: aTimestamp
Convert aTimestamp to the OSC representation of the same

class initialization

initialize
Initialize the values of the shared variables.

examples

example1
Simple example

example2

Class: **OSCPort**

Environment: Siren
Superclass: Siren.PortModel
Category: MusicIO-OSC
Instance variables: socket address
Class variables: DefaultIP DefaultPort

An OSCPort represents a connection to an OSC client.

Instance Variables:
socket how am I connected?
address where do I go?

Shared Variables:
DefaultIP where's my default server?
DefaultPort where's my default server?

accessing

portNumber: newPortNum
Reset the receiver's UDP socket port number

actions

send: aMessage
Send something

initialize-release

init: t1
Set up

MetaClass: OSCPort class

instance-creation

cslHost
Answer the default port for CSL

default
Answer the default instance

localhost
Answer an OSC port on the local host

scHost
Answer the default OSC server reference

to: t1
Open a port on the given device

to: ip port: port
^an OSCPort with the given attributes

toHostName: t1
Answer an instance on the given host

toHostName: aHostName portNumber: aPort
^an OSCPort
Convenience method

toLocalhostPortNumber: aPort
^an OSCPort
Convenience method

class initialization

initialize
Setup the defaults

defaults

defaultCSLOSCPort
The default for CSL

defaultIP
Answer the class var default

defaultIP: value
Answer the class var default

defaultPort
Answer the class var default

defaultPort: value
Set the class var default

defaultSCOSCPort
Answer the default port for OSC

examples

demo
OSCPort demo

demo2
OSCPort demo2

sendFreq
PSCPort sendFreq

sendStart
OSCPort sendstop

sendStop
Send a stop message

Class: **TypedOSCMessage**

Environment: Siren
Superclass: Siren.OSCMessage
Category: MusicIO-OSC

A TypedOSCMessage represents the new form of (typed) OSC messages

osc

argumentsToOSCBytes: convertor
Convert the args and type string to OSC format

typesToOSC: converter
Convert the arguments to an OSC type string.

MetaClass: TypedOSCMessage class

examples

changeFreq
TypedOSCMessage changeFreq

scDecreaseVolume
TypedOSCMessage scDecreaseVolume

scIncreaseVolume
TypedOSCMessage scIncreaseVolume

scRun
TypedOSCMessage scRun

scStop
TypedOSCMessage scStop

start
TypedOSCMessage start

Class: **OSCByteConvertor**

Environment: Siren
Superclass: Object
Category: MusicIO-OSC

Instance variables: packetData

Instances of OSCByteConvertor translate between different OSC formats.

Instance Variables:
packetData my contents

stream

contents

next: t1 put: t2

nextPut: t1

nextPutAll: t1

nextPutType: char
Transcript show: (String with: char); space.

position

initialize-release

init

MetaClass: OSCByteConvertor class

instance creation

new

MusicIO-Sound

Class: **SoundFile**

Environment: Siren
Superclass: Siren.AbstractEvent
Category: MusicIO-Sound
Instance variables: name mode fileFormat sampleFormat rate channels size
index position sound
Indexed variables: objects

Class variables: FileFormats Interface SampleFormats

Instances of SoundFile are used for reading and writing sound objects to/from files using the libSndFile API.
They handle formatting, headers and I/O.

Instance variables:
sound the sound's samples
rate the sound's sampling rate
channels the number of channels
fileName the file's name
file the file's stream
position the current position
headerSize the size of the soundfile header
properties the property list dictionary of the receiver
format the format--currently only #linear16Bit is handled
size the size in samples

accessing

channels
Answer the instance variable 'channels'.

channels: theChannels
Accept the argument, 'theChannels', as the new instance variable 'channels'.

fileFormat

fileFormat: aValue

format

mode: theMode
Set the receiver's mode to #read or #write

name
Answer the instance variable 'name'.

name: theFileName
Accept the argument, 'theFileName', as the new instance variable 'name'.

rate
Return the instance variable 'rate'.

rate: aRate
Set the instance variable 'rate'.

sampleFormat

sampleFormat: aValue

samples
Return the instance variable 'samples'.

size
Answer the number of sample frames in the file.

size: aNumber
Set the number of sample frames in the file.

sizeInSamples
Answer the number of sample frames in the file.

sound
Answer the instance variable 'sound'.

sound: theSound
Accept the argument, 'theSound', as the new instance variable 'sound'.

read/write

readSamples
Read samples from the file into the sample buffer.

readSamples: sndClass
Read samples from the file into the sample buffer.

saveSound: snd
Write samples from the given sound to the receiver file.

printing

printOn: aStream
Format and print the receiver on the argument.

initialize-release

close
Close the receiver's file.

initialize
Set up the instance variables of a default sound file.

open
Open the named file and read the header

openForReading
Open the named file and read the header

MetaClass: SoundFile class

instance creation

named: nameString
Open the given file (EBICSF, SPARC, or NeXT soundfile format)

openFileNamed: nameString
Open the given file (EBICSF, SPARC, AIFF, or NeXT soundfile format)

readFileNamed: nameString
Open the given file (EBICSF, SPARC, or NeXT soundfile format)

readFileNamed: nameString answer: theClass
Open the given file (EBICSF, SPARC, or NeXT soundfile format)

class initialization

initialize
Set up the file format and sample format dictionaries to interface with libsndfile.

Class: **SoundPort**

Environment: Siren
Superclass: Siren.PortModel
Category: MusicIO-Sound

SoundPort instances are interfaces to real-time sound I/O streams.

Concrete subclasses add primitive interfaces to special devices such as audio ports or coprocessors.

open/close

close
Close the receiver sound port

open
Open the receiver sound port

start
Start the receiver sound port

stop
Stop the receiver sound port

initialize/release

initialize
Answer an initialized version of the receiver.

release
Terminate and release the receiver.

play/record

play: aSound
Play the argument on the receiver over the DACs.

play: aSound from: start to: stop
Play the argument on the receiver over the DACs.

record: aSound
Record into the argument via the receiver.

MetaClass: SoundPort class

class initialization

initializeData
Set up the defaults for the class constants.

instance creation

concreteClass
Answer the appropriate subclass.

default
Answer a default instance of the appropriate subclass.

defaultOrNil
Answer the default instance of the appropriate subclass if it's set up.

Class: **PortAudioPort**

Environment: Siren
Superclass: Siren.SoundPort
Category: MusicIO-Sound
Instance variables: rate format outChannels interface isOpen isRunning
bufferSize

A PortAudioPort is an interface to the external PortAudio library. It is simple, supports playback only, and does not use callbacks from C into Smalltalk. For a fancier version that supports input as well , see SmartAudioPort.

Instance Variables:
device the PortAudio device flag for my device
rate the sample rate in Hz
format sample format as in #lin16Bit
outChannels # of out channels
interface my low-level interface
isOpen am I open?
isRunning am I running?
bufferSize size of the preallocated IO buffers

Shared Variables:
Devices my total # of I/O channels and default I/O sample rates

open/close

close
Close the receiver sound port

open
Open the receiver sound port

start
Start the receiver sound port

stop
Stop the receiver sound port

terminate
Shut down the receiver.

accessing

format

format: aValue

outChannels

outChannels: aValue

rate

rate: aValue

initialize/release

initialize
Answer an initialized version of the receiver.

play/record

play: aSound
Play the argument on the receiver over the DACs.

play: aSound from: start to: stop
Play the argument on the receiver over the DACs.

MetaClass: PortAudioPort class

initialize/release

stop
Stop and close the running instance

examples

playSweep
Play a swept sine wave using the simple output-only port audio port.

playSweepLong
*Play a *long* swept sine wave using the simple output-only port audio port.*

Class: **SmartAudioPort**

Environment: Siren
Superclass: Siren.PortAudioPort
Category: MusicIO-Sound
Instance variables: inChannels callbackBlock useProcess ioProcess ioSemaphore inClient outClients inBuffer outBuffer counter

SmartPortAudioPort is a more advanced IO port for PortAudio; it uses a Smalltalk process that sleeps on a semaphore that is signalled by the PortAudio callback. this allows one to synthesize sound interactively or to record into Siren sounds. The methods nextInBuffer and nextInBuffer are used to copy buffers or call synthesis clients.

Instance Variables:
inChannels # of input channels
ioSemaphore semaphore sent from the low-level driver
inClient sample index> the sound being recorded into
outClients Integer)> my playing sounds and their sample indices
ioProcess my IO thread
inBuffer sample buffer handed to PortAudio
outBuffer sample buffer handed to PortAudio

play/record

callback
Handle a callback from PortAudio; this is sent an an external callback

doIO
Handle a callback from PortAudio; this is sent an an external callback through the class

nextInBuffer
Get the next input buffer for the clients from the interface.

nextOutBuffer
Get the next output buffer from the clients and sum it into my buffer for playing.

play: aSound
Play the argument on the receiver over the DACs.

record: aSound
Record into the argument via the receiver.

start: duration
Start the receiver's IO loop for the given duration.

accessing

inChannels

inChannels: aValue

outClients

setInputDevice
Set the port to use the default input device

open/close

close
Close the receiver sound port

open
Open the receiver sound port

initialize/release

initialize
Answer an initialized version of the receiver.

MetaClass: SmartAudioPort class

examples

testPlay
Test playing a sound through the call-back interface

testRecord
SmartAudioPort testRecord

callbacks

callback
Forward the callback to an instance

instance creation

new
Answer a new instance, or the singleton

Class: **SoundEvent**

Environment: Siren
Superclass: Siren.ActionEvent
Category: MusicIO-Sound
Instance variables: sound
Indexed variables: objects

accessing

sound

sound: aSnd

MetaClass: SoundEvent class

examples

example
Create and play an event list that plays the same file a few times.

exampleLoops
Create and play an event list that loops 2 samples, with one of the loops starting with a rest.

instance creation

sound: theSnd
Create and answer a new sound event for the given sound

MusicIO-External

Class: **SirenExternalInterface**

Environment: Siren
Superclass: ExternalInterface
Category: MusicIO-External
Imports: private Siren.SirenExternalInterfaceDictionary.*
Attributes:
#(#(#includeFiles #()) #(#includeDirectories #()) #(#libraryFiles #()) #(#libraryDirectories #()) #(#beVirtual false) #(#optimizationLevel #full))

MetaClass: SirenExternalInterface class

class var accessing

initialization

unload
SirenExternalInterface unload

Class: **PortAudioInterface**

Environment: Siren
Superclass: Siren.SirenExternalInterface
Category: MusicIO-External
Instance variables: cbProcess ioSemaphore
Class variables: PA_Constants
Imports: private Siren.PortAudioInterfaceDictionary.*
Attributes:
#(#(#includeFiles #('portaudio_lite.h')) #(#includeDirectories #('Siren7.5/DLLCC')) #(#libraryFiles #('portaudio_lite.dylib' 'libportaudio.dylib')) #(#libraryDirectories #('/usr/local/lib')) #(#beVirtual false) #(#optimizationLevel #full))

types

Callback

CallbackFcn

OEoop

procedures

pa_close

pa_dev_in: which

pa_dev_out: which

pa_dev_rate: which

pa_get_name: which

pa_get_name: which with: resultString

pa_initialize

pa_in_device

pa_in_rate

pa_num_devices

pa_num_in

pa_num_out

pa_open_p: device with: out_ch with: format with: rate with: blockSize

pa_open_s: device with: in_ch with: out_ch with: format with: rate with: blockSize with: cbBlock with: in_buffer with: out_buffer

pa_out_device

pa_out_rate

pa_play: out_buffer with: numChannels with: numFrames

pa_play: out_buffer with: numChannels with: numFrames with: swap

pa_start

pa_stop

pa_terminate

accessing

cbProcess

cbProcess: aValue

ioSemaphore

ioSemaphore: aValue

MetaClass: PortAudioInterface class

class initialization

const: flag

initialize
Set up the class constants dictionary (PortAudio 19)

examples

example0
Demonstrate using the PortAudioInterface with the semaphore-signalling interface

example1
Demonstrate using the PortAudioInterface with the semaphore-signalling interface

example2
Demonstrate using the PortAudioInterface with the non-semaphore-signalling interface

---

Class: **LibSndFileInterface**

Environment: Siren
Superclass: Siren.SirenExternalInterface
Category: MusicIO-External

Class variables: SF_Constants

Imports: private Siren.LibSndFileInterfaceDictionary.*

Attributes:

#(#(#includeFiles #('sndfile_lite.h')) #(#includeDirectories #('Siren7.5/DLLCC')) #(#libraryFiles #('libsndfile.dylib' 'sndfile_lite.dylib')) #(#libraryDirectories #('/usr/local/lib')) #(#beVirtual false) #(#optimizationLevel #full))

procedures

lsf_close: which

lsf_create: name with: mode with: format with: rate with: channels

lsf_get_channels: which

lsf_get_format: which

lsf_get_frames: which

lsf_get_rate: which

lsf_open: name with: mode

lsf_read_Fsamples: which with: where with: count

lsf_read_lsamples: which with: where with: count

lsf_seek: which with: pos with: key

lsf_write_Fsamples: which with: where with: count

lsf_write_lsamples: which with: where with: count

MetaClass: LibSndFileInterface class

class initialization

const: flag

initialize
Set up the class constants dictionary

examples

example1: filename
Demonstrate using the LibSndFileInterface; this will dump some messages to the transcript

---

Class: **FFTWInterface**

Environment: Siren
Superclass: Siren.SirenExternalInterface
Category: MusicIO-External
Imports: Siren.FFTWInterfaceDictionary.*
Attributes:
#(#(#includeFiles #('fftw_lite.h')) #(#includeDirectories #('Siren7.5/DLLCC')) #(#libraryFiles #('fftw_lite.dylib' 'libfftw3f.a')) #(#libraryDirectories #('/usr/local/lib')) #(#beVirtual false) #(#optimizationLevel #full))

procedures

fftw_float_to_short: data

fftw_forward_transform

fftw_initialize: size with: samples with: spectrum

fftw_mag_spectrum: data

fftw_phas_spectrum: data

fftw_reverse_transform

fftw_short_to_float: data

MetaClass: FFTWInterface class

examples

example
Demonstrate using the FFTWInterface by taking the FFT of a sawtooth wave

---

Class: **PortMidiInterface**

Environment: Siren
Superclass: Siren.SirenExternalInterface
Category: MusicIO-External
Imports: private Siren.PortMidiInterfaceDictionary.*
Attributes:
#(#(#includeFiles #('portmidi_lite.h')) #(#includeDirectories #('Siren7.5/DLLCC')) #(#libraryFiles #('portmidi_lite.dylib' 'portmidi.dylib' 'CoreMIDI')) #(#libraryDirectories #('/usr/local/lib' '/System/Library/Frameworks/CoreMIDI.framework/Versions/Curre #(#beVirtual false) #(#optimizationLevel #full))

types

OEoop

procedures

pm_close: which

pm_count_devices

pm_default_input_device

pm_default_output_device

pm_dev_dir: which

pm_get: which

pm_get_name: which

pm_has_error: which

pm_initialize

pm_open: device with: direction

pm_poll: which

pm_read: which

pm_read_controllers: which with: fromController with: toController with: data

pm_start_controller_cacheing

pm_stop_controller_cacheing

pm_terminate

pm_test

pm_write_data2: which with: d1 with: d2

pm_write_data3: which with: d1 with: d2 with: d3

pm_write_long: which with: msg with: length

pm_write_short: which with: msg

MetaClass: PortMidiInterface class

examples

example1
Demonstrate using the PortMidiInterface

testMIDI
Demonstrate using the PortMidiInterface to call the test note function in the driver

testMIDI2
Demonstrate using the PortMidiInterface to play a note on/off cmd pair

---

MusicUI-DisplayLists

Class: **DisplayList**

Environment: Siren
Superclass: DependentComposite
Category: MusicUI-DisplayLists
Instance variables: offset

Instances of DisplayList are used for representing composed structured graphics. They can be nested (see the class examples) and can display themselves and their components.

Instance Variables:

offset *the object's offset distance relative to its container*

accessing

color

flatten
Answer a copy of the receiver with its hierarchy flattened.

itemsFromX: x1 toX: x2
Answer the list of items whose offsets are within the given X range.

itemsIntersecting: rect
Answer the list of items whose boundingBoxes intersect the given rectangle.
Clip them to the box's border if necessary.

itemsWithin: rect
Answer the list of items whose boundingBoxes are entirely within the given rectangle.

itemsWithin: dist of: point
Answer the list of items whose boundingBoxes are entirely within the given rectangle.

itemWithin: dist of: point
Answer the list of items whose boundingBoxes are entirely within the given rectangle.

modelsFromX: x1 toX: x2
Answer the list of model-space-items whose offsets are within the given X range.

nodeAt: aPath
Answer the element described by the collection of items in the argument.

nodeFor: aModel
Answer the element whose model is the argument.

offset
Answer the receiver's offset.

offset: aPoint
Set the receiver's offset.

wrapperClass
Raise an error.

transforming

scaledBy: aPoint
Scale the receiver's offset by the argument.

translateBy: aPoint
Translate the receiver's offset by the argument.

printing

printCompleteOn: aStream depth: depth
Descend the hierarchy printing on the stream.

printCompletePostScriptOn: aStream depth: depth
Descend the hierarchy printing on the stream.

printOn: aStream

Print the receiver on the argument using the recursive method.

printPostScriptOn: aStream
Print the receiver on the argument using the recursive method.

testing

hasItems
Answer whether or not the receiver has items or components.

isEmpty
Amswer whether the receiver has any items.

bounds accessing

computePreferredBounds
Compute the receiver's preferredBounds

extent
Answer the extent of the receiver.

enumerating

do: aBlock
Iterate the argument block over the receiver's components.

displaying

display
Open a DisplayListView on the receiver.

displayNonCached
Open a DisplayListView on the receiver.

displayPostScriptOn: aPostscriptContext
Display each of the receiver's components.

adding-removing

add: aVisualComponent
Add the argument to the receiver.

add: aVisualComponent at: aPoint
Add aVisualComponent to the receiver's components with its offset set to aPoint.

add: anItem atPath: aPath
Add the first argument at the node described by the second.

addAll: aCollection
Add the argument to the receiver.

addComponent: aVisualComponent
Add the argument to the receiver.

addWrapper: aVisualWrapper
Raise an error.

As yet unclassified

displayOn: aGraphicsContext
Display each of the receiver's components.

MetaClass: DisplayList class

examples

exampleHierarchical
Create and answer a large display list with a lines, strings, and visuals.

gridExample
Create and answer a large display list with lines and strings.

gridExampleX: x byY: y
Create and answer a large display list with lines and strings.

polylineExample
Answer a display list with randomly-placed random-color polylines over the given extent.

polylineExampleHuge
Answer a display list with randomly-placed random-color polylines over the given (very large) extent.

polylinesX: x byY: y
Answer a display list with randomly-placed random-color polylines over the given extent.

polylinesX: x byY: y items: num
Answer a display list with randomly-placed random-color polylines over the given extent.

postScriptExample
Create and answer a large display list with a lines, strings, and visuals.

randomExample
Create and answer a large display list with a lines, strings, and visuals.

rectangleExample
Answer a display list with randomly-placed random-color rectangles over the given extent.

rectanglesX: x byY: y
Answer a display list with randomly-placed random-color rectangles over the given extent.

stringExample
Open a display list view with randomly-placed random-color strings over the given extent.

stringsX: x byY: y
Answer a display list with randomly-placed random-color strings over the given extent.

visualsX: x byY: y
Answer a display list with randomly-placed random-color visuals over the given extent.

Class: **DisplayItem**

Environment: Siren
Superclass: DependentPart
Category: MusicUI-DisplayLists
Instance variables: offset color

Instances of the subclasses of the abstract class DisplayItem are used as the items in display lists. They

can generally display themselves on graphics contexts.
These are done this way because I don't think wrappers are right for use in display lists.

Instance Variables:
offset the object's offset relative to its container (display list)
color the object's display color (or nil)

The subclasses add special display-related state and behavior such as an extent point and display method, or a visual item such as a string or image to display.
They generally implement displayOn: aGraphicsContext and bounds accessing methods.

This implementation is MODE 1.1, STEIM, Amsterdam, May/June 1990; updated at the Lagoon in Palo Alto, July, 1991-May, 1992.

transforming

asVisualComponent
Answer an encapsulated version of the receiver which understands VisualComponent protocol.

scaleBy: aPoint
Translate the receiver's offset by the argument.

scaledBy: aPoint
Translate the receiver's offset by the argument.

translateBy: aPoint
Translate the receiver's offset by the argument.

printing

printCompleteOn: aStream depth: depth
Print the receiver in the given Stream.

printOn: aStream
Print the receiver in the given Stream.

accessing

color
Answer the receiver's display color.

color: aVal
Set the receiver's display color to the argument.

extent
Answer the extent of the receiver (dummy in this class).

extent: aPoint
Ignored

offset
Answer the receiver's offset.

offset: aPoint
Set the receiver's offset to the argument.

testing

isDisplayItem
Answer whether the receiver is a kind of DisplayItem

initialize-release

initialize
Initialize the instance variables of the receiver.

displaying

displayOn: aGraphicsContext
Display the receiver on the argument.

displayPostScriptOn: aPostScriptContext
Display the receiver on the argument in PostScript.

copying

copy
Answer a shallow copy of the receiver.

bounds accessing

computePreferredBounds
Answer the receiver's bounds--hack.

MetaClass: DisplayItem class

instance creation

model: aM offset: aPt
Answer an instance of DisplayItem with the given instance variables.

offset: aPt
Answer an instance of DisplayItem with the given instance variables.

Class: **DisplayLine**

Environment: Siren
Superclass: Siren.DisplayItem
Category: MusicUI-DisplayLists
Instance variables: width extent

Instances of DisplayLine are used for visual lines in display lists.

Instance Variables:
width the object's line width
extent the object's visual extent

accessing

corner: aPoint
Set the corner of the receiver (offset + extent).

extent
Answer the extent of the receiver.

extent: aPoint
Set the extent of the receiver.

width: aVal

printing

printOn: aStream
Print the receiver in the given Stream.

displaying

displayOn: aGraphicsContext
Display a line between startPoint and endPoint.

displayPostScriptOn: aPostscriptContext
Display the receiver on the argument as a PostScript item.

bounds accessing

bounds
Answer the receiver's bounds.

MetaClass: DisplayLine class

instance creation

from: oPoint to: endPoint
Answer an initialized instance.

offset: oPoint extent: ePoint
Answer an initialized instance.

Class: **DisplayRectangle**

Environment: Siren
Superclass: Siren.DisplayLine
Category: MusicUI-DisplayLists
Instance variables: fill stroke

Instances of DisplayRectangle can be used to display bordered or filled rectangles in display lists. This is a subclass of DisplayLine for reasons of practicality. a "purist" (e.g., David Liebs), would create an intermediate abstract class (e.g., BoundedDisplayItem) for both DisplayLine and DisplayRectangle.

Instanve Variables:
fill whether or not to fill the receiver on display (no by default)

accessing

fill: aBoolean
Set the receiver's filling Boolean.

stroke: aNum
Set the receiver's stroke line thickness.

displaying

displayOn: aGraphicsContext
Display a line between startPoint and endPoint.

displayPostScriptOn: aPostscriptContext
Display the receiver on the argument as a PostScript item.

MetaClass: DisplayRectangle class

examples

rectangleExample
Open a display list view with randomly-placed random-color rectangles over the given extent.

Class: **DisplayPolyline**

Environment: Siren
Superclass: Siren.DisplayRectangle
Category: MusicUI-DisplayLists
Instance variables: vertices

A DisplayPolyline is a poly-line-segment display object

Instance Variables:
vertices my point array

accessing

bounds
Answer the receiver's bounds.

extent
Answer the receiver's extent.

vertices: pointArray
Set the receiver's vertices.

transforming

scaledBy: aPoint
Scale all the receiver's points by the argument; answer a copy.

displaying

displayOn: aGraphicsContext
Stroke the receiver's edges on the supplied GraphicsContext.

MetaClass: DisplayPolyline class

instance creation

offset: offPt vertices: arrayOfPoints
Answer a new DisplayPolyline with the arguments as its vertices.

vertices: arrayOfPoints
Answer a new DisplayPolyline with the arguments as its vertices.

examples

polylineExample
Open a display list view with randomly-placed random-color polylines over the given extent.

Class: **DisplayString**

Environment: Siren
Superclass: Siren.DisplayItem
Category: MusicUI-DisplayLists
Instance variables: string font

Instances of DisplayString are used for visual text items in display lists.

Instance Variables:
string the object's string/text
font the object's display font

accessing

bounds
Answer a rectangle that circumscribes the receiver.

extent
Answer a rectangle that circumscribes the receiver.

font
Answer the receiver's font.

font: aFont
Set the receiver's font.

string
Answer the receiver's string.

string: aString
Set the receiver's string.

printing

printOn: aStream

comment stating purpose of message

displaying

displayOn: aGraphicsContext
display the receiver on the argument.

displayPostScriptOn: aPostscriptContext
Display the receiver on the argument as a PostScript item.

initialize-release

initialize
Initialize the instance variables of the receiver.

MetaClass: DisplayString class

examples

stringExample
Open a display list view with randomly-placed random-color strings over the given extent.

instance creation

new
Answer a new initialized instance.

string: aString
Answer a new instance with the argument as its string.

string: aString offset: oPoint
Answer a new instance with the arguments as its string and offset point.

Class: **DisplayVisual**

Environment: Siren
Superclass: Siren.DisplayItem
Category: MusicUI-DisplayLists
Instance variables: visual
Imports: MusicConstants

Instances of DisplayVisual are used for displaying arbitrary visual objects (e.g., cached images) in display lists.

Instance Variables:
visual the object's visual object, typically an image

accessing

bounds
Answer the visual's bounds translated by the receiver's translation.

visual

Answer the receiver's visual.

visual: aVisual
Set the receiver's visual.

printing

printOn: aStream
Print the receiver in the given Stream.

storeOn: aStream
Print the receiver in the given Stream.

displaying

displayOn: aGraphicsContext
Display the receiver's visual on the given graphics context.

displayPostScriptOn: aPostscriptContext
Display the receiver on the argument as a PostScript item.

MetaClass: DisplayVisual class

examples

convertMusicConstants
Convert all the forms to opaque forms

displayMusicConstants
Draw a nice table of the hierarchical image dictionary.

storeMusicConstants
Write out all the forms to opaque forms

visualExample
Open a display list view with randomly-placed random-color visuals over the given extent.

instance creation

model: aM visual: aV offset: aPt
Answer an instance of DisplayItem with the given instance variables.

visual: aV
Answer an instance of DisplayItem with the given instance variables.

visual: aV offset: aPt
Answer an instance of DisplayItem with the given instance variables.

visual: aV origin: aPt
Answer an instance of DisplayItem with the given instance variables.

MusicUI-DisplayListViews

Class: **DisplayListController**

Environment: Siren
Superclass: ControllerWithMenu
Category: MusicUI-DisplayListViews
Instance variables: selection selOffset

Instances of DisplayListController (or its subclasses) are used by DisplayListViews (or subclass instances) for menu management and selection.

By default, a simple YellowButton menu allows for group/inspect/redraw/zoom.
The RedButton is used for display item selection (by pointing or boxing-in).
LeftShift extends the selection, and dragging a selection moves it.

Instance Variables:
selection the object's selected items

menu messages

again
Repeat the last operation

copyItem
Copy the selection.

cutItem
Cut the selection.

doInspect
Inspect the view (if leftShiftDown) or the model.

flattenItems
Flatten the hierarchy of the selection.

groupItems
Group the selection.

inspectDisplayList
Inspect the display list.

inspectEditor
Inspect the display list editor.

inspectModel
Inspect the model.

optionsDialog
Run the options dialog box.

pasteFromBuffer
Paste the selection.

pasteItem
Paste the selection.

redraw
Redraw the view and clear the selection.

redrawView
redraw the display list.

undo
Un-do the last operation

ungroupItems
Un-group the selection.

zoom
Prompt the user for a scale point, zoom the view's scale, and redraw.

zoomInPoint
Zoom the view's scale, and redraw.

zoomOutPoint
Zoom the view's scale, and redraw.

zoomTo1
Zoom the view's scale, and redraw.

control defaults

blueButtonActivity
Drag scroll amplified by 2@2 with the blue button.

redButtonActivity
Hit-detect items from the display list or drag a selection.

redButtonPressedAt: pt
Handle the red mouse click

yellowButtonActivity0
Drag scroll amplified by 2@2 with the blue button.

selecting

selectAtPoint: aPoint
Hit-detect the model for the given pt.

selectFromPoint: aPoint toPoint: anotherPoint
Hit-detect the model for the given range.

selectRangeWhile: durationBlock
Hit-detect items from the display list or drag a selection as long as the block is true.

accessing

selection
Answer the receiver's 'selection'.

selection: anObject
Set the receiver's instance variable 'selection' to be anObject.

selectionTracker

events

dragSelection: anEvent
Dragging is event oriented.

mouseMovedEvent: anEvent

Mouse dragging

redButtonPressedEvent: event

redButtonReleasedEvent: event

selectEvent: anEvent

initialize-release

initializeMenu
Set up the receiver's YellowButtonMenu

MetaClass: DisplayListController class

resources

defaultEditMenu
MenuEditor new openOnClass: self andSelector: #defaultEditMenu

---

Class: **DisplayListTracker**

Environment: Siren
Superclass: SelectionTracker
Category: MusicUI-DisplayListViews
Instance variables: inside

A DisplayListTracker is a controller for mouse tracking in DLViews

Instance Variables:
inside is the mouse in my view?

private

trackSelectionFor: aPoint

events

mouseMovedEvent: aMouseMovedEvent

redButtonPressedEvent: aMouseButtonEvent

MetaClass: DisplayListTracker class

Class: **DisplayListSubcanvas**

Environment: Siren

Superclass: ApplicationModel

Category: MusicUI-DisplayListViews

Instance variables: displayList componentPart displayListView scroller vZoom hZoom

A DisplayListSubcanvas is used to plug a display list view into an aplpication.

Instance Variables:
displayList comment
componentPart my app. pane
displayListView my view (does all the work)
scroller my scroller (container)
vZoom vertical zoom factor
hZoom horiz. zoom factor

actions

nextPage
This stub method was generated by UIDefiner

prevPage
This stub method was generated by UIDefiner

zoomChanged
Sent when the users moves the zoom sliders.

zoomTo1
Sent when the users presses the zoom-to-1 button.

zoomToLast
This stub method was generated by UIDefiner

aspects

createDisplayListView
Create and answer the receiver's display list view

hZoom
This method was generated by UIDefiner. Any edits made here may be lost whenever methods are automatically defined. The initialization provided below may have been preempted by an initialize method.

setComponent: aComponent
Set the receiver's component part

setDisplayList: aDisplayList
Set the receiver's display list

vZoom
This method was generated by UIDefiner. Any edits made here may be lost whenever methods are automatically defined. The initialization provided below may have been preempted by an initialize method.

MetaClass: DisplayListSubcanvas class

interface specs

windowSpec
Answer the user's favorite window layout

windowSpec4Square
UIPainter new openOnClass: self andSelector: #windowSpec4Square

windowSpecLeftBottom
Tools.UIPainter new openOnClass: self andSelector: #windowSpecLeftBottom

instance creation

onList: aDisplayList

openOnList: aDisplayList
DisplayListSubcanvas openOnList: (DisplayList rectanglesX: 6000 byY: 6000)

Class: **DisplayListEditor**

Environment: Siren
Superclass: Siren.EditorModel
Category: MusicUI-DisplayListViews
Instance variables: selection list

An instance of DisplayListEditor is used as the intermediary model for viewing lists.
It handles selection and operation on the list.
It adds itself as a dependent of the list and echoes its change messages (i.e., the update: message says self changed).

Instance Variable:
list the 'subject' list model
selection the current selection or nil

accessing

displayList
Answer the receiver's list.

list
Answer the receiver's list.

list: aList
Set the receiver's list, removing the dependency to the former list if necessary.

moveSelectionTo: newPoint
Move the editor's selection to the new point in the list.

select: sel
Set the receiver's selection.

selection
Answer the receiver's selection.

initialize-release

release
Remove the dependency to the list.

editing

doDisplay
inspect button

doEdit
edit button

doFile
file i/o button

doInspect
Inspect the editor (if shiftDown) or the model.

doZoom
zoom button

updating

update: anAspect with: anArg from: aModel
Echo the model's changes, assuming a view is a dependent of the receiver.

MetaClass: DisplayListEditor class

instance creation

on: model

Class: **DisplayListView**

Environment: Siren
Superclass: AutoScrollingView
Category: MusicUI-DisplayListViews
Instance variables: displayList pixmap background page zoom grid extent inset backgroundColor foregroundColor redrawn cache
Class variables: MusicConstants

DisplayListViews can display and scroll structured graphics display lists generated by various models or layout managers.
Using their page offset, they can scroll over very large display lists.
They can have colored or gridded background pixmaps and can scroll/page on demand; they display their lists on their graphics contexts.

Subclasses generally override the displayOn: or displayOnImage methods, and add initialization or transformation methods.

Instance Variables
displayList the view's display list
pixmap the view's cached display pixel map (optional)
background the view's background form (e.g., gridding)
zoom the display list's zoom-in factor or nil
pageOffset the offset in "pages" used for very large display lists
backgroundColor graphics background color
foregroundColor graphics display color
redrawn set to nil to re-draw cache
cache should I cache a Pixmap of redisply n the fly?

See the class examples for numerous ways of using display list views.

The subclasses add knowledge of smart display list generation, background pixmap generation (e.g., gridding), display of item or x/y-range selection, x- or y-scaling or step/grid, property->color mapping, "clef forms" or other special pixmaps, multiple-model viewing, etc.

accessing

background: anObject
Set the receiver's 'background' to be anObject.

backgroundColor
Answer the receiver's backgroundColor or the default.

bounds: aRectangle
Set the receiver's bounds (and page offset).

cache: aBoolean
Set the receiver's 'cache' to be aBoolean.

displayList
Answer the receiver's display list.

displayList: anObject
Set the receiver's 'displayList' to be anObject.

foregroundColor
Answer the receiver's foregroundColor or the default.

inset: aPoint
Set the receiver's inset to aPoint (pixels).

list
Answer the receiver's display list.

pageOffset
Answer the receiver's 'pageOffset'.

pageOffset: anObject
Set the receiver's 'pageOffset' to be anObject.

pixmap: aPMorNil
Set (or destroy) the receiver's cached pixmap.

preferredBounds
Answer the displayList's bounds.

zoom: aPoint
Set the receiver's 'zoom' to be aPoint.

private

scrollableExtent
Answer the extent of the receiver's display object bounding box.

setModel: aModel

visibleExtent
Answer the extent of the receiver's clipping box.

controller access

defaultControllerClass

transformPoint: aPoint
zoom, scroll, and page the given point, used for hit-detection.

initialize-release

component
Answer a DLView for use as a component.

initialize
Initialize the receiver's instance variables--the default is not to cache.

initializeCache
Initialize the receiver's instance variables for cacheing the pixmap during display.

initializeWithGrid: gPoint color: gColor
Initialize the receiver's instance variables--the default is to cache.

release
Remove the receiver as a dependent of its model.

displaying

displayOn: aGraphicsContext

displaySelectionOn: aGC
Display the model's selection using some form of highlighting.

invalidate

MetaClass: DisplayListView class

instance creation

componentOnList: dlist
Answer a DLView for use as a component.

componentOnList: dlist withGrid: gPoint inColor: gColor
Answer a DLView for use as a component.

on: anEditor

open4SquareOn: aDisplayList
Open the receiver on the given display list.

openOn: aDisplayList
Open the receiver on the given display list.

constant access

backgroundColor
Answer the default background color for display list views.

constants

constants: aDictionary

defaultCacheExtent
Answer the default extent of the receiver's instances' cache.

foregroundColor
Answer the default foreground color for display list views.

inset
Answer the default display inset.

class initialization

initialize
Init the class

examples

classListExample
DisplayListView classListExample

classTreeExample
DisplayListView classTreeExample

colorClassListExample
DisplayListView colorClassListExample

exampleHierarchical
DisplayListView exampleHierarchical

MusicUI-Layout

Class: **StructureAccessor**

Environment: Siren
Superclass: Object
Category: MusicUI-Layout
Instance variables: subject itemGenerator

Instances of StructureAccessor and its subclasses are 'protocol convertors' which provide predictable interfaces (e.g., tree-speak), for diverse data structures.
The basic display item generating protocol is:

```
    (aStructureAccessor itemFor: aModelNode).

InstanceVariableNames:
    subject
```