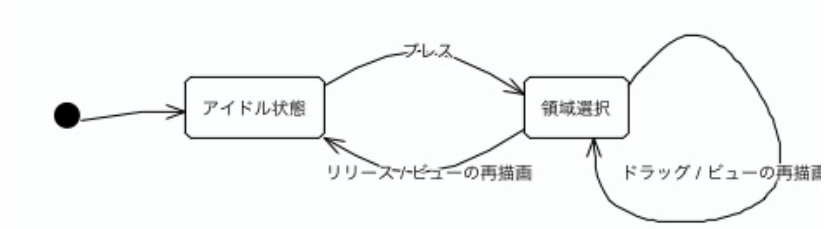
USB メモリの中身
Fourier.tar
SSK_20120111_094507.st
vw78jun792mac.zip
vw78jun792win.zip

先月休会だったので、思い出しつつ…

File Browser から SSK_20120111_094507.st を File in

Fourier1dModel の example1 を使いつつ復習
右下の PowerSpectrum だけ、 専用の MVC を作った状態(他の物は PaneMVC そのまま)
それの実装は SSK, SSK-Fourier, FourierPaneController, Instance, events, redButtonPressedEvent: event

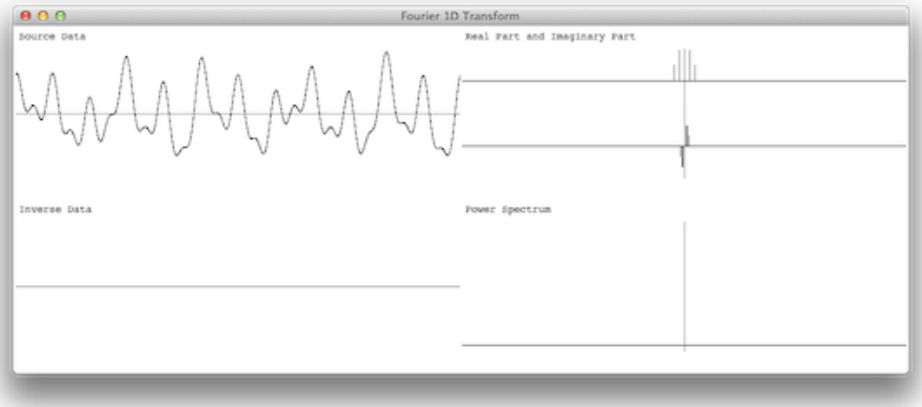


こんな、感じの実装をした。

redButtonReleasedEvent (や、 Drag)もあるけれども、インスタンス変数に状態(どの座標でクリックをしたか)を持たなければならないので、かっこ悪いので、 redButtonPressedEvent 内で
[self sensor redButtonPressed] whileTrue:
として、全て実装してしまっている。

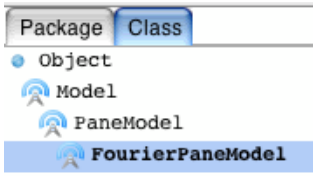
yield することで、このイベントだけで CPU を食い尽くさないようにしている事が大切である事も思い出しておくように。

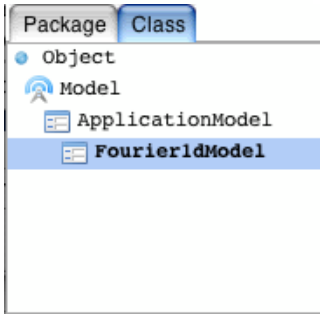
すーっかり忘れてしまっていますが、



右下の Power Spectrum の部分をドラッグして選択すると、その部分だけフーリエ変換する仕様のものを作成していた。
ということで、 Power Spectrum をドラッグしたときのイベントを誰に伝えるか？
今のコードでは誰にも伝えていない。

FourierPaneModel と Fourier1dModel でスーパークラスが違う。





どうにかして、 FourierPaneModel をアプリケーションに伝える
元のデータを全て知っているのはアプリケーションだけなので、どうにか伝えないと必要なデータを取り出すことが出来ない。

SSK, SSK-Fourier, FourierPaneModel の Definition の instanceVariableNames に application を追加(既に追加されている)。

application を使うための下準備
SSK, SSK-Fourier, FourierPaneModel, Instance, initialize-release, initialize
initialize

```
super initialize.
application := nil.
^self
```

を追加(コンストラクタに相当)

SSK, SSK-Fourier, FourierPaneModel, Instance, accessing, application
application

```
^application
```

を追加(ゲッターに相当)

SSK, SSK-Fourier, FourierPaneModel, Instance, accessing, application:
application: anApplication

```
application := anApplication
```

を追加(セッターに相当)

SSK, SSK-Fourier, Fourier1dModel, Instance, aspects, powerSpectrumView
powerSpectrumView

```
| aModel aView |
aModel := (SSK.FourierPaneModel on: self)
                picture: nil;
                yourself.
aModel label: 'Power Spectrum'.
aView := SSK.FourierPaneView model: aModel.
aView alignmentSymbol: #center.
^aView
```

赤文字部分を追加する。
元々は何も与えずにインスタンスを作っていたが、これで Fourier1dModel が FourierPaneModel に含まれるようになる。

SSK, SSK-Fourier, FourierPaneModel, Class, instance creation, on:
on: anApplication

```
^(self new)
	application: anApplication;
	yourself
```

SSK, SSK-Fourier, FourierPaneController, Instance, events, redButonPressedEvent
redButtonPressedEvent: event

```
| startingPoint previousPoint currentPoint graphicsContext |
startingPoint := self sensor cursorPoint.
previousPoint := startingPoint.
graphicsContext := self view graphicsContext.
[self sensor redButtonPressed] whileTrue:
		[currentPoint := self sensor cursorPoint.
		currentPoint = previousPoint
			ifFalse:
				[| aRectangle |
				self view displayOn: graphicsContext.
				aRectangle := Rectangle vertex: startingPoint vertex: currentPoint.
				graphicsContext
					paint: ColorValue red;
					displayRectangularBorder: aRectangle.
				self model application
					ifNotNil:
						[:anApplication |
						Transcript
							cr;
							show: Time now printString].
				previousPoint := currentPoint].
		Processor yield].
self view displayOn: graphicsContext.
^nil
```

application がちゃんとやってきているか、いつも通り、 Transcript に書いてみて確認。
ここまで変更を加えて、 example1 を実行

確認できたら

SSK, SSK-Fourier, FourierPaneController, Instance, events, redButonPressedEvent
redButtonPressedEvent: event

```
| aBoolean startingPoint previousPoint currentPoint graphicsContext |
aBoolean := self sensor shiftDown.
startingPoint := self sensor cursorPoint.
previousPoint := startingPoint.
graphicsContext := self view graphicsContext.
[self sensor redButtonPressed] whileTrue:
		[currentPoint := self sensor cursorPoint.
		currentPoint = previousPoint
			ifFalse:
				[| aRectangle |
				self view displayOn: graphicsContext.
				aRectangle := Rectangle vertex: startingPoint vertex: currentPoint.
				graphicsContext
					paint: ColorValue red;
					displayRectangularBorder: aRectangle.
				self model application
					ifNotNil: [:anApplication | anApplication framedArea: aRectangle shiftDown: aBoolean].
				previousPoint := currentPoint].
		Processor yield].
self view displayOn: graphicsContext.
^nil
```

accept して framedArea: shiftDown: がないと言われるので proceed しておく。
(Shift キーを押すことで、選択部分を非選択状態にする仕様なので、 Shift キーが押されているか伝える)

先ほどないと言われたので

SSK, SSK-Fourier, FourierPane1dModel, Instance, actions, framedArea: shiftDown:
framedArea: aRectangle shiftDown: aBoolean

```
Transcript
	cr;
	show: aRectangle printString
```

を追加。

SSK, SSK-Fourier, FourierPaneController, Instance, events, redButonPressedEvent
redButtonPressedEvent: event

```
	| aBoolean startingPoint previousPoint currentPoint graphicsContext |
	aBoolean := self sensor shiftDown.
	startingPoint := self sensor cursorPoint.
	previousPoint := startingPoint.
	graphicsContext := self view graphicsContext.
	[self sensor redButtonPressed] whileTrue:
		[currentPoint := self sensor cursorPoint.
		currentPoint = previousPoint
			ifFalse:
				[| aRectangle |
				self view displayOn: graphicsContext.
				aRectangle := Rectangle vertex: startingPoint vertex: currentPoint.
				graphicsContext
					paint: ColorValue red;
					displayRectangularBorder: aRectangle.
				self model application
					ifNotNil:
						[:anApplication |
						aRectangle := (self view wherePoint: aRectangle origin)
									corner: (self view wherePoint: aRectangle corner).
						anApplication framedArea: aRectangle shiftDown: aBoolean].
				previousPoint := currentPoint].
		Processor yield].
	self view displayOn: graphicsContext.
	^nil
```

と変更。
これで、 Power Spectrum の座標系がとれるようになった。

SSK, SSK-Fourier, FourierPaneController, Instance, private, convertToIntervals:
は先月やる予定だったのだが、休会したので実装しておいたとのこと。
一番左が 512 で真ん中の左が 1023 で、真ん中の右が 0 で一番右が 511 になっているので
左右がひっくり返っているのでそれを元に戻す物

それが保存されている変数が SSK, SSK-Fourier, FourierPane1dModel の Definition の interactiveTable
それの作り方は SSK, SSK-Fourier, FourierPane1dModel, InstanceVariable, interactiveTable, computeAux を見ると書いてある。
これは画像を計算して書き直してくる。

色々と確認したところでドラッグで選択された場所のデータを持ってくる
SSK, SSK-Fourier, FourierPane1dModel, Instance, actions, framedArea: shiftDown:
framedArea: aRectangle shiftDown: aBoolean

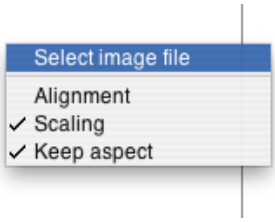
```
	(self convertToIntervals: aRectangle) do:
		[:interval |
		interval do:
			[:n |
			| index |
			index := n + 1.
			aBoolean
				ifTrue:
					[(interactiveTable at: #realPart) at: index put: 0.0d.
					(interactiveTable at: #imaginaryPart) at: index put: 0.0d.
					(interactiveTable at: #powerSpectrum) at: index put: 0.0d]
				ifFalse:
					[(interactiveTable at: #realPart) at: index put: (realPart at: index).
					(interactiveTable at: #imaginaryPart) at: index put: (realPart at: index).
					(interactiveTable at: #powerSpectrum) at: index put: (realPart at: index)]]].
	self computeAux
```

を追加。
緑色の部分がデータを持ってきているのだが、
SSK, SSK-Fourier, FourierPane1dModel, InstanceVariable, interactiveTable, flushAll
と同様の書き方である。
flushAll はデータを初期化するときに使う物である。

これで、選択したところのみを変換できる。

また、 Shift キーを押しながら選択すると、その部分を変換していない状態になる。

Select image file を Model が、他の3つを VC が持っている



これを出すために

PaneController, Instance, events, yellowButtonPressedEvent:
yellowButtonPressedEvent: event

```
	| aMenu |
	aMenu := self model yellowButtonMenu.
	aMenu addPart: self yellowButtonMenu.
	aMenu startUp evaluate.
	^nil
```

この様な実装になっており、緑色の部分で Model のものと Controller のものを複合している

FourierPaneController, Instance, menu accessing, yellowButtonPressedEvent
yellowButtonPressedEvent

```
	^nil
```

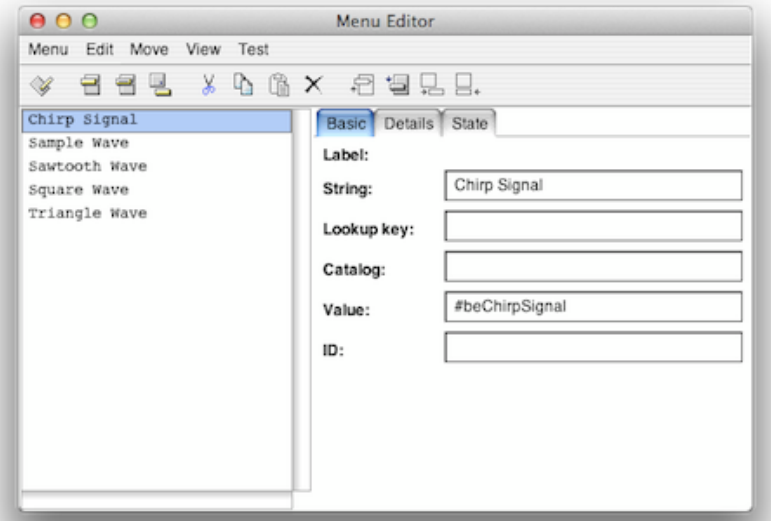
を追加し、初期化してしまう。

FourierPaneMocel, Class, resources, yellowButtonMenu
yellowButtonMenu

```
	"Tools.MenuEditor new openOnClass: self andSelector: #yellowButtonMenu"

	<resource: #menu>
	^#(#{UI.Menu} #(
			#(#{UI.MenuItem}
				#rawLabel: 'Select image file'
				#value: #selectImageFile ) ) #(1 ) nil ) decodeAsLiteralArray
```

を source に作り accept すると Visual タブが追加される。(中身は次の作業で書き換わるが…)
Visual から Edit
Chirp Signal, Sample Wave, Sqwtooth Wave, Square Wave, Triangle Wave を下記のように追加。
他の物も Value 部分は #be でスペースがない状態



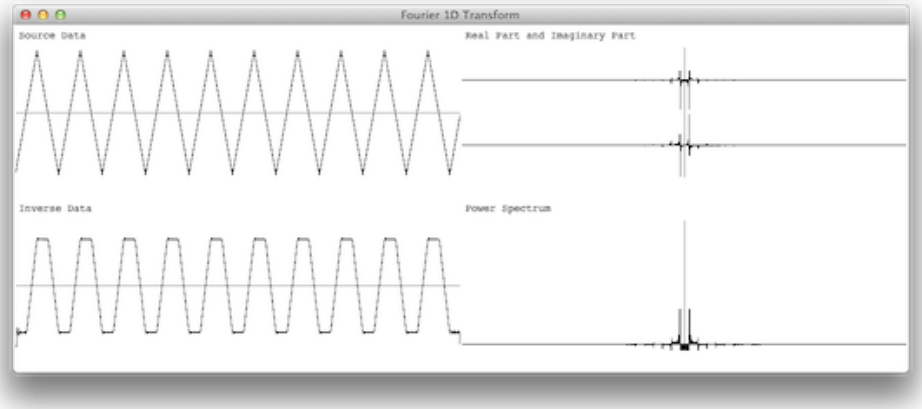
最後に Install

FourierPaneModel, Instance, menu accessing 以下に
beChirpSignal, beSampleWave, beSawtoothWave, beSquareWave, beTriangleWave
を何も実装しないまま追加して、とりあえず、アプリケーションが止まらないようにする。

次にちゃんと実装。
FourierPaneModel, Instance, menu accessing, beChirpSignal
beChirpSignal

```
self application
	ifNotNil:
		[:anApplication |
		anApplication
			setSourceData: anApplication defaultFourierTransformClass dataChirpSignal;
			computeAll]
```

上と同様の物で青色部分をそれぞれの名前に変えた物を実装していく。



これで、一通り完成。

ただ、先ほどの実装は酷いので、書き直し。
Fourier1dModel, Instance, actions, beChirpSignal
beChirpSignal

```
self
	setSourceData: self defaultFourierTransformClass dataChirpSignal;
```

computeAll

を追加。
やはり、これも、青色部分をそれぞれの名前に対応して変更する。

先ほどの物は
FourierPaneModel, Instance, menu accessing, beChirpSignal
beChirpSignal

```
self application ifNotNil: [:anApplication | anApplication beChirpSignal]
```

と変更。

とりあえず、今回はこれで、保存。

Fourier1dModel, Instance, actions, beChirpSignal
beChirpSignal

```
self application ifNotNil: [:anApplication | anApplication perform: thisContext selector]
```

実は、この様に書く事で実行できる。
これを応用すると、もっと余計なコード(コードクローン)を削除していくことが出来るのだが…それは来月の話。