本日は宮崎君回

USB メモリの中身
20120704/
VisualWorks78nc.zip
vw78jun793mac.zip
vw78jun793win.zip

20120704/ の中身は
ClickEarth.html
ClickEarth.pdf
ClickEarth.st
ClickEarth_files

GUI シリーズ
GUI パーツの扱い方をメインにやっていくので、あまりプログラムをがりがり書く感じではないとのこと

VisualWorks を起動
ClickEarth.html を見てみる。
今回やることの一通りの手順が書いてある

System Browser を開く。
KSU Package に Example と System だけがある事を確認

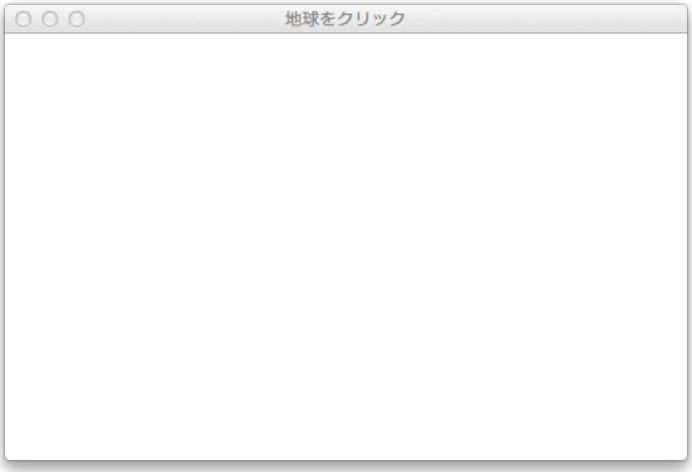
以下を workspace に入力して実行

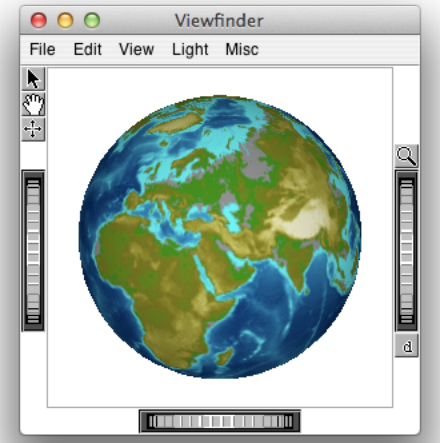
```
| fileInBlock packageBlock |
fileInBlock :=
	[
	[| aFilename |
	 aFilename := JunFileRequesterDialog requestFilename.
	 aFilename ifNil: [^nil].
	 aFilename fileIn]
		on: UserInterrupt
		do: [:anException | anException proceed]].
packageBlock :=
	[| aCollection |
	 (aCollection := OrderedCollection new)
		add: #comment: -> 'Copyright 2008-2012 KSU (Kyoto Sangyo University). All Right Reserved.';
		add: #bundle: -> #KSU;
		add: #package: -> 'KSU-Template';
		add: #nameSpace: -> #KSU;
		add: #category: -> 'KSU-Template';
		yourself.
	JunSystem
		perform: ((aCollection collect: [:each | each key]) inject: String new
			into: [:selector :key | selector , key]) asSymbol
		withArguments: (aCollection collect: [:each | each value]) asArray].
fileInBlock value.
packageBlock value
```

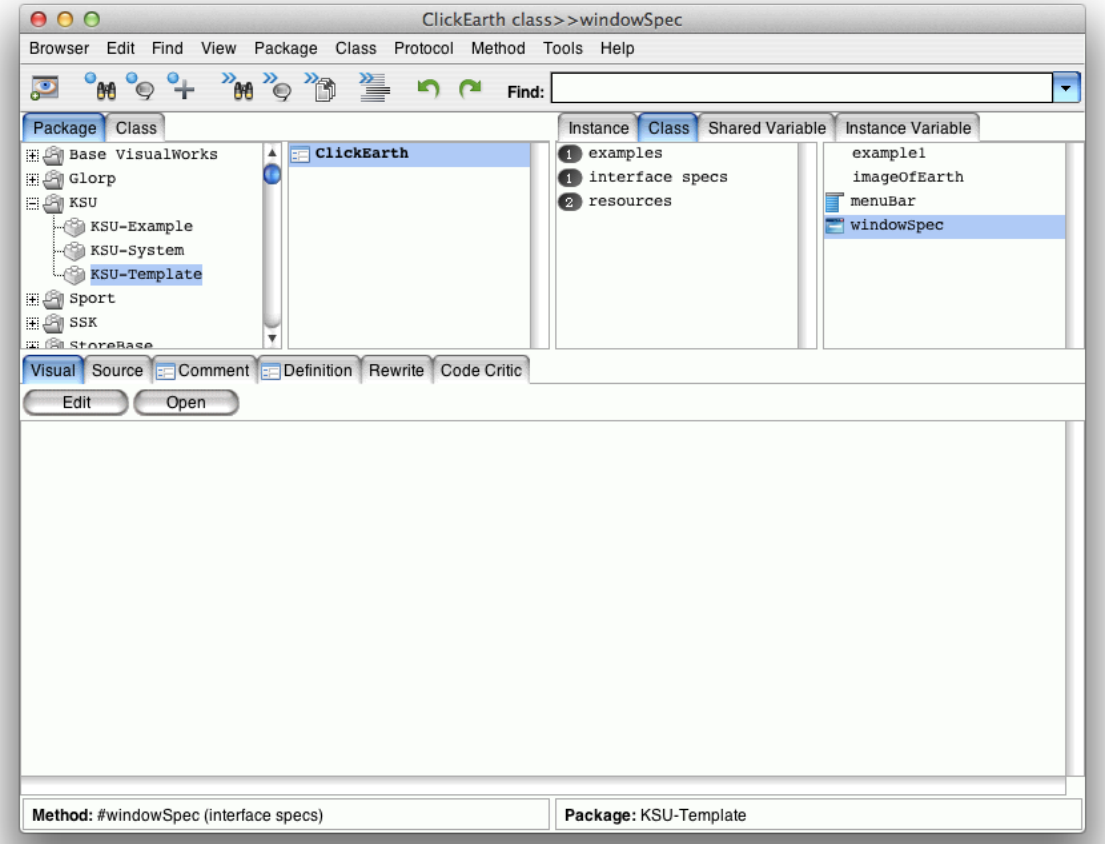
ClickEarth.st.st を選択すると読み込まれる。

KSU Package に KSU-Template が増える。
名前の通り、プログラムファイルのテンプレートが一式、入っている

Template の中の example1 を実行してみると
2つのウインドウが開く
白い方にいろいろと手を加えてプログラムを作成していく

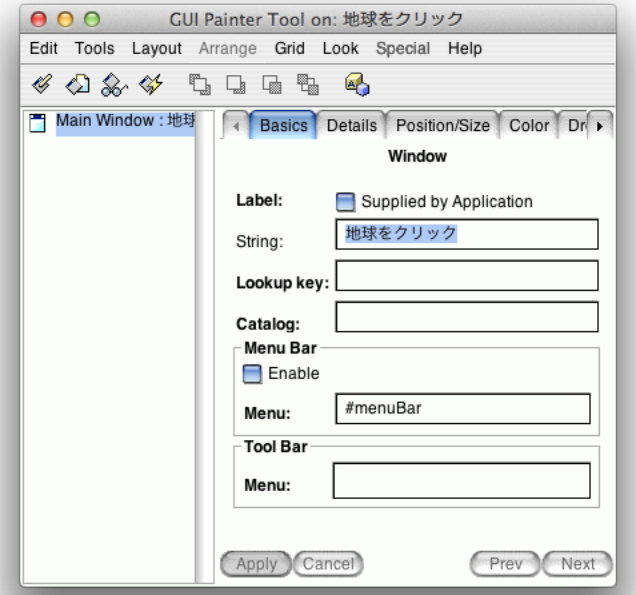




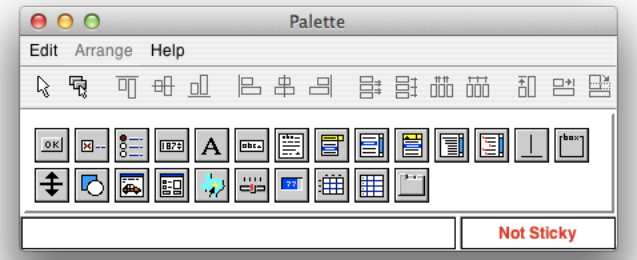


windowSpec から Edit を押して GUI のデザインをやっていく
先ほど開いた画面と同じものが開かれここにデザインをしていく

その親玉

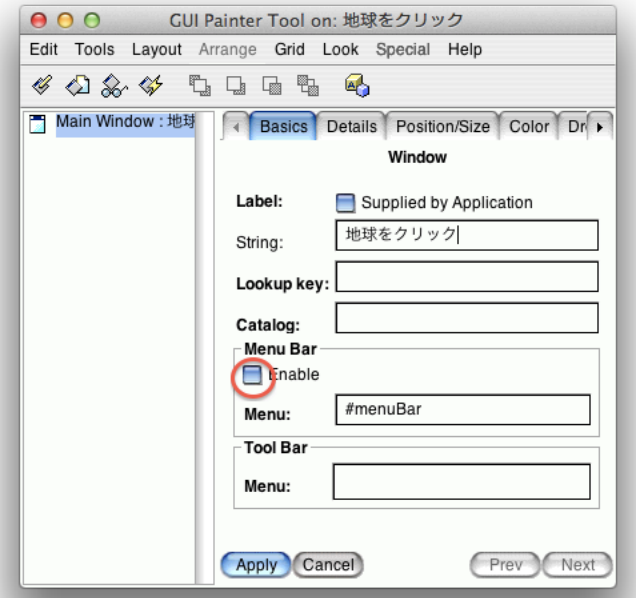


パーツのパレット

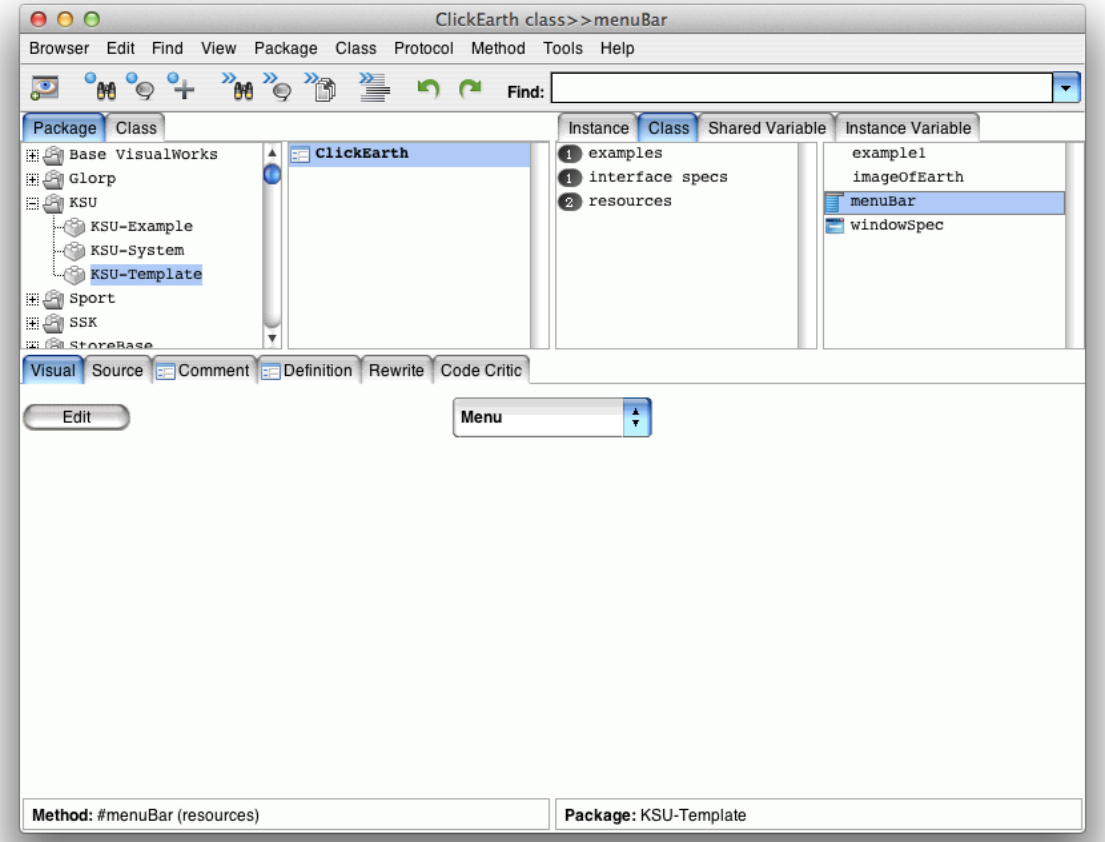


この3つが開く

まずはメニューバーを追加

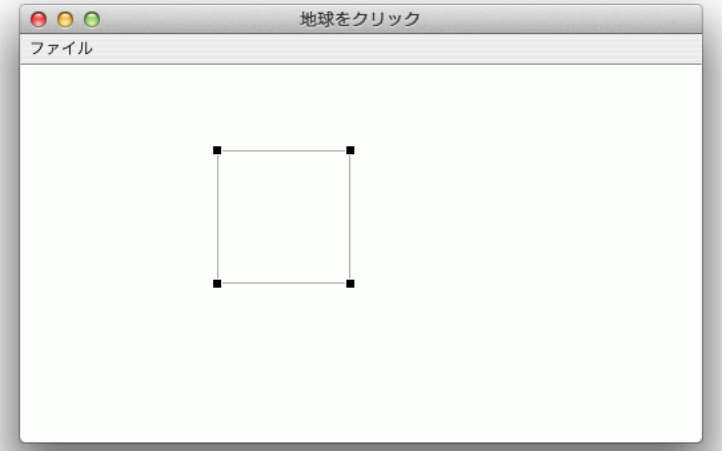


Menu Bar のチェックボックスをチェックして Apply するだけ！ Menu: に #menuBar と既に入っているが普段から元々入っているわけではない(はず)
よく見ると、 Menu に何か書いてあって resources の menuBar があるのでコレでヒモ付いたというだけ

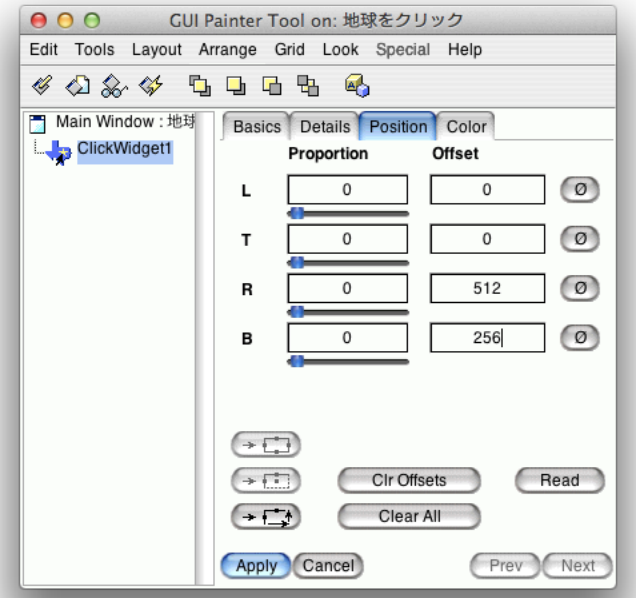


先ほどのパーツのパレットの中の

これが、 Click Widget 。
コレをクリックして、
先ほどの Window の適当な部分をドラッグすると



こんな感じで、枠が出来る
親玉の方に ClickWidget1 が追加されているので、こんな感じに Position を修正
(それぞれ選択すると対応付いているものが選択された状態になる
例を挙げると上の画面の枠を選択すると、下の画面の ClickWidget1 が選択された状態になり
下の画面の ClickWidget1 を選択すると、上の画面の枠が選択された状態になる)



Propotion はいい感じに比率でオフセットしてくれる物
Offset は普通にピクセルで指定

Apply すると

枠が広がる。

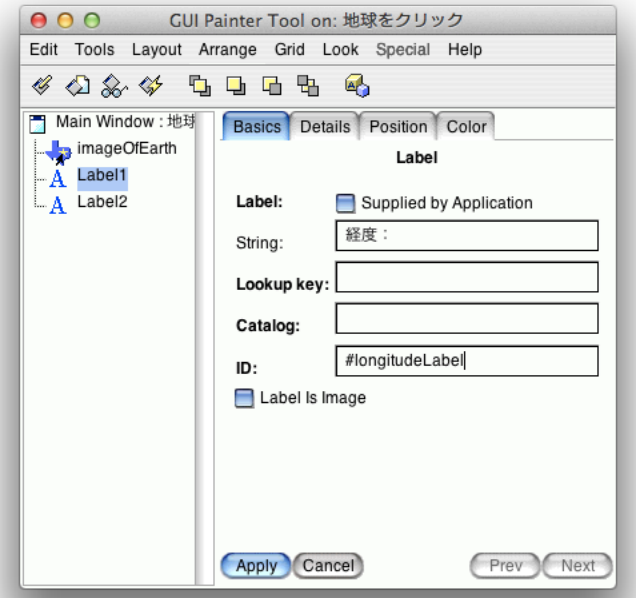
Basics タブを開いて、 ID を #imageOfEarth と指定。
後ほどコレを用いて指定するので、覚えておきましょう。

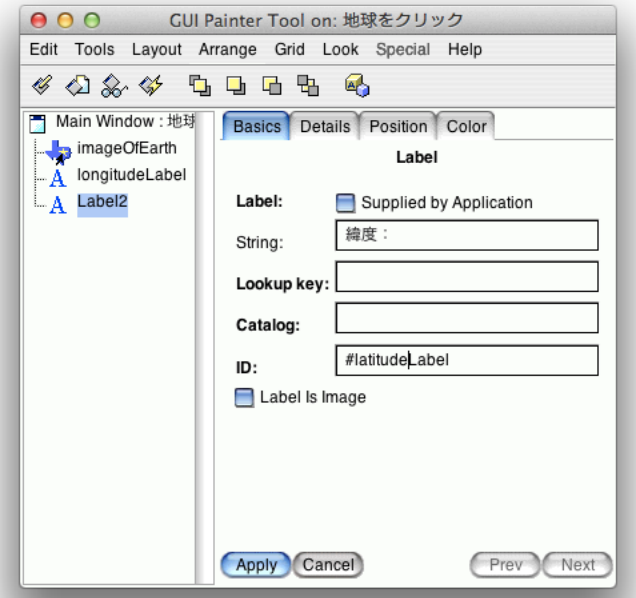
経度と緯度を表示するラベルを付ける


これがラベル

適当に2つ設置

String が表示される文字列
ID は使わないけれども、ちゃんとつけておくように



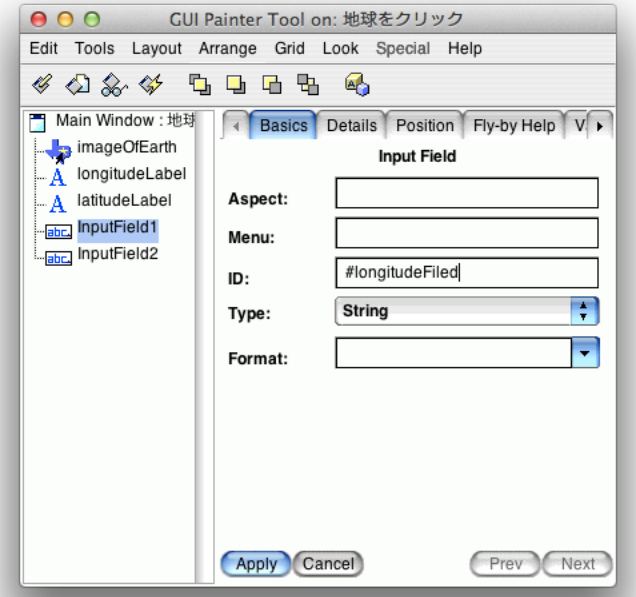


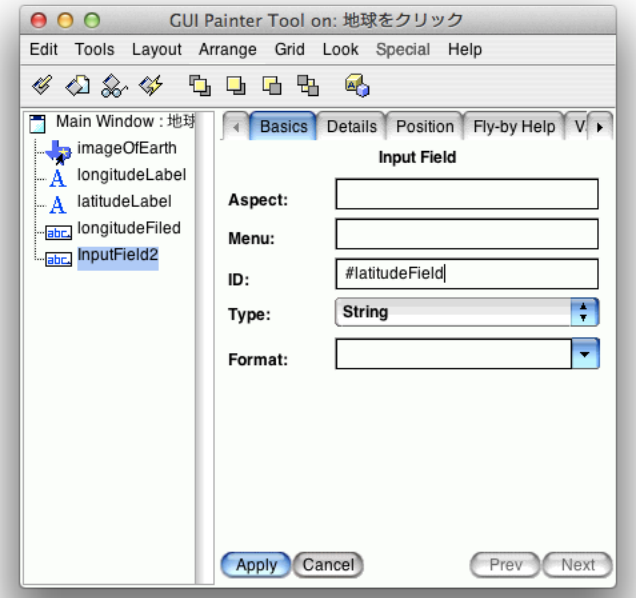
二つをこんな感じで設定して Apply

その結果、こんな感じ。

次は経度と緯度の値を表示するための Input Field を用意
同様に適当に2つ用意

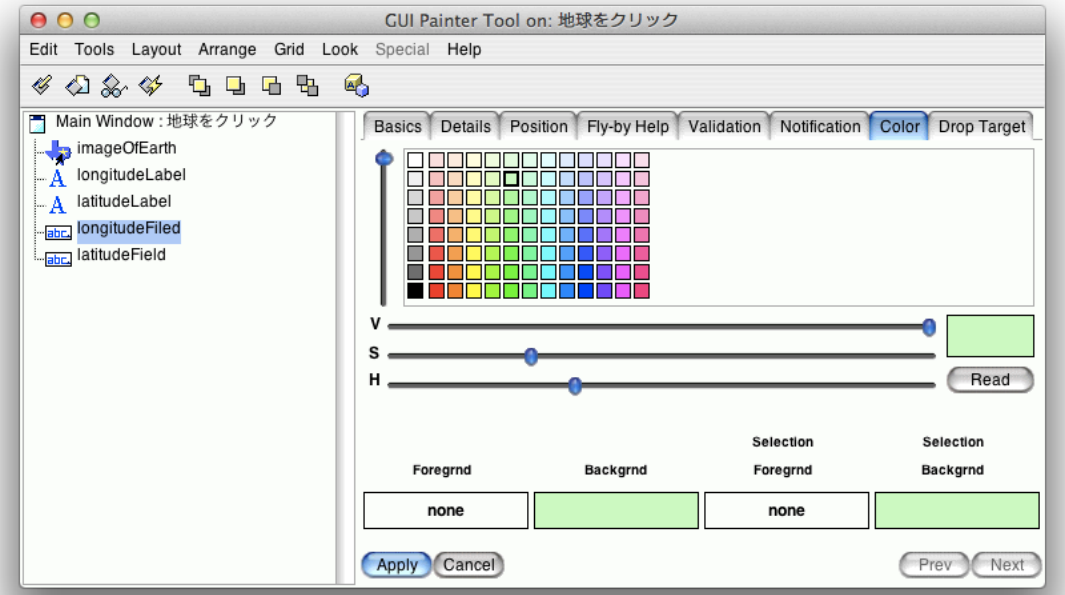
同様に、 2 つのラベルの値を設定していく
ID に #longitudeFiled と入力し Type は String のままで、 Apply

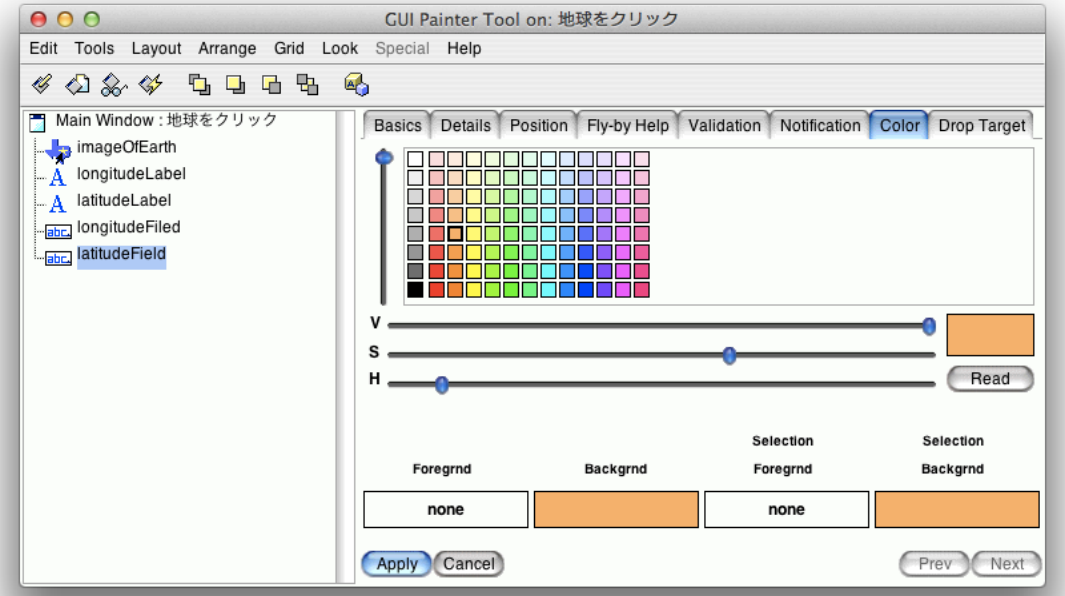




コレで Apply。(恐らく、 Position も何かしら設定したと思うのだが、失敗したので、適当に)

見た目は特に変わらず
label を選んだ状態で Color タブを開く

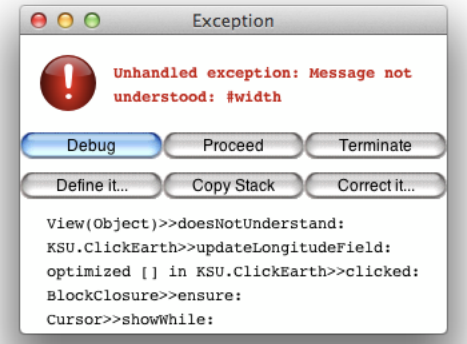




適当に色を選んで指定。すると、ラベルの枠の色が指定した色に変わっている。

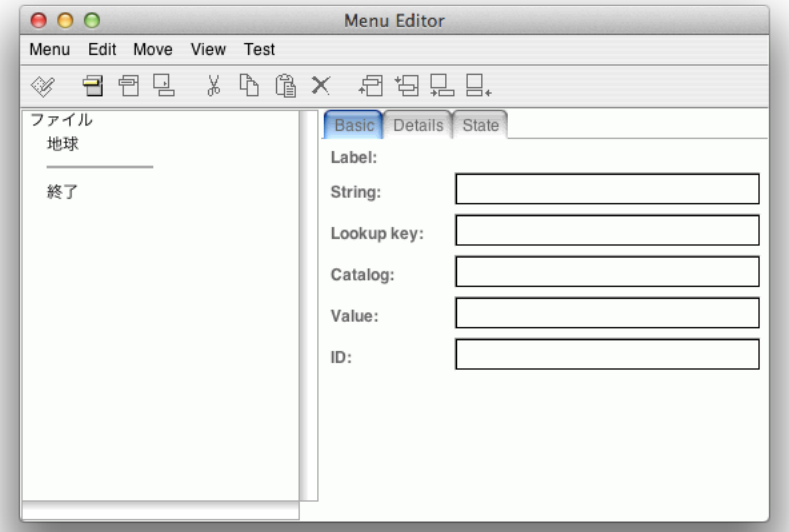

ここまで来たら、 Install する

この時点で example1 を動かすと



なにかしら、やり忘れがあるらしい
ということで作業続行

次は Menu Bar をいじっていくが、先ほどのパーツを置いていく元とは別に専用の Menu Bar エディタがある。
これ



Smalltalk で GUI なアプリケーションを作るときは Application Model のサブクラスを作って、 windowSpec を作っていく

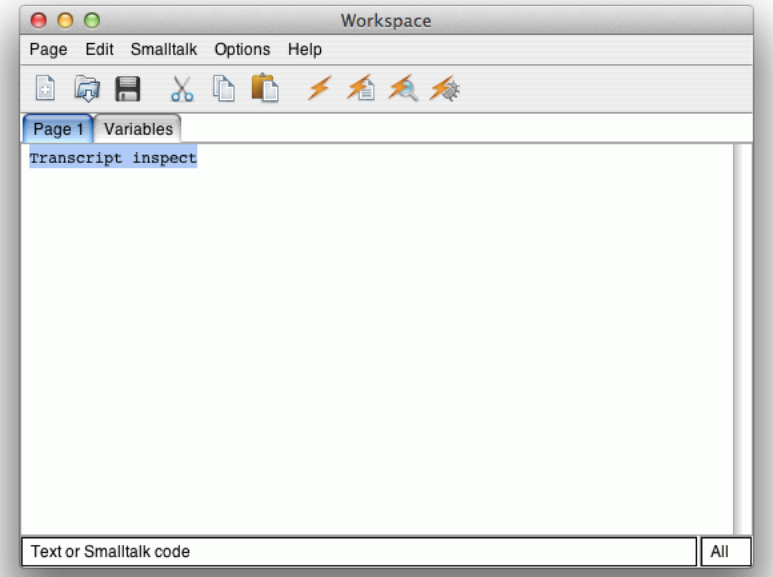
windowSpec のソースを見てみると
KSU-Template, ClickEarth, Class, windowSpec

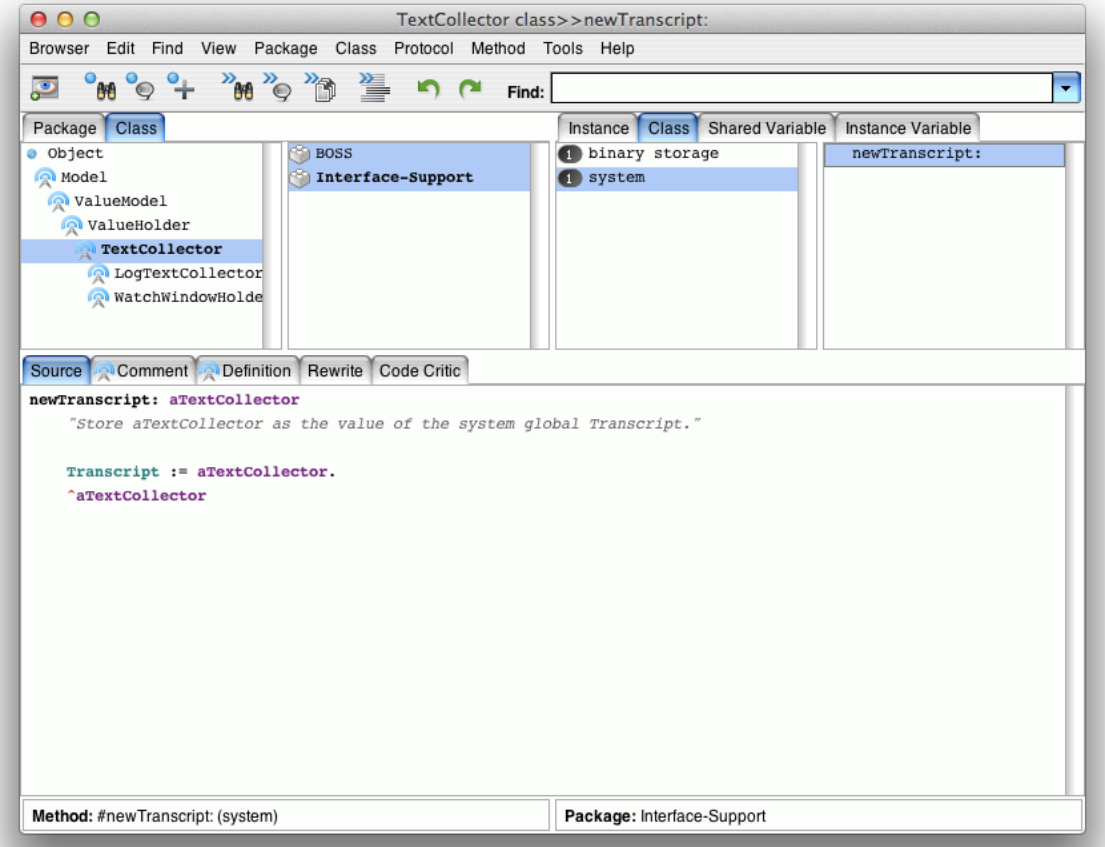
```
windowSpec
	"Tools.UIPainter new openOnClass: self andSelector: #windowSpec"

	<resource: #canvas>
	^#(#{UI.FullSpec}
		#window:
		#(#{UI.WindowSpec}
			#label: '地球をクリック'
			#min: #(#{Core.Point} 512 320 )
			#max: #(#{Core.Point} 512 320 )
			#bounds: #(#{Graphics.Rectangle} 468 280 980 600 )
			#flags: 4 <-- ここを消すと Menu Bar が消える とっても特別扱い
			#menu: #menuBar )
		#component:
以下省略
```

ビットマップエディタとメニューバーエディタはちょっと特別扱いされてる(巨大だからかなぁ)

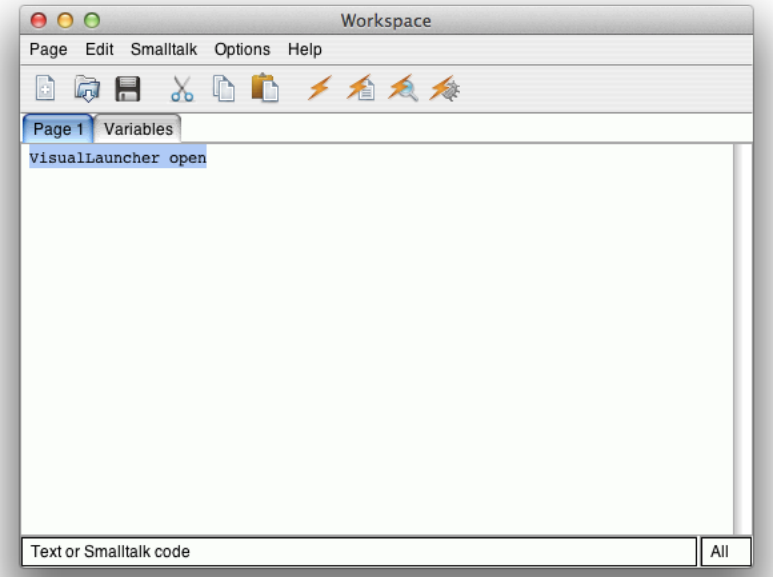
##############ここから、脱線##############
宮崎君の Transcript が行方不明になってしまったので、頑張って探しましょう

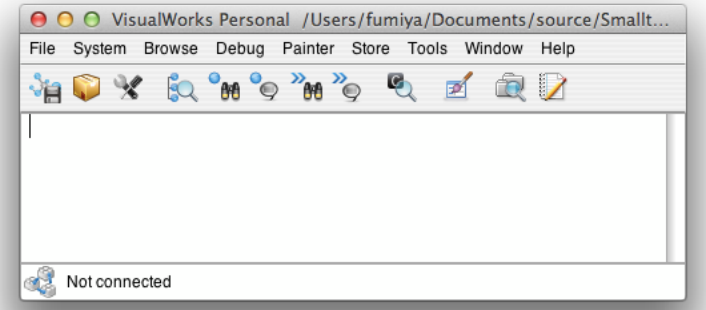




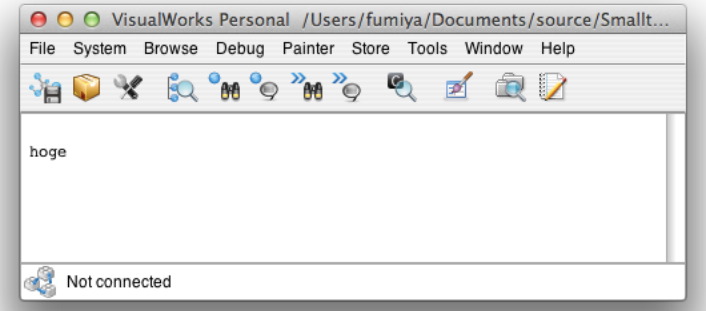
これで、新しいのは作れるけど、今はxを押して、消してしまっただけで、 Transcript のインスタンス自体は生きてるはず
なので、新しく作るのはちょっとイヤ

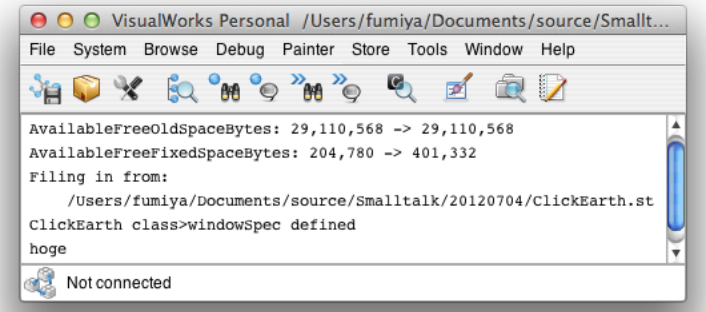
正解はコレで、



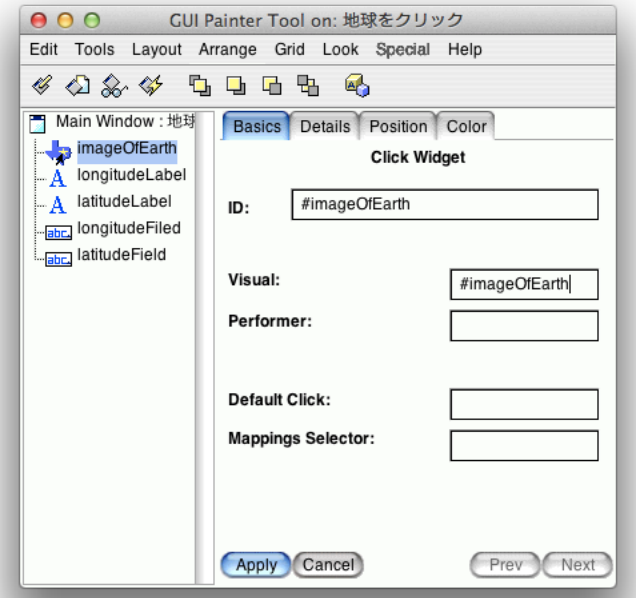


新しく開いた
Transcript は
実態は同じだそうで、
Transcript cr; show: 'hoge' を Do it すると、両方に hoge と表示される

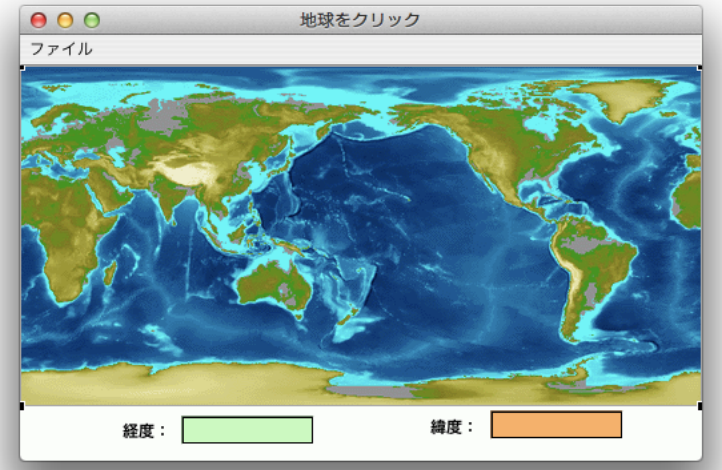




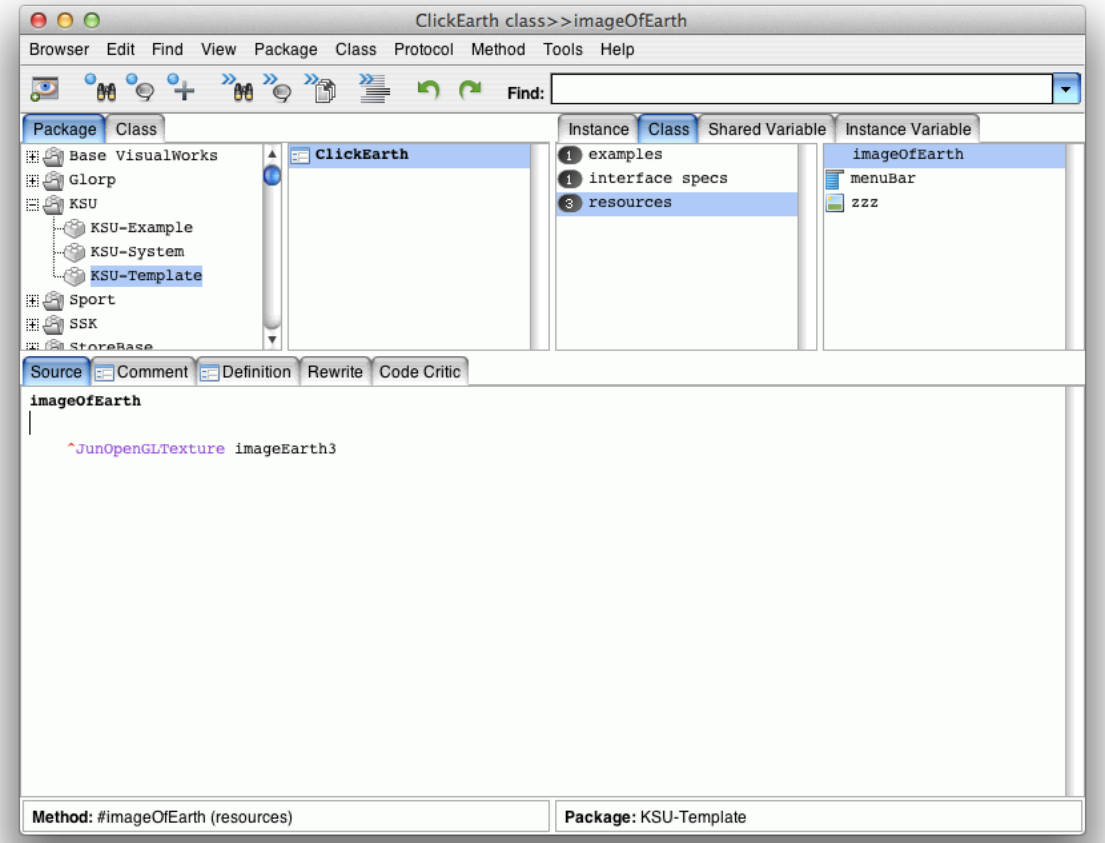
##############ここから本筋に戻って##############
メルカトル図法の地図を表示するために、 windowSpec から Edit



この様に書くと、
Visual に #imageOfEarth と入力して、 Apply すると、



こんな感じ
resources の中から imageOfEarth を探している
実態は画像を用意してくれるメソッド



自分で画像を追加するには

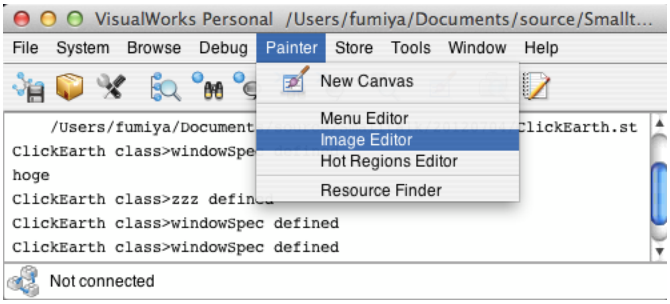
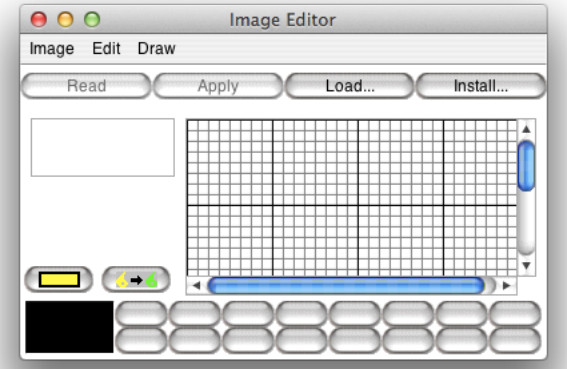
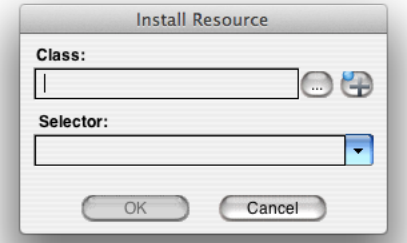
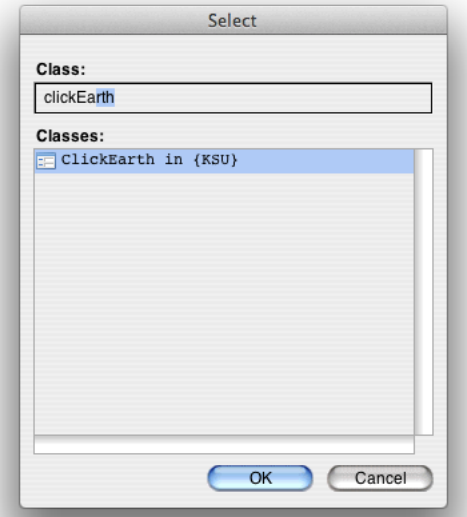


Image Editor を使う

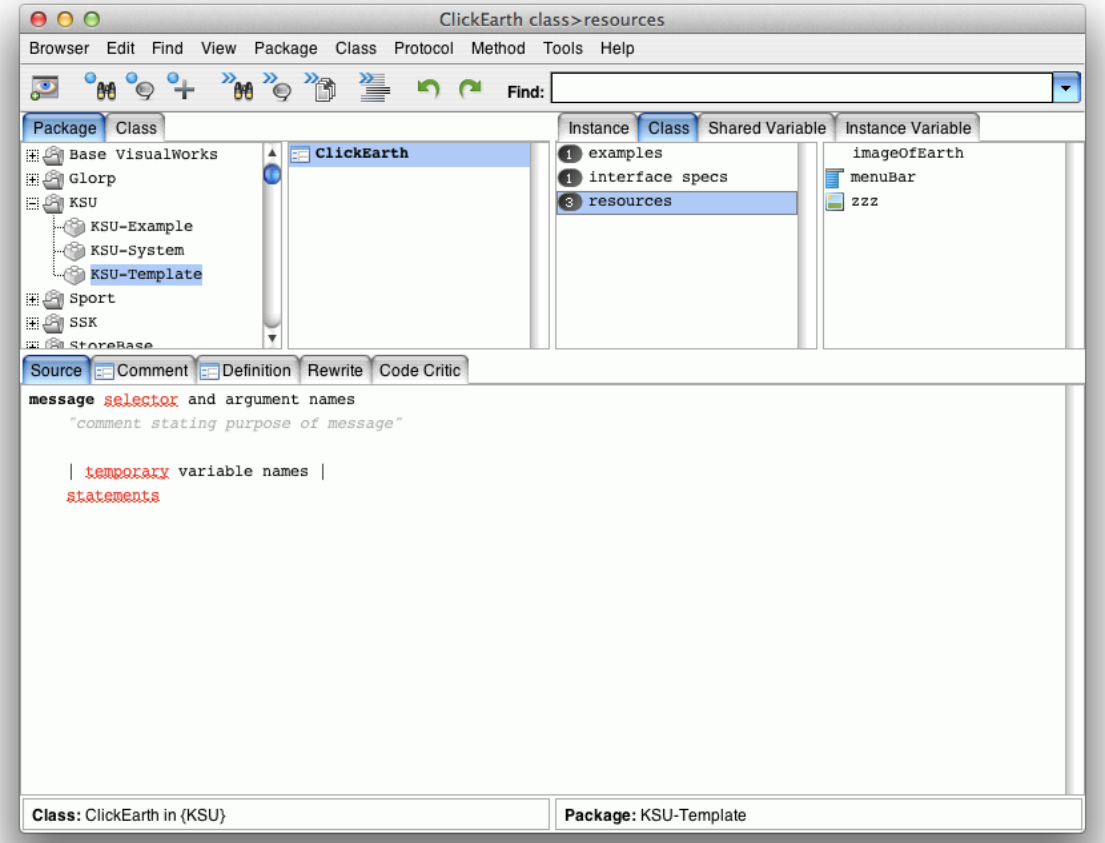


Install を押すと、 Install 先を聞かれる



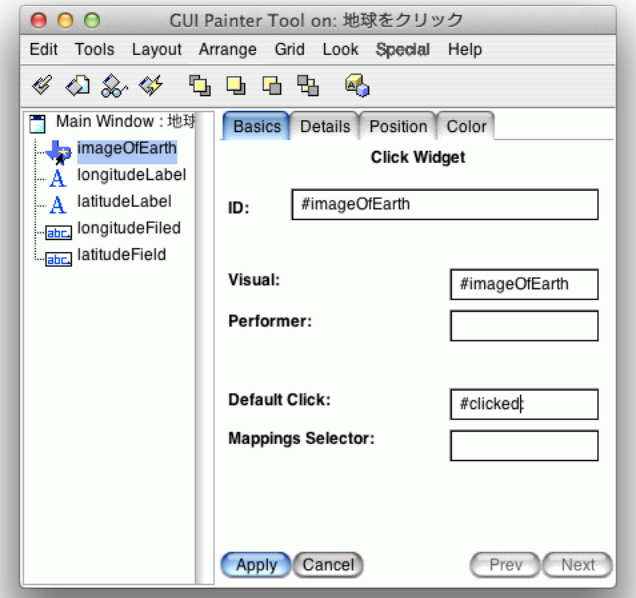


Selector に zzz 等と入れると、



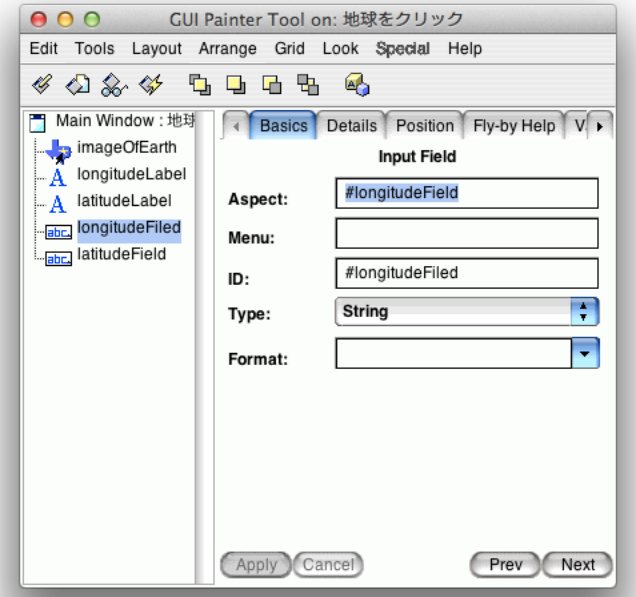
こんな感じでふえる

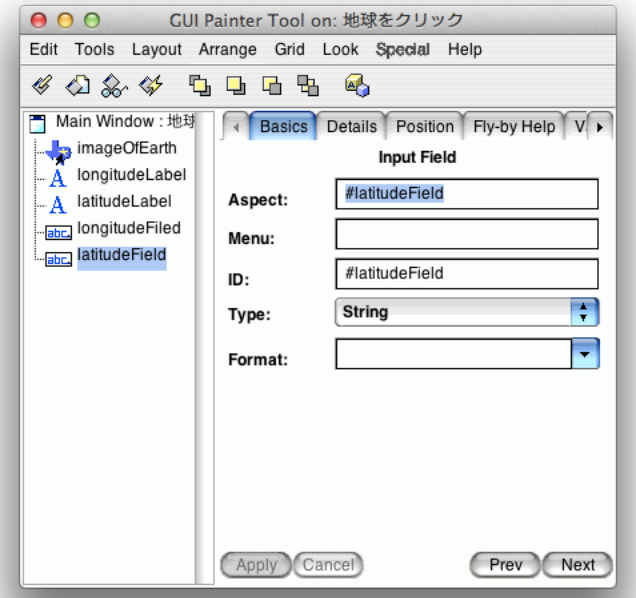
ペインタに戻って



Default Click: に #clicked: とする

経度と緯度の Filed の Aspect を指定





ここまで出来たら Install

Filed の実態はコレ
KSU-Template, Instance, aspects, latitudeField

latitudeField

```
latitudeField ifNil: [latitudeField := String new asValue].
^latitudeField
```

KSU-Template, Instance, aspects, longitudeField

longitudeField

```
longitudeField ifNil: [longitudeField := String new asValue].
^longitudeField
```

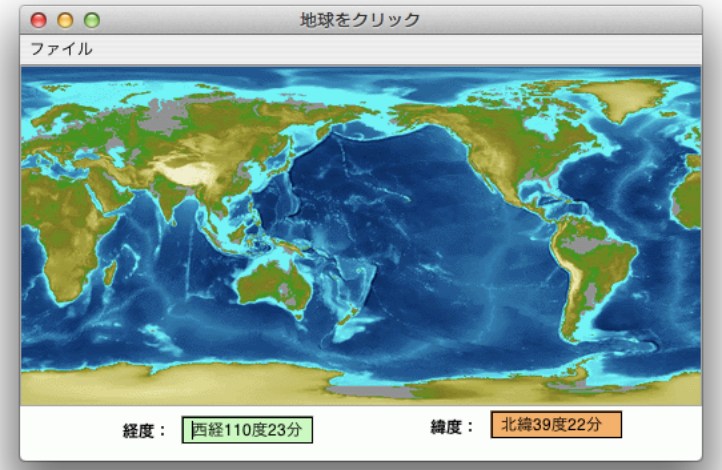
imageOfEarth の #clicked: の実態はコレ
KSU-Template, Instance, actions, clicked:

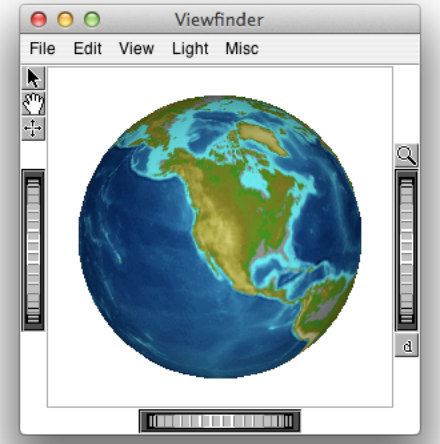
clicked: thePoint

```
| aPoint aWrapper aSensor aBoolean |
aPoint := thePoint.
(aWrapper := self builder
                ifNil: [^nil]
                ifNotNil: [:aBuilder | aBuilder componentAt: #imageOfEarth]) ifNil: [^nil].
aSensor := aWrapper widget controller sensor.
JunCursors crossCursor showWhile:
        [aBoolean := true.
        [aBoolean] whileTrue:
                [aPoint := aSensor cursorPoint.
                aPoint y = 0 ifTrue: [aPoint := aPoint x @ JunGeometry accuracy].
                (self pictureOfEarth bounds containsPoint: aPoint)
                        ifTrue:
                                [self
                                        updateLongitudeField: aPoint;
                                        updateLatitudeField: aPoint;
                                        updateViewfinderOfEarth: aPoint.
                                aBoolean := aSensor shiftDown]]]
```

赤色文字部分が重要で、座標位置の更新をしている部分

この時点で example1 を実行してみると、



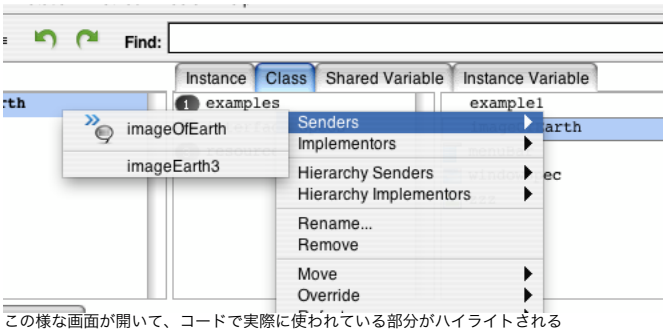


ちゃんと動いた。

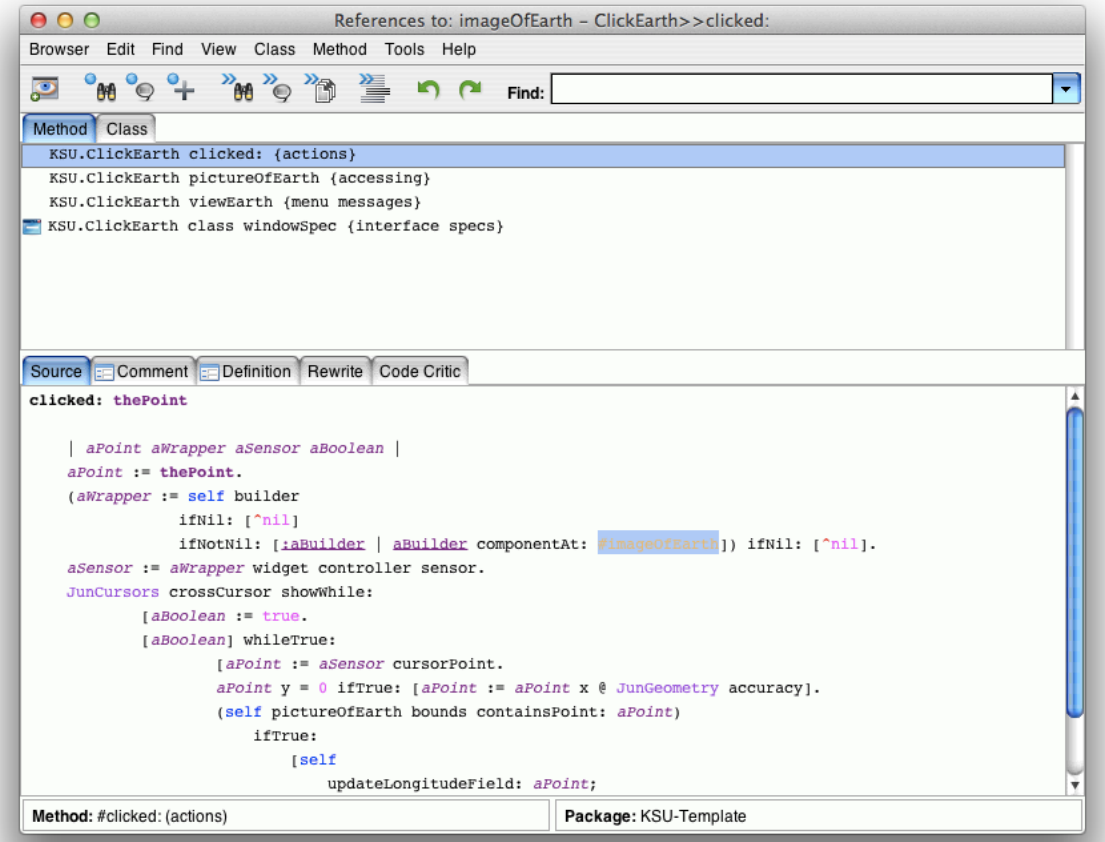
この時点では、 click を離したときにしか反応しない。
例えば、ドラッグをしてもくるくる回らない。

ここまでの話で imageOfEarth の ID を使っていないけど、
imageOfEarth の ID はドラッグ機能を実装するときに使う(既にこのコードでは使われている)

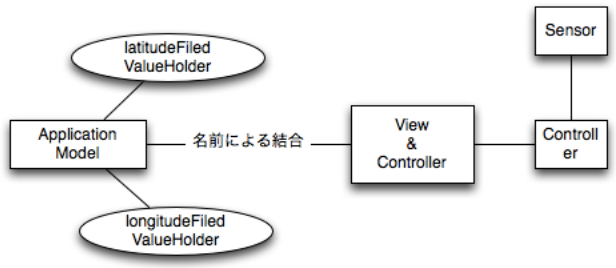
本当に #imageOfEarth を使っている人がいるのか探してみる
探したいものに対して右クリックして、 Senders を選んで一番上の imageOfEarth を選択すると



この様な画面が開いて、コードで実際に使われている部分がハイライトされる



Smalltalk における MVC の関係



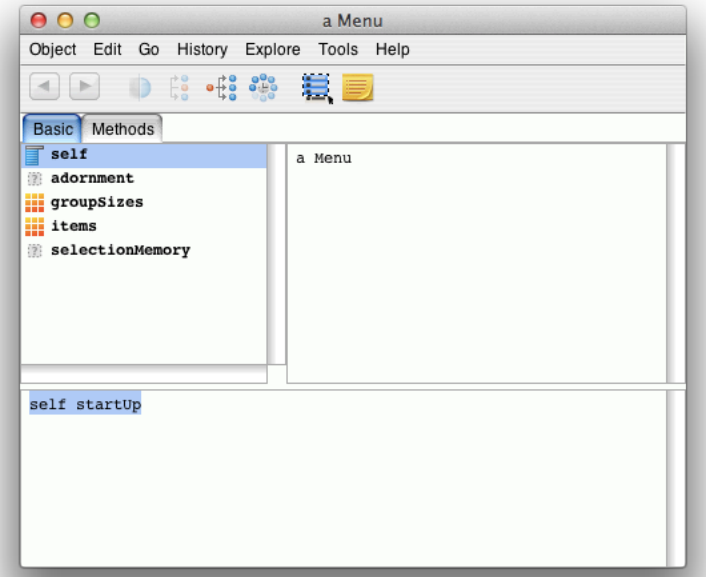
Application Model 側から Sensor の値(クリックした座標など)が欲しければ、 Builder に対して ID を指定して
そこから Sensor をもらってくる
View & Controller は Filed どの Filed とヒモ付けるかに関しては、 aspects に指定して、 Application Model に
どれと対応付けるべきか聞きに行く
Model と View & Controller は名前による結合だけで、疎な結合をしてる

menuBar の Source を見ると、設計図があって、 decode してそれを返せと書いてある
KSU-Template, ClickEarth, Class, resources, menuBar

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

	<resource: #menu>
	^#(#{UI.Menu} #(
			#(#{UI.MenuItem}
				#rawLabel: 'ファイル'
				#submenu: #(#{UI.Menu} #(
						#(#{UI.MenuItem}
							#rawLabel: '地球'
							#value: #viewEarth )
						#(#{UI.MenuItem}
							#rawLabel: '終了'
							#value: #closeRequest ) ) #(1 1 ) nil ) ) ) #(1 ) nil ) decodeAsLiteralArray
```

青色部分を inspect it してみると、



Menu が返ってくる
workspace を開いて self startUp とすると

menuBar が出てくる

既に clicked: があるけど、(現在のバージョンは良い感じに修正されているので)古いバージョンで話を進めために
現在のバージョンはバックアップしておく
_clicked: thePoint

```
| aPoint aWrapper aSensor aBoolean |
aPoint := thePoint.
(aWrapper := self builder
                ifNil: [^nil]
```

以下省略

先頭に _ を付けて accept

古いバージョンを改めて作る
KSU-Template, ClickEarth, Instance, actions, clicked:

clicked: aPoint

```
(self pictureOfEarth bounds containsPoint: aPoint) ifFalse: [^nil].
self
        updateLongitudeField: aPoint;
        updateLatitudeField: aPoint;
        updateViewfinderOfEarth: aPoint
```

それを修正
KSU-Template, ClickEarth, Instance, actions, clicked:

clicked: aPoint

```
| aSensor |
(self pictureOfEarth bounds containsPoint: aPoint) ifFalse: [^nil].
self
        updateLongitudeField: aPoint;
        updateLatitudeField: aPoint;
        updateViewfinderOfEarth: aPoint.
aSensor := (self controllerAt: #imageOfEarth) sensor. "<-- Click Widget の sensor をもらう"
aSensor altDown inspect "<-- alt(option) キーが押されているかを inspect して確認"
```

昔は yellowButton などと呼んでいたけど、 Windows のせいで alt キーと…
KSU-Template, ClickEarth, Instance, actions, clicked:

clicked: aPoint

```
| aSensor |
(self pictureOfEarth bounds containsPoint: aPoint) ifFalse: [^nil].
self
        updateLongitudeField: aPoint;
        updateLatitudeField: aPoint;
        updateViewfinderOfEarth: aPoint.
aSensor := (self controllerAt: #imageOfEarth) sensor.
```

```
[aSensor altDown] whileTrue: "<-- alt キーを押し続けている間は"
        [Processor yield.
        Transcript
            cr;
            show: aSensor cursorPoint printString]
```

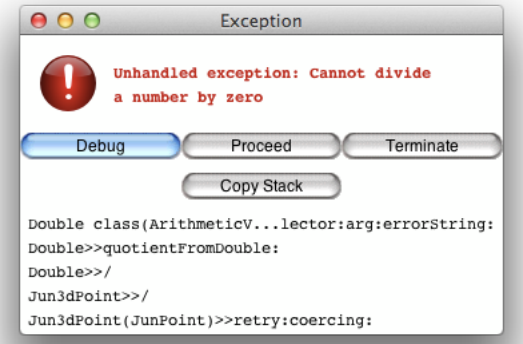
clicked (クリックをやめた後)が呼ばれたあとに alt が押されていたら反応する
ドラッグではない
KSU-Template, ClickEarth, Instance, actions, clicked:

clicked: aPoint

```
| aBlock aSensor |
aBlock :=
        [:thePoint |
        (self pictureOfEarth bounds containsPoint: thePoint) ifFalse: [^nil].
        self
            updateLongitudeField: thePoint;
            updateLatitudeField: thePoint;
            updateViewfinderOfEarth: thePoint].
aBlock value: aPoint.
aSensor := (self controllerAt: #imageOfEarth) sensor.
[aSensor altDown] whileTrue:
        [Processor yield.
        aBlock value: aSensor cursorPoint]
```

コードクローンがあると嫌なので、座標アップデート部分をクロージャ化して、更新するように

ドラッグ状態(alt キーが押された状態)で北極の方を超えると

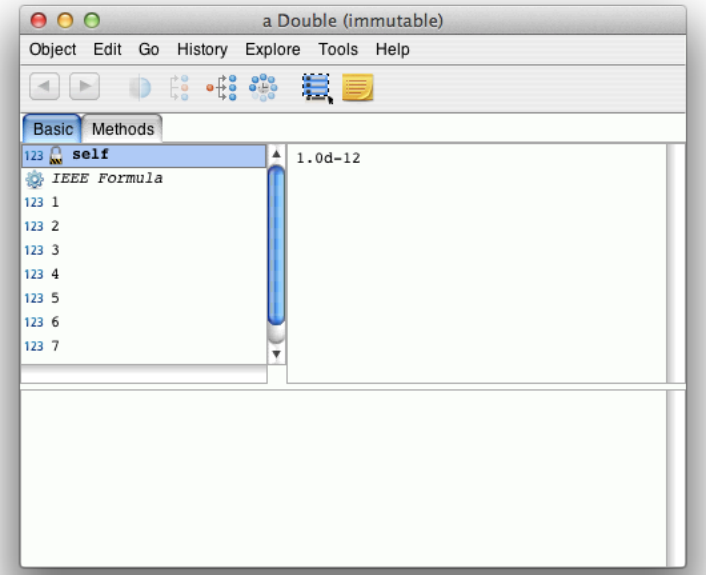


zero で割るなと怒られる
北極の方に座標を出すときに除算が含まれているので、こうなってしまう

clicked: aPoint

```
| aBlock aSensor |
aBlock :=
        [:thePoint |
        thePoint y = 0 ifTrue: [thePoint y: JunGeometry accuracy].
        (self pictureOfEarth bounds containsPoint: thePoint) ifFalse: [^nil].
        self
            updateLongitudeField: thePoint;
            updateLatitudeField: thePoint;
            updateViewfinderOfEarth: thePoint].
aBlock value: aPoint.
aSensor := (self controllerAt: #imageOfEarth) sensor.
[aSensor altDown] whileTrue:
        [Processor yield.
        aBlock value: aSensor cursorPoint]
```

青色部分を inspect すると



限りなく 0 に近い値なので、これで 0 除算はでなくなる

この時点では、メルカトル図法の地図からはみ出ると、それ以降、更新されなくなる
バックアップを取った元のコードでは、その問題は起きない
KSU-Template, ClickEarth, Instance, actions, _clicked:
_clicked: thePoint

```
	| aPoint aWrapper aSensor aBoolean |
	aPoint := thePoint.
	(aWrapper := self builder
			ifNil: [^nil]
			ifNotNil: [:aBuilder | aBuilder componentAt: #imageOfEarth]) ifNil: [^nil].
	aSensor := aWrapper widget controller sensor.
	JunCursors crossCursor showWhile:
		[aBoolean := true.
		[aBoolean] whileTrue:
			[aPoint := aSensor cursorPoint.
			aPoint y = 0 ifTrue: [aPoint := aPoint x @ JunGeometry accuracy].
			(self pictureOfEarth bounds containsPoint: aPoint)
				ifTrue:
					[self
						updateLongitudeField: aPoint;
						updateLatitudeField: aPoint;
						updateViewfinderOfEarth: aPoint.
					aBoolean := aSensor shiftDown]]]
```

それを直すために ifFalse で nil を返して抜け出しているので、逆に ifTrue の時だけ更新する様にして
抜け出すことがないようにすると動き続けるようになる
KSU-Template, ClickEarth, Instance, actions, clicked:

clicked: aPoint

```
	| aBlock aSensor |
	aBlock :=
		[:thePoint |
		thePoint y = 0 ifTrue: [thePoint y: JunGeometry accuracy].
		(self pictureOfEarth bounds containsPoint: thePoint)
			ifTrue:
				[self
					updateLongitudeField: thePoint;
					updateLatitudeField: thePoint;
					updateViewfinderOfEarth: thePoint]].
	aBlock value: aPoint.
	aSensor := (self controllerAt: #imageOfEarth) sensor.
	[aSensor altDown] whileTrue:
		[Processor yield.
		aBlock value: aSensor cursorPoint]
```