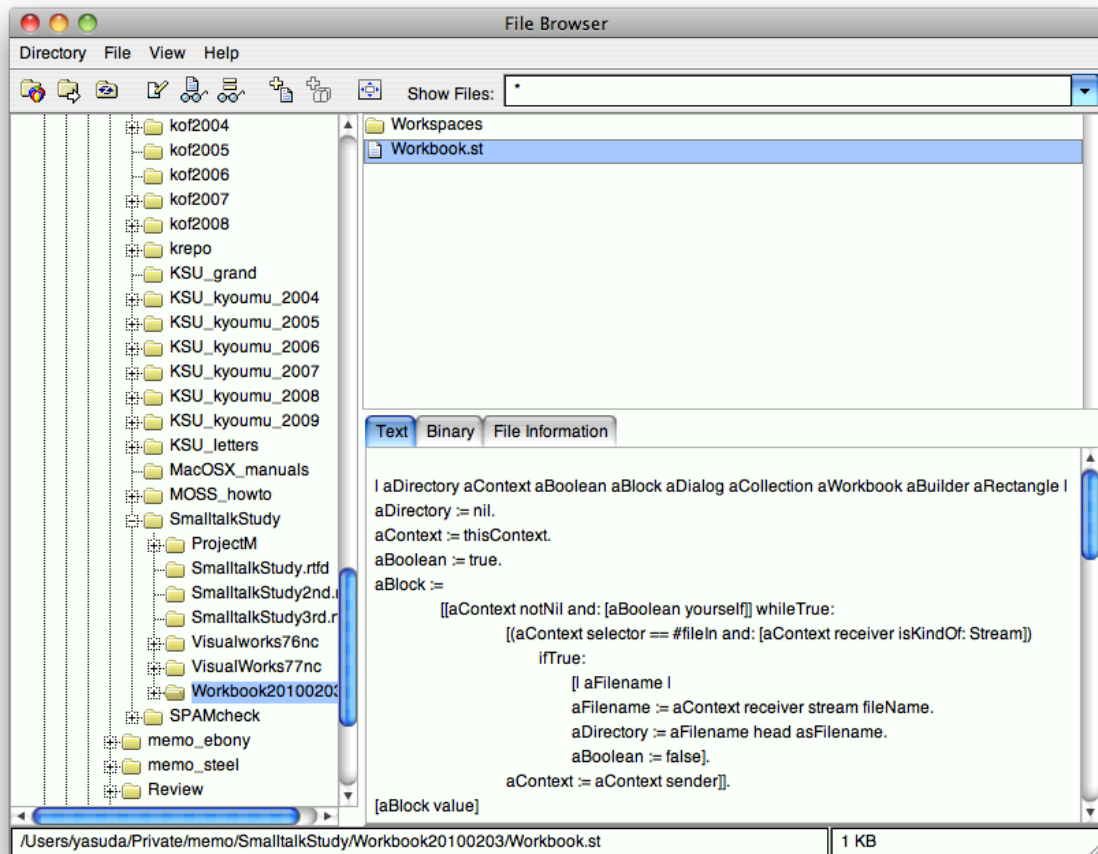


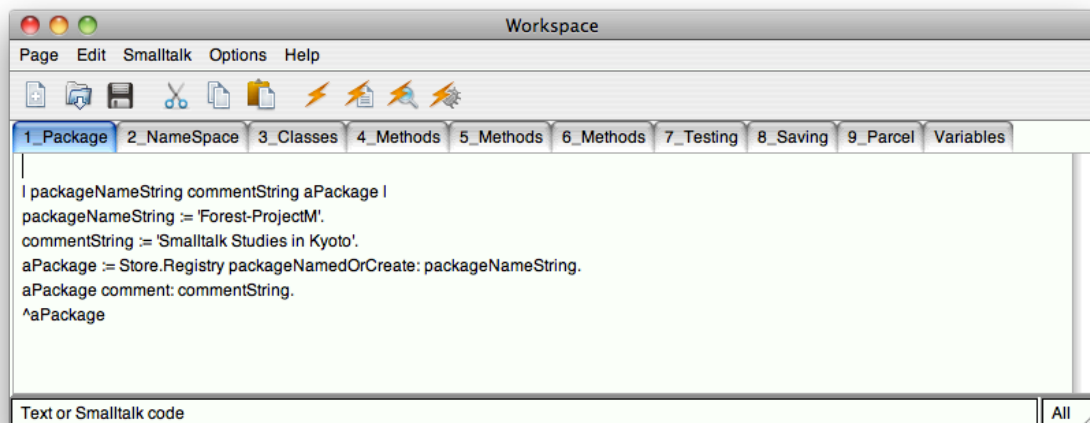
□ 2010.2.3

今回から 7.7 のフルセットで、かつ Jun なしになっている。1.1GBほどある。

Packages メニューから New Package をやって、それを選んで NameSpace を作って、クラスを作って、という感じでやっていく。



ファイルブラウザを開いて（F2 キーか、File メニューの一番上）
Workbook.st ファイルを File in する。
すると Workspace にどどっとタブが開いて読み込まれる。



この 1 - 9 までの手順が基本的な一連の作業なので、このようにして作業するのだと思えば良い。

普通は Smalltalk の GUI を使ってやるのだが、それを書き下したものだと思えば良い。

（Unix システムで全部コマンドラインでできるのが当たり前である、そうでなく GUI が必要なのは未熟である、ということと一緒に、なのか？）

= 1_Package

所定のパッケージを作る。これって恐らく New Package メニューなんだろうなあ。

```
| packageNameString commentString aPackage |
packageNameString := 'Forest-ProjectM'. ★これがパッケージ名。
commentString := 'Smalltalk Studies in Kyoto'. ★こちらコメント
aPackage := Store.Registry packageNameCreate: packageNameString. ★このような名前のパッケージがあれば返す、なければ作る
aPackage comment: commentString.
^aPackage
```

これを doit してからシステムブラウザを見ると（ただし Browser メニューの Refresh (F5) をせんとあかんかも）Forest-ProjectM というクラスができてい

= 2_NameSpace

```
| nameSpaceSymbol packageNameString commentString aNameSpace aPackage |
nameSpaceSymbol := #ProjectM.
packageNameString := 'Forest-ProjectM'.
commentString := 'Smalltalk Studies in Kyoto'.
aPackage := Store.Registry packageNameCreate: packageNameString. ★またパッケージを作るんだが orCreate が無い（ので無いならエラーになる）
aPackage ifNil: [^nil].
aNameSpace := Smalltalk at: nameSpaceSymbol ifAbsent: [nil]. ★ProjectMなどという Namespace はまだ無いので nil になる
aNameSpace
  ifNil: ★nil だったので作る
    [aNameSpace := Smalltalk
      defineNameSpace: nameSpaceSymbol
      private: false
      imports: 'private Smalltalk.'"
      category: packageNameString. ★この一文で NameSpaceを一つ作った
      aNameSpace comment: commentString. ★コメントつけて
      aNameSpace yourself].
([aNameSpace package] ★カッコについては後述：ここはPackage を指定せずにNamespaceを作ったので、それを
on: aNameSpace messageNotUnderstoodSignal
do: [:exception | Store.Registry containingPackageForNameSpace: aNameSpace])
  = aPackage ★これは比較をしている。ここも後述。
  ifFalse:
    [Store.XChangeSet current moveWholeObject: aNameSpace toPackage: aPackage].
^aNameSpace
```

() でくくっているのは 7.7 では動くけれども 7.6 では動かないことへの対処。do: exception 以下に（messageNotUnderstoodSignalがあったら）、7.6 相

当のコードを書いておいた。

= aPackage で比較しているのは、

「パッケージがもし違ったら、正しいパッケージにその NameSpace をつなぎなおせ」

といった処理をするつもりらしい。結局、NameSpace を作る処理 Smalltalk defineNameSpace: ... にパッケージ指定がないのが問題というか根源なんだな。

つまり Package を指定せずにまず作って、後で既存のパッケージに放り込む（XchangeSet..）という処理をすることになる。

まあ慣習、かな。

= 3_Classes

クラスを三つ作る。Model, View, Controller

```
| nameSpaceSymbol packageNameString commentString aPackage aNameSpace aCollection |
nameSpaceSymbol := #ProjectM.
packageNameString := 'Forest-ProjectM'.
commentString := 'Smalltalk Studies in Kyoto'.
aPackage := Store.Registry packageNameCreate: packageNameString.
aPackage ifNil: [^nil]. ★まあここまでは一緒ね。
aNameSpace := Smalltalk at: nameSpaceSymbol ifAbsent: [^nil].
aCollection := (OrderedCollection new) ★配列を作る
  add: (Array with: #Model with: #{UI.Model} with: 'forest picture'); ★classのシンボル(Model)、superclass のシンボル(UI.Model)、インスタ
  ns変数 (forest と picture)
  add: (Array with: #View with: #{UI.View} with: 'offset');
  add: (Array with: #Controller with: #{UI.Controller} with: "");
  yourself.
^aCollection collect: ★このコレクションを巡りつつ、値を取り出して Class 定義を行う
  [:anArray |
  | aClass |
  aClass := aNameSpace
    defineClass: (anArray at: 1) ★ class のシンボル (ex. Model)
    superclass: (anArray at: 2) ★superclass のシンボル(ex. UI.Model)
    indexedType: #none
    private: false
```

```

instanceVariableNames: (anArray at: 3) ★インスタンス変数 (ex. forest と picture)
classInstanceVariableNames: " ★これはシングルクォート二つ (つまり空文字) だよー (ダブルクォートのコメントじゃないよ)
imports: " ★同上
category: packageNameString. ★これはカテゴリであってパッケージ指定ではないよ (上と同じくパッケージ無しで作られる)
aPackage addClass: aClass. ★なので後で明示的に目的のパッケージに突っ込むように書く
aClass comment: commentString.
aClass yourself]

```

= 4_Methods

今度はModel クラスに対して必要なメソッドを登録する。

```

| aClass aCollection |
aClass := ProjectM.Model. ★クラスをとりあえず手元にとってきて、
aCollection := (OrderedCollection new)
    add: (Array with: 'accessing' with: ' ' ★んでこっから、
        ★ (取り除かれる無用な改行群 (下記参照) の #1)

picture
    ^picture
        ★ (取り除かれる無用な改行群 (下記参照) の #2)
        ); ★ここまでが method なのだ。この下のコードでこれを分解 (解釈) して当て込み処理するのだ
    add: (Array with: 'accessing'
        with: '

picture: anImage
    picture := anImage

        ');
    yourself.
^aCollection collect: ★ここから上で作った変数 aCollection を解釈して取り出す
    [:anArray |
        | aProtocol aCode startIndex endIndex aSelector aMethod |
        aProtocol := (anArray at: 1) asSymbol. ★第一要素は 'accessing' だぞ (上を見よ)
        aCode := (anArray at: 2) yourself. ★第二要素が例の ' から ' までの数行 (上を見よ) (なおここ yourself じゃなくて asString が本当かな?)
        startIndex := aCode findFirst: [:aCharacter | aCharacter isSeparator not]. ★前の改行群 (#1) を除く
        endIndex := aCode reverse
            findFirst: [:aCharacter | aCharacter isSeparator not]. ★後ろの改行群 (#2) を除く
        endIndex < 1
            ifTrue: [endIndex := aCode size]
            ifFalse: [endIndex := aCode size - endIndex + 1].
        aCode := aCode copyFrom: startIndex to: endIndex.
        aSelector := aClass compile: aCode classified: aProtocol. ★クラスを対象にコンパイルを指定
        aMethod := aClass compiledMethodAt: aSelector.
        aMethod yourself]

```

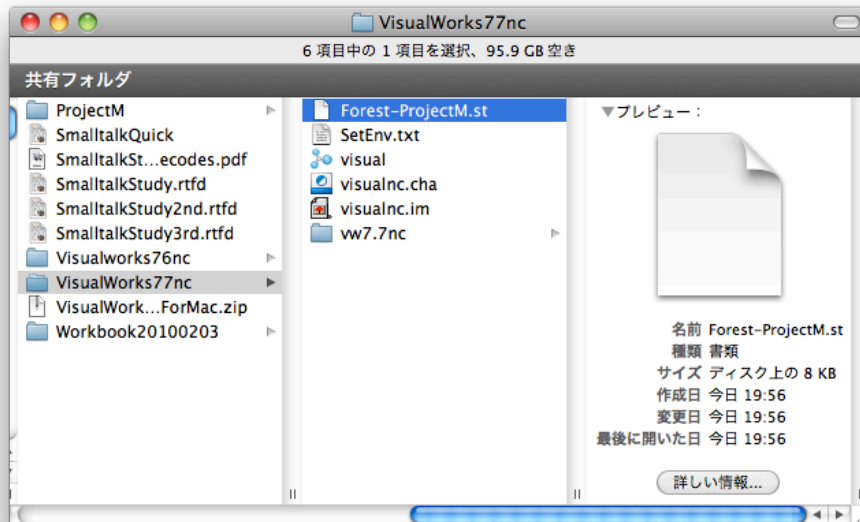
= 5, 6 Methods はそれぞれ残りの二つのクラスにメソッドを作るのみ

= 7_Testing

単にテストを行う。

= 8_Saving

できあがった Package の内容を全部 file out する。



VisualWorks のあったディレクトリに .st つきで書き出された。

本質的に重要なのは以下の★の fileOutOnFileNamed: だけか。

```
| packageNameString aPackage aFilename |
packageNameString := 'Forest-ProjectM'.
aPackage := Store.Registry packageNamed: packageNameString.
aPackage ifNil: [^nil].
aFilename := Filename defaultDirectory construct: packageNameString , '.st'.
aPackage fileOutOnFileNamed: aFilename. ★ここで出力
^aPackage
```

=9_Parcel

パーセル形式で出力する。 .pcl と .pst の二つのファイルに分かれている。

.pcl はコンパイルしたものが入る。

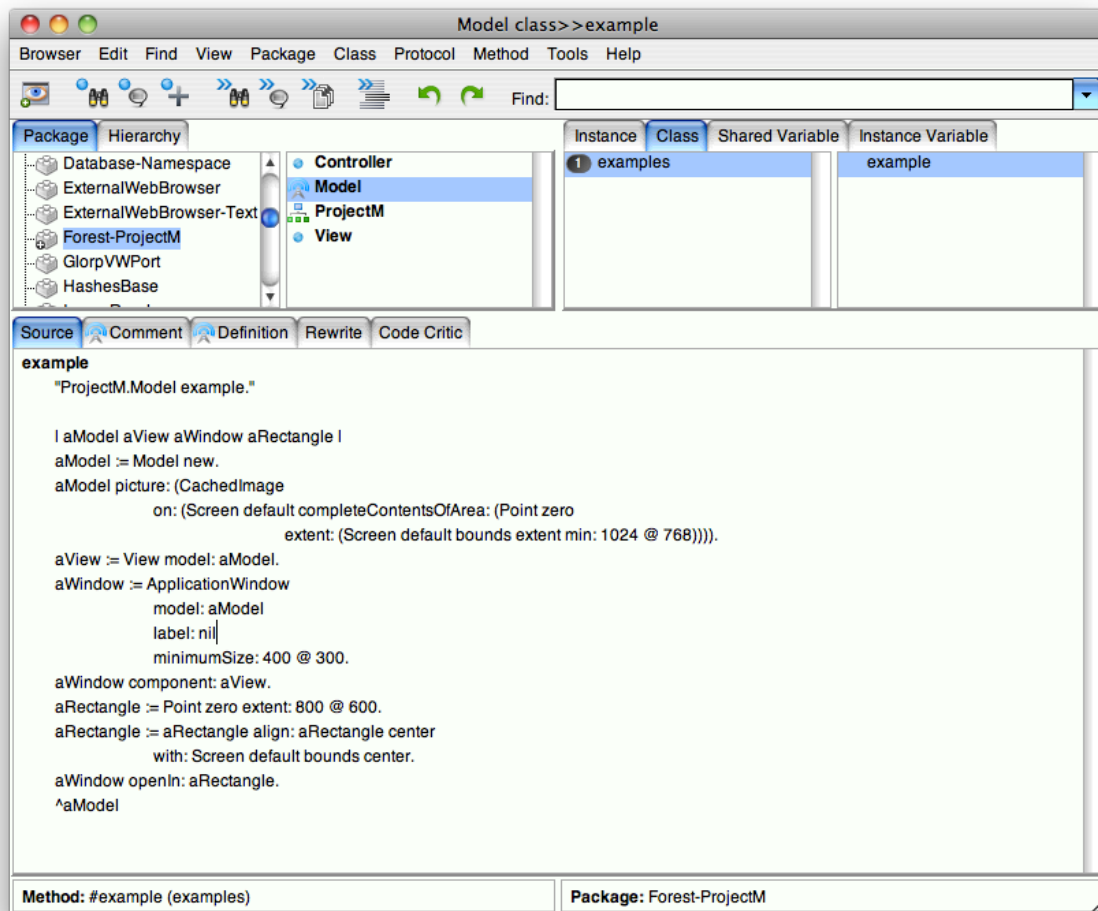
.pst はソースコードが入る。

両者は対照されており、ブラウザに読み込ませると、コンパイルされている pcl を信用してそのまま導入し、.pst をそのまま信用してソースとして入れる。普通は .pst は更新しない。修正すると再コンパイルすることになるしね。

```
| packageNameString nameSpaceSymbol commentString aPackage aNameSpace aParcel |
packageNameString := 'Forest-ProjectM'.
nameSpaceSymbol := #ProjectM.
commentString := 'Smalltalk Studies in Kyoto'.
aPackage := Store.Registry packageNamed: packageNameString.
aPackage ifNil: [^nil].
aNameSpace := Smalltalk at: nameSpaceSymbol ifAbsent: [^nil].
aParcel := Parcel parcelNamed: packageNameString.
aParcel isNil
    ifTrue:
        [aParcel := Parcel createParcelNamed: packageNameString.
         aParcel comment: commentString].
aParcel addNameSpace: aNameSpace. ★ Namespace をパーセルに追加
aPackage allClasses do: [:aClass | aParcel addEntiretyOfClass: aClass]. ★パッケージの中のクラスを巡りつつ、クラスごとにパーセルに追加
aParcel
    parcelOutOn: packageNameString , '.pcl' ★ここで出力
    withSource: packageNameString , '.pst' ★ここも出力
    hideOnLoad: false
    republish: false
    backup: false.
^aParcel
```

fileOut の st と Parcel のpst はそれほど変わらないように思うが、まあ開発者が違うとかそういうので統一されないのであろうなあ。

==さて example からコードを読む



example

```
"ProjectM.Model example."
```

```
| aModel aView aWindow aRectangle |
aModel := Model new. ★とりあえず作る
aModel picture: (CachedImage
    on: (Screen default completeContentsOfArea: (Point zero
        extent: (Screen default bounds extent min: 1024 @ 768)))). ★ピクチャとしてスクリーンキャプチャを入れる
aView := View model: aModel. ★View さんよ、君の Model は aModel だよ、といって作り出す
aWindow := ApplicationWindow
    model: aModel ★なおこのモデルを指定する必要はない、らしい（なくても動く）
    label: nil
    minimumSize: 400 @ 300. ★それとは別に非常に小さなウィンドウを一つ作る
aWindow component: aView.
aRectangle := Point zero extent: 800 @ 600. ★矩形を作って
aRectangle := aRectangle align: aRectangle center
    with: Screen default bounds center. ★画面の真ん中に出すようにして
aWindow openIn: aRectangle. ★ウィンドウを開ける
^aModel
```

file:///Users/yasuda/Private/memo/SmalltalkStudy/ProjectM/BasicDesign/index.html
の 外観図「ウィンドウ構成とスクロール描画」を見ると、上のあたりの関係図がある。

これ、実行すると分かるが、つまり画面上にイメージ（今回はたまたまキャプチャ画像）を表示して、ドラッグしてスクロールさせる、というアプリ。（ひながた）
このイメージのところに樹状図を放り込めば課題のアプリは完成なのだ。

===さて View の各メソッドの説明

displayOn: 画面を表示しなさい、と言われたとき

```

displayOn: aGraphicsContext
  aGraphicsContext
    paint: ColorValue white;
    displayRectangle: self bounds. ★まず画面全部を白色に塗りつぶして表示して、
  model ★モデルに対して
    ifNotNil: ★別に ifNotNilでテストしなくても良いけど
      [model picture ifNotNil: [:it | it displayOn: aGraphicsContext at: offset]] ★

```

update: まあよくわからんけどおきまりでこう書いてねと（update イベントか）

```

update: aNode
  self invalidateNow ★ invalidate てのは validate でなくなる、ということなので、これをもらったウィンドウシステムは再描画しようと思う

```

このとき、再描画すべき領域の検定をして伝えないといけないだろうが、ここでは何もしていない。
面倒なので「全部描いて！」でやってる。

順番的には update が先で（自分で起こす）、するとシステムが displaying を呼んでくれるので（GraphicsContext に再描画が必要な region 情報なども詰めた状態で）、それに反応して再描画するようなコードを displayOn に書いておけばよい。

次、スクロール。

```

scrollTo: aPoint
  offset := aPoint
scrollBy: aPoint
  self scrollTo: offset + aPoint

```

スクロールはオフセットなどを更新するだけなので、ここで update を本当は後で呼ぶんだろなあ。誰が？？

さて Controller。

redButtonPressEvent などイベント処理なのだが、これらは各色とも mouseButtonActivity に流合する。

```

mouseButtonActivity
  Cursor crossHair showWhile: ★ボタンが押されたらまず十字カーソルに変更
    [| symbol |
      symbol := self mouseButtonChecking. ★自分自身に対してボタンをチェックせえと言う
      symbol = #click ifTrue: [self clickActivity]. ★もし click だったらこれを投げ直せ
      symbol = #grab ifTrue: [self grabActivity]]. ★もし grab だったらこれを投げ直せ
    self sensor waitNoButton ★ボタンが押されなくなるまで待て（離されるまで待て）

```

というわけで次、mouseButtonChecking

```

mouseButtonChecking
  | mouse limit |
  mouse := self sensor.
  limit := Time millisecondClockValue + 333. ★今の時刻が +333ms に達しない間に、
  [Time millisecondClockValue < limit] whileTrue: ★ボタンが、
    [mouse anyButtonPressed ifFalse: [^#click]. ★離されたら、これはクリックという動作であろう
      Processor yield]. ★CPU にちょっと処理を回す
  ^#grab ★もし離されなかったら、これはグラブであろう

```

というわけで、クリックかグラブの判定をしている。
どへえ、これくらいやってよ。

はい次、clickActivity

```

clickActivity
  Transcript
    cr;
    show: thisContext printString

```

何もしない。Transcript にデバッグメッセージを出して終わり。
次、グラブ。

```

grabActivity
  | mouse previous current |
  mouse := self sensor.
  previous := current := mouse cursorPoint. ★現在の座標をとってきて、
  [mouse anyButtonPressed] whileTrue: ★
    [current := mouse cursorPoint. ★今の値をとってきて、
      previous = current ★以前の値と比べて違っていたら、
      ifFalse:
        [(self view)
          scrollBy: current - previous; その差分のぶんだけ scrollBy して、

```

```
displayOn: view graphicsContext. ★displayOn を直接呼び出して再描画させる（後述）
previous := current]]
```

displayOn: view graphicsContext. で直接再描画をしているが、ここを
invalidate; としたらだめ。ドラッグが全部終わってから最後に描く。
invalidateNow; としたら、すぐ再描画する。まあここに
update; と書いてもいいかもしれん、が、、、

これ、本当なら update しなければならないか、と思ったら、基本的には View に何か変更があった時 update を呼んでやる、という感じだそうで。
Controller と View は普段密結合していて、Controller から update を呼ぶようなことはない、という慣習らしい。
特に 上のドラッグの例で、直接 displayOn: を呼び出している（ウィンドウシステムとの再描画に関する協調処理をとる必要がないと考えている）のは、この
ウィンドウがドラッグされるときは、必ずフォーカスされた状態であるため。協調せんでもええさ、と。

なお、drag 中にブロック待ち的に待つので、そのあたりにちゃんと CPU を回すためには、

```
[mouse anyButtonPressed] whileTrue:
    [current := mouse cursorPoint.
     .....
     previous := current].
Processor yield] ★これを追加せえ
```

とすべきかと。

で、以上。終わり。

□ 2010.3.3

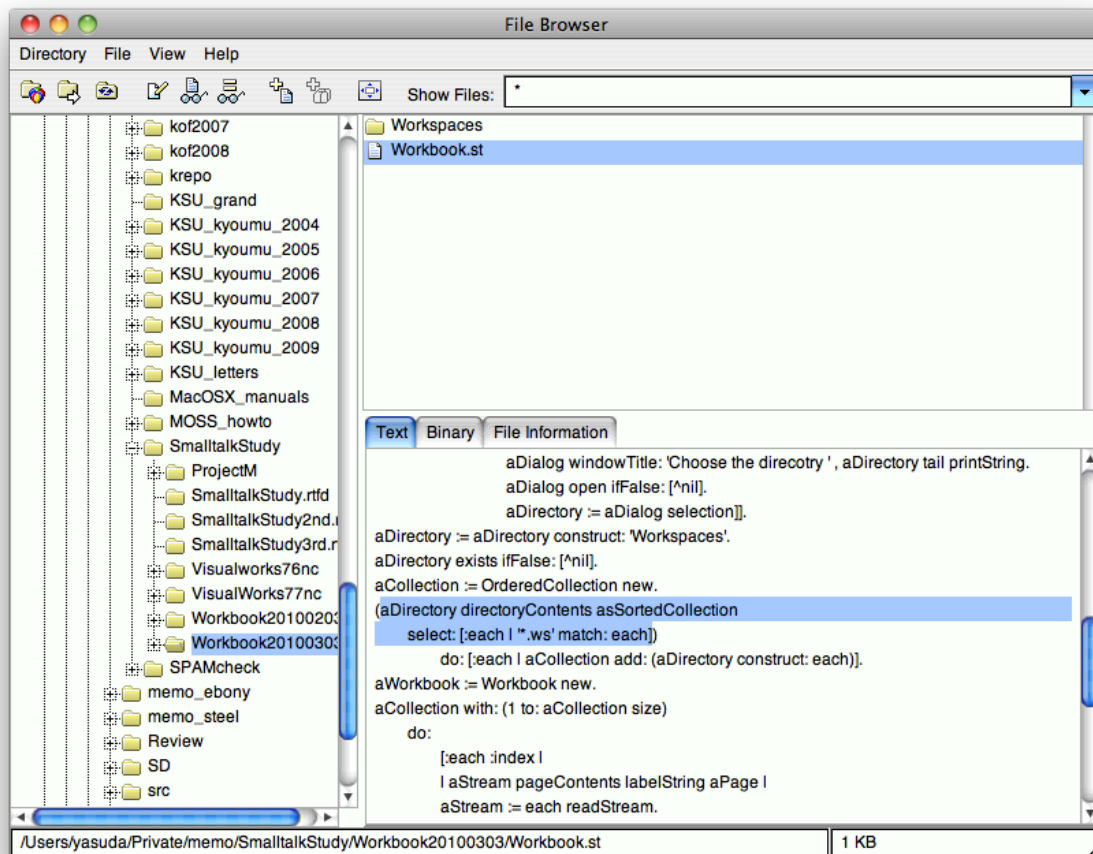
今回、ProjectM/BasicDesign/images/Forest_BD01.jpg の下半分（model から下）を作る。

図から読めるのは、

- ・ Model は Forest Class を一つだけ持つ。（ 1 と図にあるよね）
- ・ Forest は Node （節）を 0 以上持っている（0..* とあるよね）
- ・ Forest は Branch （枝）を 0 以上持っている
- ・ Node, Branch とともに superclass は Object （頂上オブジェクト）である。（つまりあまり意味はない）
- ・ Branch は Node を二つ持つてよ。（枝の両端にはノードがあるよ、という表現。つまり枝を実体化しておくのだ。これはGUI操作のためね。）
- ・ Example クラスは特に必要ないのだけれど、まあ例題を記録するために作った。（Smalltalk 流儀としては要らない。Java などとの対応のため。）
- ・ 名前空間は ProjectZ とする。

隣接行列でもってノード間の関係（論理的なもの）を表現し、グラフィックス的な表現についてはこの枝を中心とした表現にする。

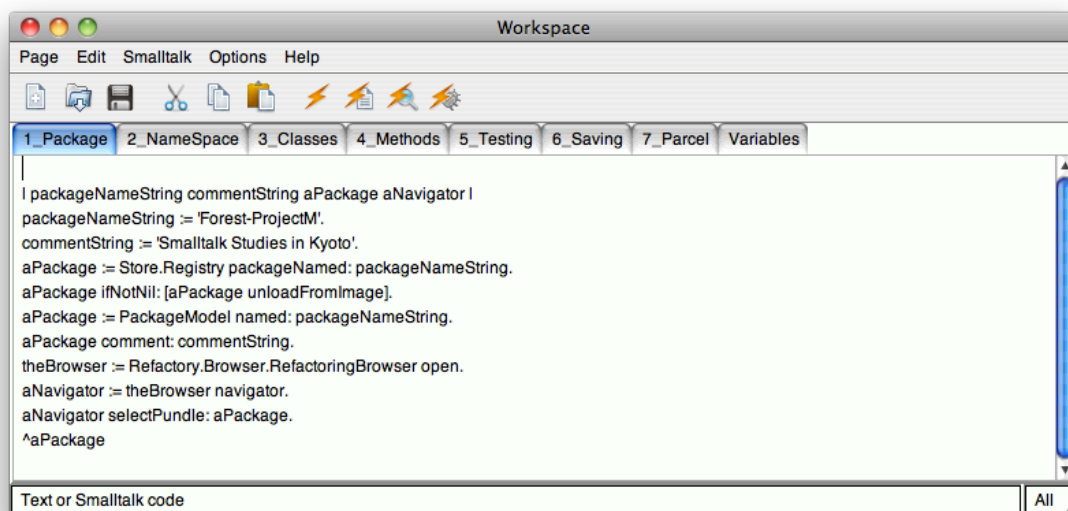
さて作業開始。File Manager で今回の workbook ディレクトリのなかの Workbook.st を読み込ませてみる。



前回との相違はここだけ。

```
(aDirectory directoryContents asSortedCollection
  select: [:each | '*.ws' match: each])
```

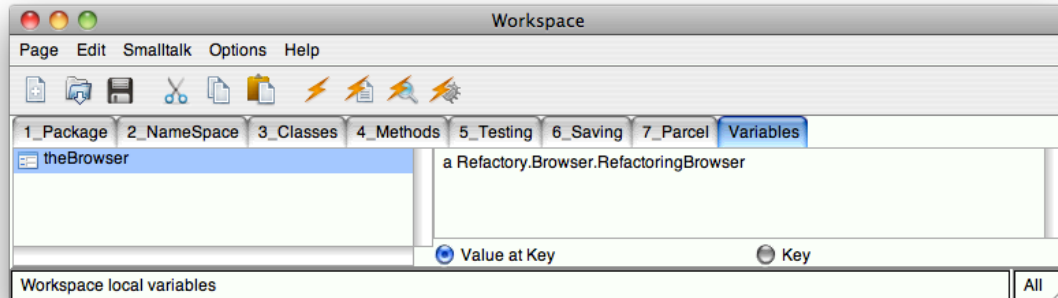
つまり *.ws でもって取り上げるファイル名についてフィルタリングしている。だけ。
で、コンテキストメニューで Filein する（あるいは開いて doit ）とワークスペースが開いて、必要なものが並ぶ。



・まず 1_Package。

```
aPackage ifNotNil: [aPackage unloadFromImage].
```


とあるので、つまり前回にロードしたパッケージがもし存在していたら全部消せ（今回入れ直したから）、というようになっている。
 ああおそしい。
 theBrowser := Refactory.Browser.RefactoringBrowser open.
 aNavigator := theBrowser navigator.
 aNavigator selectPundle: aPackage.
 とあるが、the としている（aでない）のは、この変数を workspace 内の一時変数にする、という意味（慣用）。
 で、実際 doit すると、Variables のところに theBrowser が確認できる。



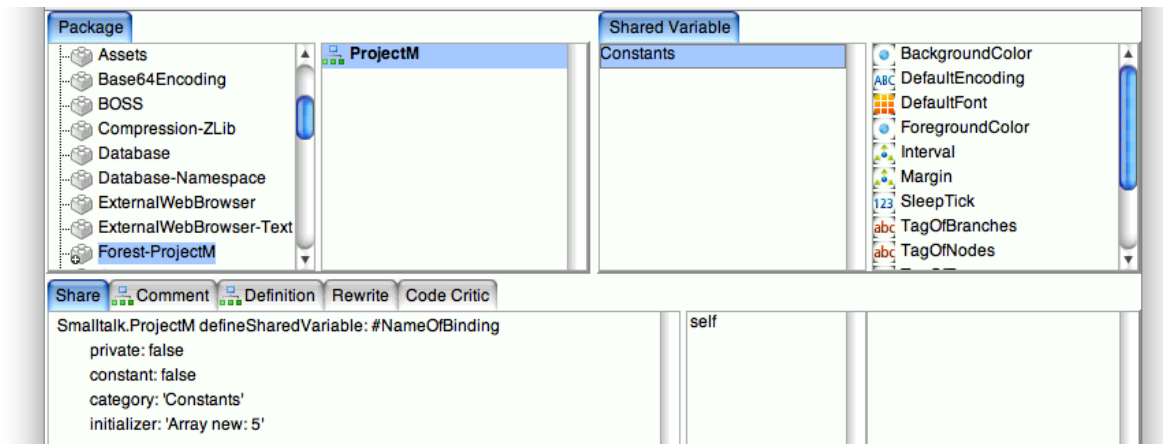
・ つぎ 2_NameSpace

```
(aNameSpace
  defineSharedVariable: #BackgroundColor
  private: false
  constant: false
  category: 'Constants'
  initializer: 'ColorValue white') initialize.
```

などはつまり、このネームスペースに共有変数を設定していつている。必要なグローバルなものを #define しているような気分。
 で、いろいろやって、さっき作った theBrowser に向かって、開け、と言う。（というか isOpen True なら raise せえ、False なら open せえ）
 (aWindow := theBrowser builder window) isOpen

```
ifTrue:
  [aWindow isCollapsed ifTrue: [aWindow expand].
  aWindow raise]
ifFalse: [theBrowser := Refactory.Browser.RefactoringBrowser open].
```

さて doit すると、以下のように Browser の ProjectM ネームスペースの SharedVariable にドカドカといろいろグローバルなものが定義された。



・ 次 3_Class

```
add: (Array with: #Example with: #[Core.Object] with: 'forest picture');
add: (Array with: #Model with: #[UI.Model] with: 'forest picture');
add: (Array with: #View with: #[UI.View] with: 'offset');
add: (Array
  with: #Controller
  with: #[UI.Controller]
  with: String new);
add: (Array with: #Forest with: #[Core.Object] with: 'nodes branches bounds');
add: (Array
  with: #Node
  with: #[Core.Object]
  with: 'name location extent status')
```

```

with: 'textAttributes');
add: (Array with: #Branch with: #{Core.Object} with: 'start end');

```

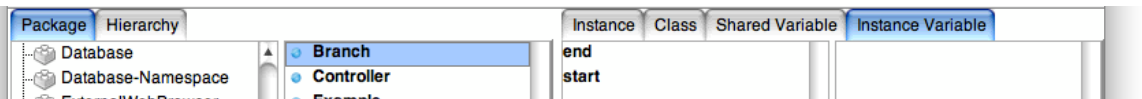
これらのクラスを作る。Array の三つある要素はそれぞれ前からクラス名、親クラス名、インスタンス変数名。
四つ目はクラスインスタンス変数名。

```

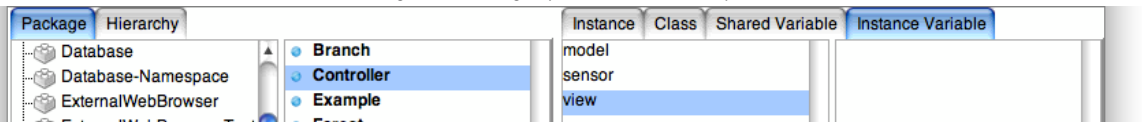
aClass := aNamespace
    defineClass: (anArray at: 1)
    superclass: (anArray at: 2)
    indexedType: #none
    private: false
    instanceVariableNames: (anArray at: 3)
    classInstanceVariableNames: (anArray size >= 4
        ifTrue: [anArray at: 4]
        ifFalse: [String new])
    imports: String new
    category: packageNameString.

```

例えば Branch クラスには指定したとおり start, end の二つがはいった。



ところで Controller クラスはインスタンス変数無し (String new は " (single quart x 2, not double quote!)) と言いたのと同じ)



と指定したのに model, sensor, view とできている。これはスーパークラスとして指定した UI.Controller のインスタンス変数を継承してできたもの。

・次 4_Methods

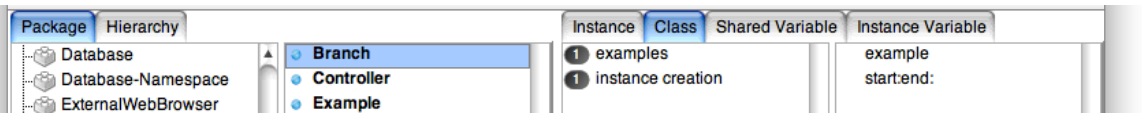
これでメソッドを追加。実はこのプログラムはかなり荒技。

```

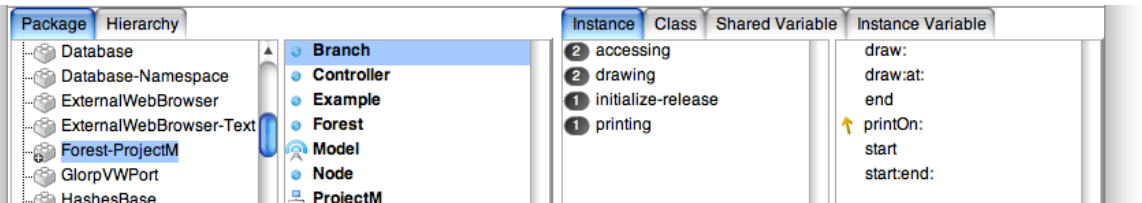
'<?xml version="1.0"?>
<st-source>
..... 略 .....
</st-source>
' readStream fileIn.

```

となっている。つまりこの文字列は実際に作った method を FileOut したソースコードをそのまま入れて、fileIn させた、という話。
結果を見ると、こんな感じ。まずはクラスメソッド。



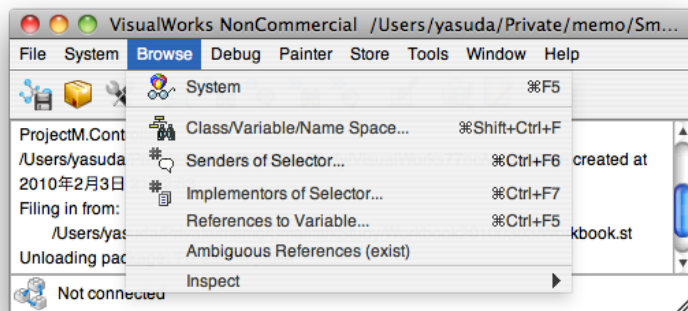
インスタンスメソッド。



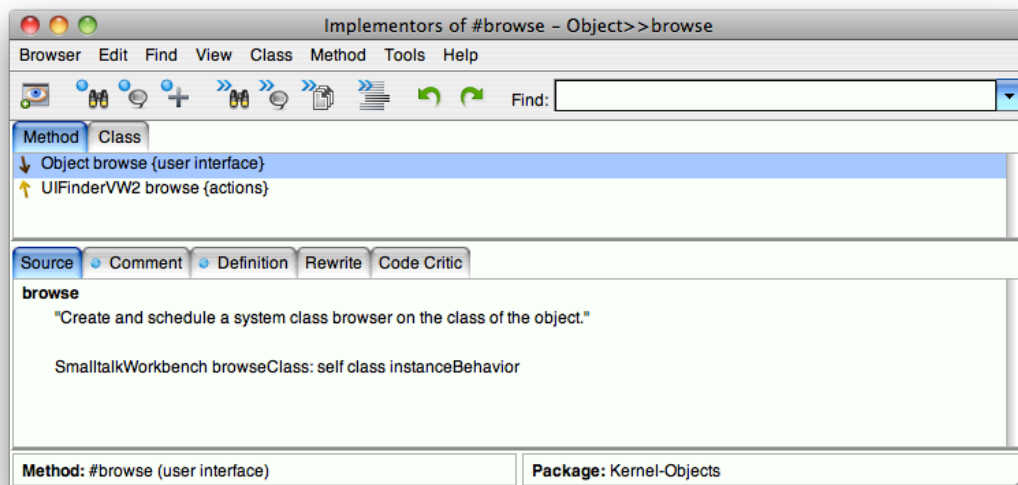
入っとる入っとる。

・次 5_Testing

ここではついでにリストメソッドブラウザー、というのが出てくる。



のImplementors of Selector（そのメソッドを実装している（セレクトがある）ものを探し出せ）をやると、問い合わせダイアログがでくるので、そこで例えば browse というメソッドを探しに行かせると、



こんな感じで出てくる。

さて Testing でやってること。

```
targetString := 'Example run: #tree.'
```

```
aMethodCollector := MethodCollector new.
```

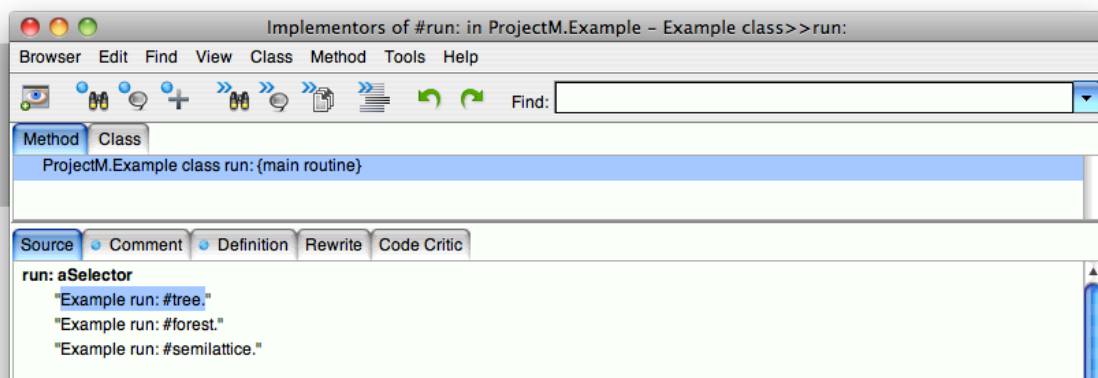
```
aFilter := (aMethodCollector searchClassHierarchy: ProjectM.Example)
           & (aMethodCollector implementorsOf: #run:).
```

```
aValue := aMethodCollector select: aFilter.
```

上のメソッドコレクターを new して、そのフィルターに対してこういう条件を設定するのだと指定して、コレクターに対してそいつでセレクトせえ、と言って、その結果 aValue を使って

```
aBrowser := Refactory.Browser.RefactoringBrowser
           openListBrowserOn: aValue
           label: aFilter displayString
           initialSelection: targetString.
```

ブラウザで出せ、と言ってみた。こうなる。



というわけで、ここでコンテキストメニューを出して、Dolt すればいいんだけど（そうすれば Example run: #tree が実行される）、その操作（コンテキストメニューを出す）を以下の記述でやろうとしている。（やらんでええんやけどね）

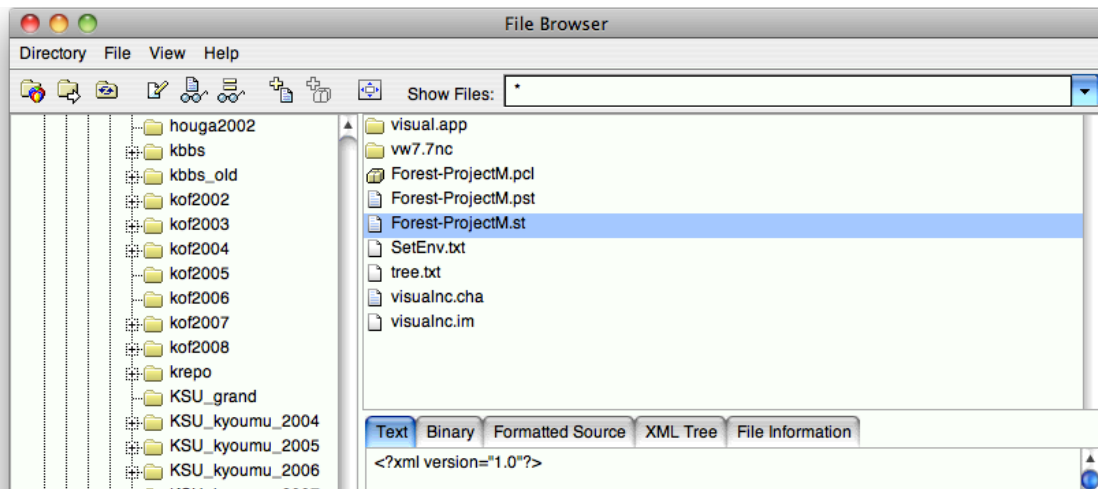
```

aMenu := aBrowser builder actionAt: #menuBar.
anItem := aMenu atNameKey: #Edit.
aMenu := anItem submenu.
anItem := aMenu atNameKey: #Do It'.
InputState default
    cursorPoint: aBrowser builder window displayBox center rounded.
aValue := aMenu startUp.
aValue = 0 ifTrue: [^nil].
aBlock := anItem value.
aValue := aBlock value.
^aValue

```

・さて Saving

やると保存して結果を開いて見せてくれる。こんな感じ。



・さて Parcel

これはバイナリオブジェクトを Parcel として Save する、というもの。

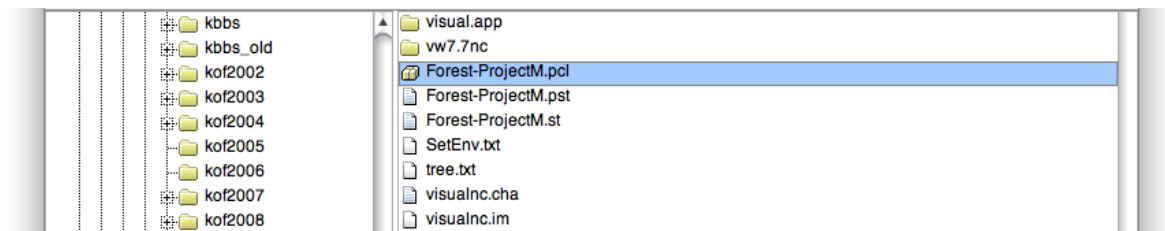
前回と違うところはここだけ。

```
aNameSpace dataBindings do: [:aBinding | aParcel addBinding: aBinding in: aNameSpace].
```

この dataBindings で取り出しているのは共有変数。（共有変数でのはこの dataBindings で取り出せる、というはなし）

これを一つずつ取り出して aParcel に addBinding で追加しているわけ。

Name空間、クラス、共有変数を入れたら良いはずなんだけどね。



できたできた。pcl, pst ができた。

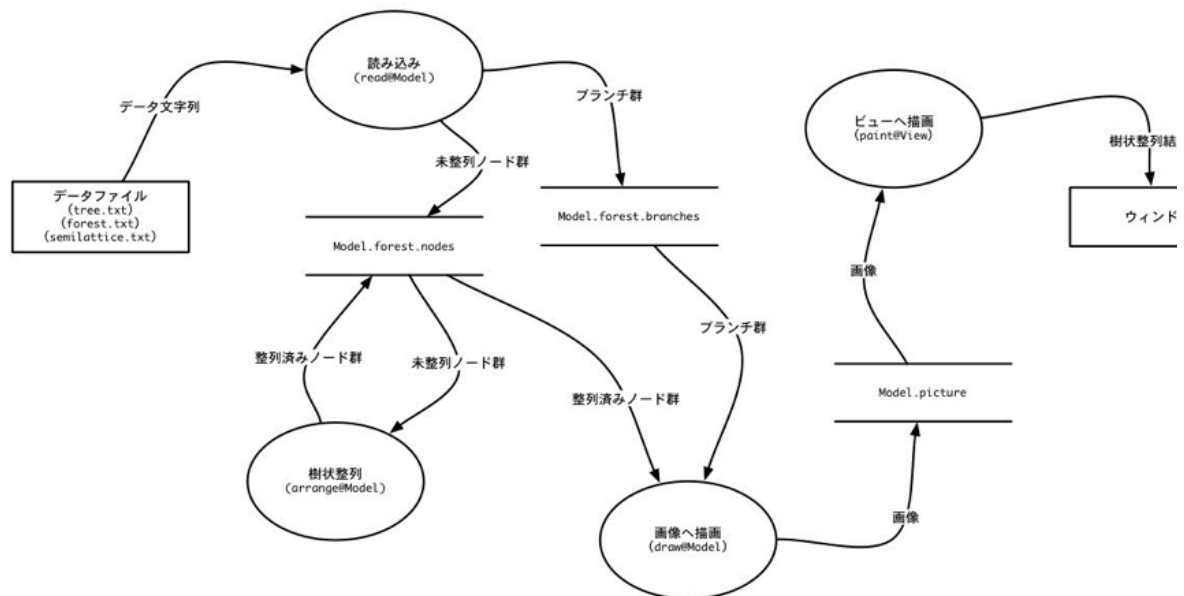
試しに image save せずに exit する形で Visualworks を終了。

この状態でファイルブラウザで上の Parcel ファイルに指定して、この pcl ファイルを load させると、同じものが復元されるはず。

= ふう。ちと休憩。

ProjectM/BasicDesign/images/Forest_BD03.jpg

（これ DFD という？）



結局データファイルを読み込んでどないかしてビューを通して画像になってウィンドウまで到達するかを示す図になる。
 実際にはノードの整列のためにはブランチの情報を得ないといけないので、branches から樹状整列のマルまでは何かデータを参照するという枝のようなものがあるはず。

UML 図はついてない。

Forest の中に Branch, Node の Collection を保持することにする。それ故に全てのインテリジェンスを Forest の中に押し込むことになる。

Branch は start, end インスタンス変数をもっている。これを指定して new する。
 あと draw する、くらい。ほとんど dumb な感じ。

Node も恐らく非常に dumb な格好になっているはず。