my study note
coding mac Desktop Application:sKakeibo.app
with SWIFT4

梅屋萬年堂

2018/11 ver 1.4

一年前まで mac のデスクトップ・ アプリケーションは XOJO を使って作っていました。objectiveC になかなか手を出せなかったからです。しかし、XOJO は有料で毎年の更新はできそうになく、SWIFT はどうなのかな？と思っていましたが、SWIFT4 になってそろそろ「安定版」になったようなので、使ってみることにしました。まずは参考書を探す事に。すると SWIFT の解説本はそのほとんどが IOS アプリのもので macOS のものは見当たりませんでした。なんとか見つけた、デストップ・アプリを扱うものが次の 3 冊でした。

- 「Swift Mac アプリ開発入門編」（中山著、カットシステム）
  SWIFT3 でのものですが、色々な例があり参考になります。ただ、私のような SWIFT 入門者にとっては”何を何から”手をつけてよいのか？わかりませんでした。また、これからは Storyboards を使っての開発が中心になるようなのですが、その説明が少ししかありませんでした。
- 「Programming Swift! Mac Apps1 Swift3 edition」(by Nick Smith, AppSmith Books)
  これが Swift 入門には最適でした。英語で書かれていることと Swift3(Xcode8) 対応であったのが残念ですがとても参考になりました。
- 「Hackin with macOS, Learn to make desktop apps with real-world Swift projects」(by Paul Hudson) これが最新のものようで Swift4(Xcode9) に対応しています。Smith 氏のものに書かれていないところとかを読むことがでします。まだ、全部は読んでなく、必要なときに参照するものにしようと思っています。

このノートでは、主に Smith 氏の本を学習しながら、さらにネット上の記述[*1]などを参考にして macOS のデストップ・アプリとして、家計簿アプリ sKakeibo.app 作る時に必要となったコードを書いていきます。Swift が初めてという場合は、何か本（私は「Swift 実践入門」、「詳説 Swift」）を読みながらが良いでしょう。

私の環境は、iMac(27-inch,2017)、macOS High Sierra(10.13.6)、Xcode9.4.1、SWIFT4.1[*2]です。SQLite データベースファイルのために、DB Browser for SQLite(3.10.1) を、このノートは TexShop で書き、画面のスナップショット作成加工に、Skitch(2.8.2) を使用しました。以下に、このノートで記述される内容について列記します。

第 1 節　Xcode と基本的なコントロール。このへんのことは、ネットなどでたくさん書かれているので、私がやったことの記録のようなものです。

第 2 節　Table View のあれこれを書きました。わからないところが多くあったのですが、なんとか”動かす”ところまではできたと思っています。似たような処理になる Combo Box についても触れておきます。

第 3 節　SQLite の使用法を書きました。いくつか便利なサードパーティのライブラリーがあるようですが、ここでは組み込みのものだけで処理しました。

第 4 節　主となるウィンドウ以外にウィンドウを表示しデータをやりとりする方法について書いてみました。

第 5 節　メニューバーとツールバーについてです。

第 6 節　印刷処理。これは、わからないところが多いのですが、少しだけ書きます。

[*1] 特に、raywenderlich.com:macOS Development Tutorials

[*2] 2018/9/23 現在、Xcode10.0, Swift4.2 になっていました。いろいろと変わっているようですが、特にオブジェクト・ライブラリーの表示が大きく変わり「`Cmd + shift + l`」キーを使うようです。

目次

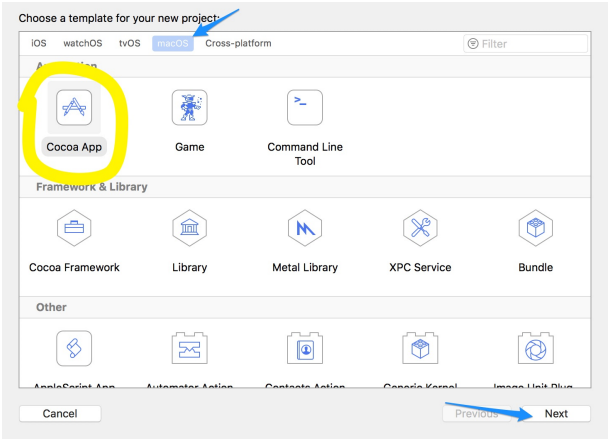
1　Xcode と基本的なコントロール

まずは、Xcode が必要なので、App Store からダウンロードしてインストールします。（無料です）

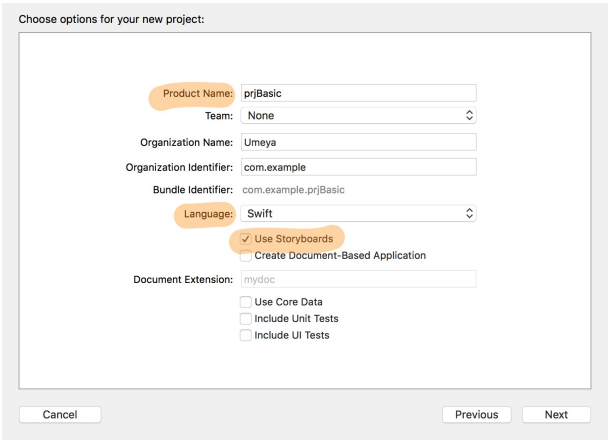
1.1　project を作る

Xcode を起動します。下図の矢印で示した Create a new Xcode project をクリックします。

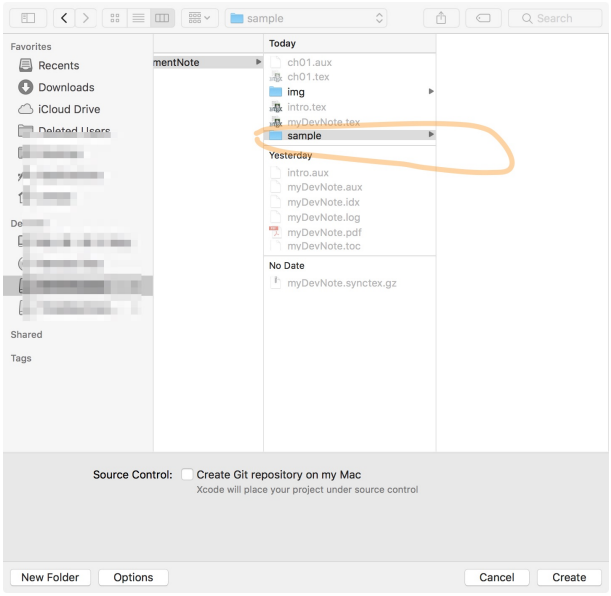
Choose a template for your new project の画面では、macOS をクリックして Cocopa App が撰択されていることを確認します。画面右下の Next をクリックします。



Choose options for your new project の画面で、Product Name にプロジェクト名を入れる。ここでは prjBasic として、いくつかのテストや説明に使う予定です。何もしなければこれがビルドした時のアプリの名前になります。Language は Swift で、その下の Use Storyboards にチェックを入れます。Team は None で、Organization identifier はそれを持っていないのであれば適当に自分の名前かサイト名の逆順とかでどうでしょう。その他の Document Extension などはわかりません？ Next をクリックし、



次の画面でプロジェクトファイルの保存場所を指定します。ここでは sample ディレクトリを指定したので、ここに prjBasic という名前のフォルダーができ、そこにプロジェクトのファイルなどが全て保存されます。Localization の設定などあるようですが省略して、これで準備は終了です。

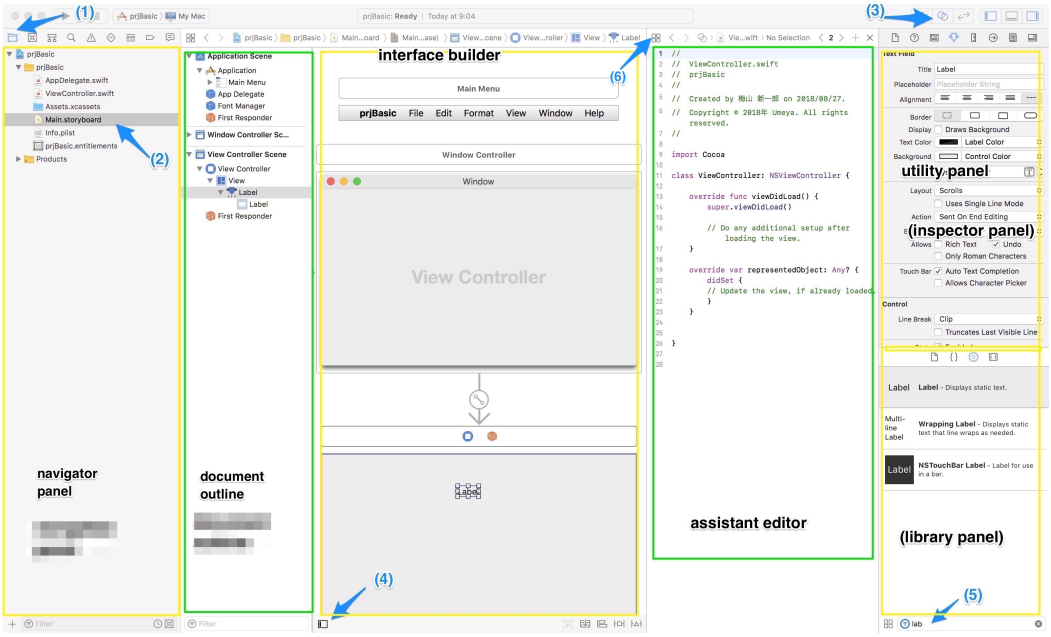


1.2 Xcode IDE を使う

prjBasic を保存したところで Xcode を終了させていたのなら、Xcode を起動すると画面右上に作成したものが表示されているので、prjBasic を開いてください。プロジェクトの設定画面になりますから、左上のフォルダアイコン矢印（１）をクリックします。これで画面左に navigator panel(プロジェクト内のファイルツリー) が表示されます。下の図は、矢印（２）の Main.storyboard をクリックして　document outline（ドキュメント構成図） と　interface builder（インターフェイス・ビルダー）を表示させ、矢印（３）で assistant editor(アシスタント・エディタ） を表示しています。その右の utility panel（ユーティリティ）は上部が inspector panel（インスペクター）で下が library panel（ライブラリー）になります。このライブラリーからコントロールなどをインターフェイス・ビルダー画面上の必要なところへドラッグして配置します。

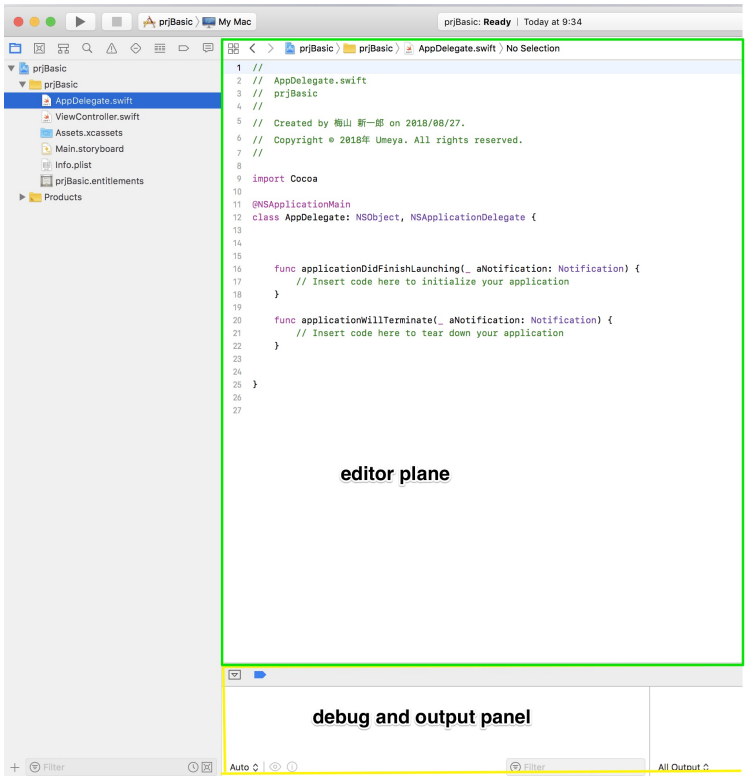
この図では Label を配置したところです。なお、矢印（４）はドキュメント構成図の表示・非表示の「切り替えに使え、矢印（５）はコントロールの検索に使えます。

インターフェイス・ビルダーには上下に２個のウィンドウがあります。上が Window Controller でその上の Main Menu でのメニュー設定やツールバー設定で使われます。その他のコントロールは下の View Controller に配置されます。



上の図で AppDelegate.swift をクリックして editor plane を表示させ、view メニューから Debag Area を表示させたものが下の図です。ここで表示したコードなどはアシスタント・エディタに表示させることができます。いくつか方法があるようですが、矢印（６）のマークをクリックすると表示したいものを選択指定できます。

このへんの Xcode の使い方や基本的なコントロールの使い方などは、ネットにあるものほうが丁寧に書かれていると思いますので、次は sKakeibo で使ったコントロールを書いていきます。

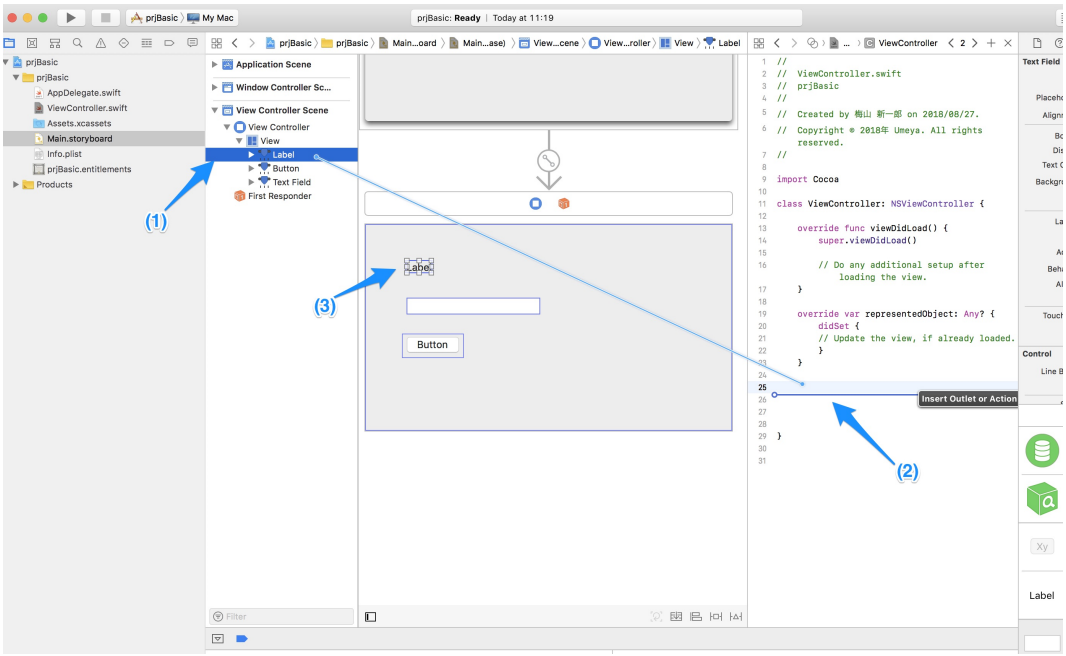

prjBasic 〉My Mac
prjBasic: Ready | Today at 9:34
prjBasic
prjBasic
AppDelegate.swift
ViewController.swift
Assets.xcassets
Main.storyboard
Info.plist
prjBasic.entitlements
Products
prjBasic 〉 prjBasic 〉 AppDelegate.swift 〉 No Selection
editor plane
debug and output panel
Auto
Filter
All Output

1.3　いくつかの基本的なコントロール

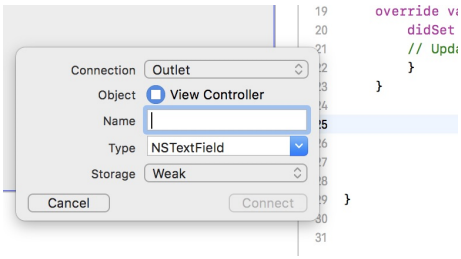
1.3.1　Label,Text Field,Button

prjBasic にいくつかのコントロールを配置して使ってみます。sKakeibo で使ったものは、Label、TextField、Button、DatePicker、それと TableView、ComboBox、MenuBar と ToolBar、ViewController などです。table view 以下のものはあとで書いていくことにします。

まずは、prjBasic で document plane とインターフェイス・ビルダーを表示させ、アシスタント・エディタで ViewContoroller.swift を表示してください。ライブラリ（library plane）から Label と Text Field それと Button を View Controller Scene（下にある Main View Controller ウィンドウ、上は Window Controller Scene）に配置してください。



図の矢印（１）を Cntl+Drag してアシスタント編集エリア（２）にもっていきます。insert Outlet or　Action のメッセージが出ます。下の図のボックスの Name に適当な名前を入れます。ここでは lblOutput として、ボックスの conect ボタンを押します。これで ViewController.swift のクラス内に次のコードが挿入されます。




```
@IBOutlet weak var lblOutput: NSTextField!
```

アウトレット（outlet）はオブジェクトの参照で、このクラス (class ViewController) のプロパティになります。

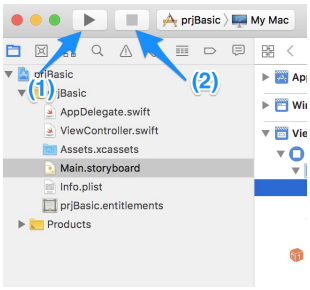
同じようにして Text Field のアウトレットも、名前を txtInput にして作っておきます。では、Button をコントロール・ドラッグして先のアウトレットの下あたりに置いてみます。今度は表示されたボックス内の connection で Outlet ではなく Action を選びます。名前 (Name）は btnTest とでもしておきます。これで connect ボタンを押すと、次のコードが挿入されます。

```
@IBAction func btnTest(_ sender: Any) {
    }
```

今度は関数が作られ、このボタンを押した時の処理をこの関数の中に書いていきます。txtInput に入力された文字を lblOutput にコピーして表示させてみます。

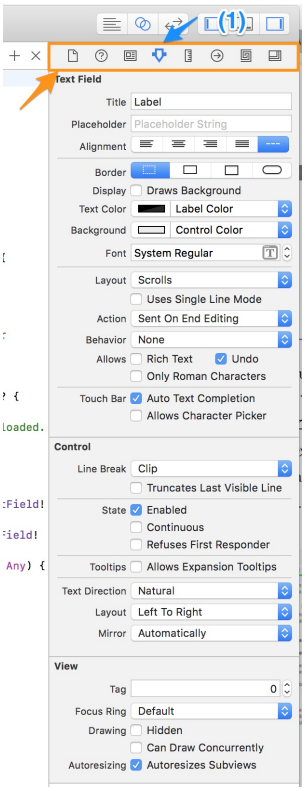
```
@IBAction func btnTest(_ sender: Any) {
        lblOutput.stringValue = txtInput.stringValue
    }
```

このコードを書いて、画面左上の矢印（１）の実行ボタンをクリックします。プログラムがコンパイル・ビルドされアプリが実行されます。その右の矢印（２）はアプリを停止します。この実行中のウィンドウのクローズボタン（右上の赤いボタン）をクリックした場合にはアプリは停止していません。このウィンドウをクローズすることで停止させるには、AppDelegate.swift の中に、次のコードを書き入れます。何かの時に使ってください。




```
func applicationShouldTerminateAfterLastWindowClosed(_ sender: NSApplication)
      -> Bool {  return true  }
```

これらのコントロールのタイトル、サイズなどの属性はコードを使わないで、インスペクターの中にあるものを使って設定できます。ドキュメント構成図の lblOutput を撰択して画面右のインスペクタを見てください。図のオレンジの矢印で指し示された長方形内のアイコンは、マウスを乗せると吹き出しが表示されるので見てください。左からファイル・インスペクタ、クイックヘルプ・インスペクタ、アイデンティティ・インスペクタ、属性・インスペクタ、サイズ・インスペクタ、接続・インスペクタ、ビルディング・インスペクタ、ビュー効果・インスペクタとなりますが、この中で使う機会があるのはアイデンティティ・インスペクタ、属性・インスペクタ、サイズ・インスペクタ、接続・インスペクタの４個で、特に矢印（１）の属性インスペクタを使います。いま表示されている lblInput の Title を何か適当なものに変えてください。Button のタイトルも変えてみましょう。



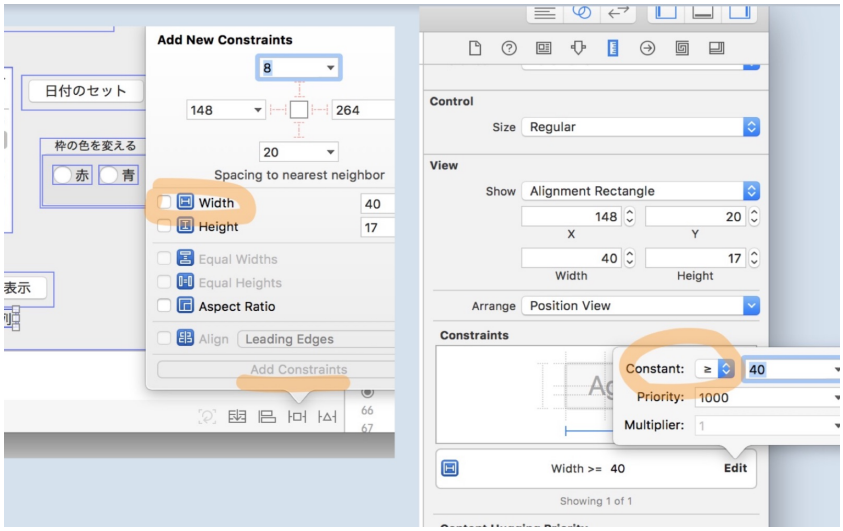
1.3.2　テキスト幅を超えた文字列の表示

ラベルなどにコードから文字列を表示した場合、設定した幅を超えた文字列は表示されません。ボタンとラベルを一つずつ配置してください。タイトルは適当に、ボタンの Action 接続、ラベルの Outelet 接続は次のコードのようにしてください。ラベルの幅は４０位、短めにしておきます。実行してみて１ください。ボタンを押すたびに長い文字列と短い文字列を交互にラベルに表示します。長い文字列は途中から切れています。Text Field の場合にはフィールド内にカーソルを置き移動させて見ることはできますが、ラベルはできません。


```
@IBOutlet weak var lblLongShortText: NSTextField!
var swLong = true
@IBAction func btnDisplayString(_ sender: NSButton) {
        if swLong {
            lblLongShortText.stringValue = "Long text Long text Long text Long text "
        }else{
            lblLongShortText.stringValue = "short"
        }
        swLong = !swLong
    }
```

上の図の矢印（１）をみてください。この近辺にあるアイコンは AutoLayout の時に使うものですが、ここではテキスト表示幅の設定のところだけにします。Xcode 上でカーソルをそこに重ねると「Add New Constraints」と表示されます。

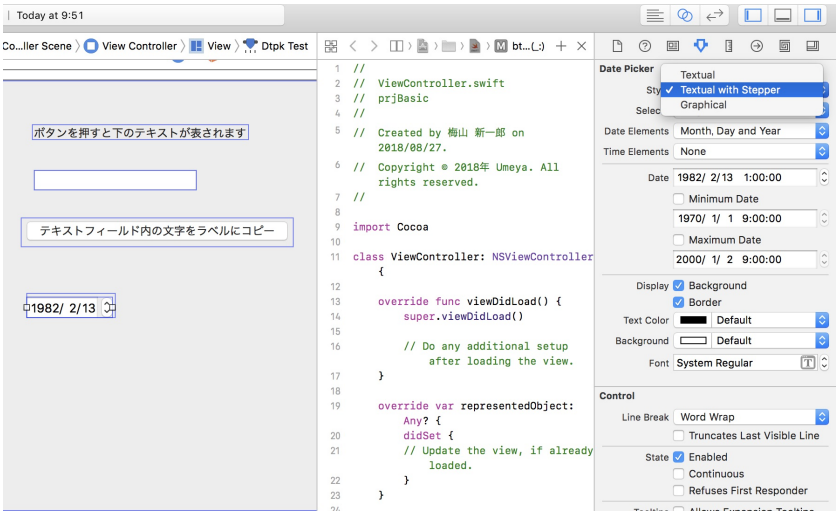
これを lblLongShortText に適応させます。lblLongShortText を選択した状態で矢印（１）をクリックします。次の図の「Add New Constraints」のメニューが表示されるので、Width にテェックを入れ Add 1 Constraints をクリックします。lblLongShortTex の周りに赤いラインが出ますが無視します。インスペクタのサイズのところを見ると、Contraints があります。Edit をクリックすると図の吹き出しが出ます。ここで Constants の記号を等号から以上の不等号に変えます。では、実行してみます。どうでしょうか、長い文字列も表示されているはずです。


日付のセット
枠の色を変える
赤
青
表示
Add New Constraints
8
148
264
20
Spacing to nearest neighbor
Width
Height
40
17
Equal Widths
Equal Heights
Aspect Ratio
Align
Leading Edges
Add Constraints
Control
Size
Regular
View
Show
Alignment Rectangle
148
20
X
Y
40
17
Width
Height
Arrange
Position View
Constraints
Constant:
≥
40
Priority:
1000
Multiplier:
1
Width >= 40
Edit
Showing 1 of 1

1.3.3 DatePicker

View Controller Scene のウィンドウサイズを大きくします。ウィンドウの辺付近にカーソルを持っていくとサイズ変更のアイコンが表示されます。ドラッグしてサイズを変えます。ライブラリで　DatePicker を探します。順に見て探すより、右下の検索欄に date とでも打ち込めば出てきます。DatePicker をドラッグしてきます。このアウトレットを作ります。名前は、dtpkTest としておきます。

この属性インスペクタを見てください。図は Style にある 3 つのスタイルをドロップしたものです。現在のスタイルは Textual with Stepper となっています。図のように年／月／日の形式で表示され右端に日付変更のステッパーがついたものです。Textual はステッパーなしのもの、Graphical は月のカレンダーを表示したものです。スタイルをこのグラフィックに変えてみます。



ViewContoroller.swift を表示してください。下のように viewDidLoad 関数にコードを追加してください。viewDidLoad 関数には、そこにあるコメントの通り、このアプリの画面が表示された直後に必要な初期処理などを記述します。実行ボタンをクリックしてください。カレンダーが今日の日付になりました。

```
override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
        dtpkTest.locale = Locale(identifier: "ja_JP")  //-----(1)
        dtpkTest.dateValue = Date()                          //.....(2)
    }
```

コード (1) はカレンダーを日本語表記にしています。これは、プロジェクトの info 内の Localization の設定でもできるようです。(2) はカレンダーに今日の日付をセットしています。カレンダーから日付を取得してみます。btnTest のタイトルを「カレンダーの日付をラベルにコピー」と変えて、btnTest 関数のコードを下のようにします。

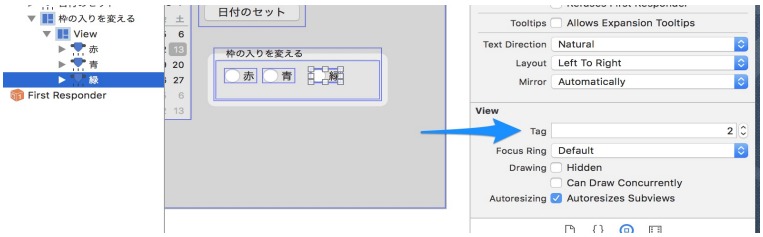
```
    @IBAction func btnTest(_ sender: Any) {
        //lblOutput.stringValue = txtInput.stringValue
```

```
        let DF = DateFormatter()
        DF.dateFormat = "YYYY-MM-dd"
        DF.timeZone = NSTimeZone.local
        let YMD = DF.string(from: dtpkTest.dateValue)
        lblOutput.stringValue = YMD
        let Y_M_D = YMD.split(separator: "-")
        txtInput.stringValue = "今日は" + String(Y_M_D[0]) + "年の"
        + String(Y_M_D[1]) + "月" + Strig(Y_M_D[2]) + "日です"
    }
```

では、実行してみてください。カレンダーで日付を変えボタンを押すと取得した日付が表示されています。逆にカレンダーの日付をテキストフィールド内の yyyy-MM-dd の形式の文字列で設定してみましょう。ボタンを追加し、btnTest2 の名前で Action 接続しておきます。その関数の中に次のようにコードを入れます。

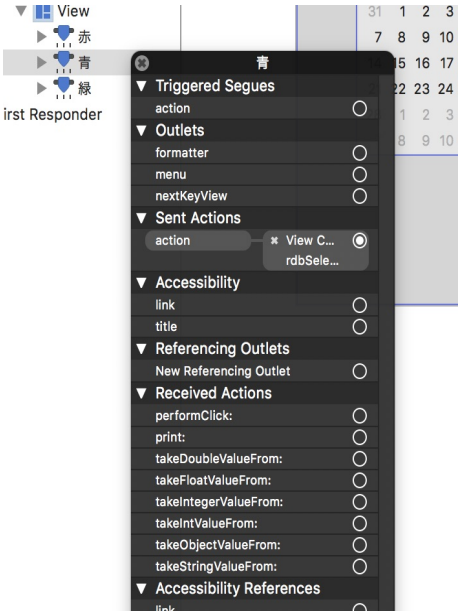
```
@IBAction func btnTest2(_ sender: Any) {
        let d = txtInput.stringValue
        let df = DateFormatter()
        df.locale = Locale(identifier: "ja_JP")
        df.dateFormat = "yyyy-MM-dd"
        dtpkTest.dateValue = df.date(from: d)!
    }
```

1.3.4 Radio Button、Box



ラジオ・ボタンは複数個のものを一つのグループにして使います。グループは、グループ内のボタンが同じ View(または、サブビュー)内にあり、それらのボタンが一つの Action メソッドに接続されていることが条件です。prjBasic の View Controller Scene の View 上にラジオ・ボタンを配置してもよいのですが、Box というサブ・ビューのなかに配置してみます。ライブラリから Box を持ってきて、Title を枠の「色を変える」としてください。次に Radio Button を 3 個、Box 内に配置してください。ボタンのタイトルを赤、青、緑にして、さらに属性インスペクターの下の方にある Tag 番号を 0、1、2 にします。タイトルが緑のラジオ・ボタンのタグ番号 (Tag) を 2 にしたものです。

次に赤のラジオ･ボタンの Action 接続を、名前　rdbSelected で作ります。ドキュメント構成図の青のボタンからドラッグ (Cntrol+Drog) して、さきほど作った rdbSelected にカーソルを持ってくると関数が水色に包まれ connection action が表示されます。そこでマウスを放します。青のラジオ･ボタンを右クリックしてください。表示された中に Sent Action で rdbSele... があります。同じ Action メソッドに接続されているのです。緑のボタンにも同様の接続をしてください。rdbSelected 関数のコードを書いてください。




```
@IBAction func rdbSelected(_ sender: NSButton) {
        txtInput.wantsLayer = true
        txtInput.layer?.borderWidth  = 2.0
        switch sender.tag{
        case 0:
            txtInput.layer?.borderColor = NSColor.red.cgColor
        case 1:
            txtInput.layer?.borderColor = NSColor.blue.cgColor
        case 2:
            txtInput.layer?.borderColor = NSColor.green.cgColor

        default:
            return
        }
    }
```

ラジオ・ボタンが 3 個の中の一つだけ撰択され、txtInput の枠の色が変わっていきます。別のラジオ・ボタンのグループが必要な時には Box でサブビューを作り、そこに配置します。

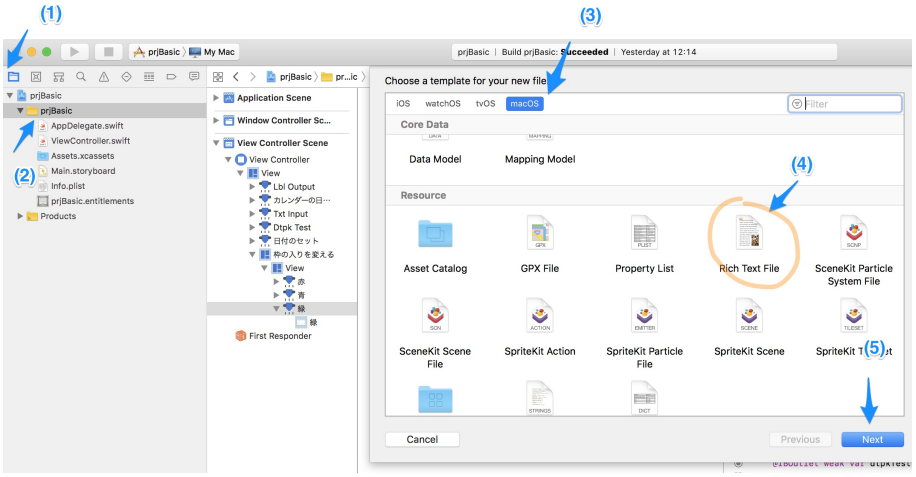
1.4　About Box と AppIcon

コンパイル・ビルドされたアプリは、プロジェクトのでぃれくとりの中にあります。prjBasic では、

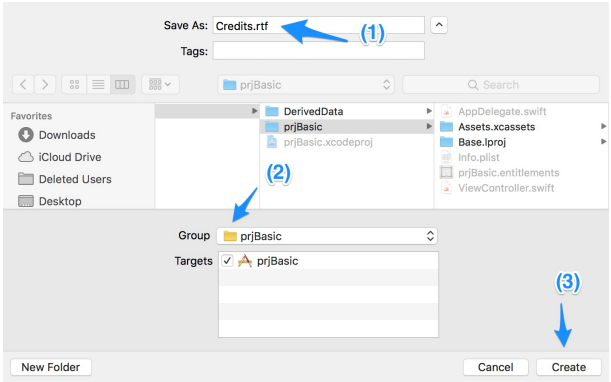
```
/prjBasic/DerivedData/Data/prjBasic/Products/Debug/prjBasic.app
```

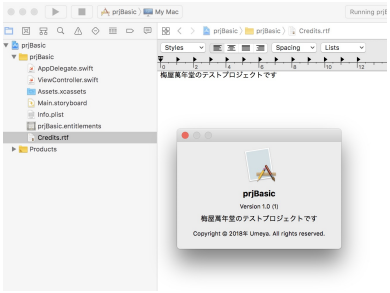
の prjBasic.app になります。これが実行ファイルになります。

ここで２つのことを追加しておきます。アプリを実行させるとメニュー prjBasic で about prjBasic のサブメニューがあり、これをクリックするとこのプログラムの説明が出るようにアプリは作られます。ここで表示するメッセージを追加する方法についてが一つ目、その表示画面でも現れますが、アプリのアイコンの設定が２つ目です。まず about prjBasic について。これは、プロジェクトの中にメッセージを書き込んだ Credits.rtf ファイルを入れるだけです。ナビゲータ領域 (navigator plane) で矢印（１）のフォルダーアイコンをクリックしプロジェクト内のファイルを表示させます。矢印（２）の黄色の prjBasic を右クリックして、表示されるメニューの中から New File... をクリック。Choose a template for your new file:で矢印（３）の masOS を確認し（指定して）矢印（４）の Rich Text File を選択し、矢印（５）の Next をクリック。



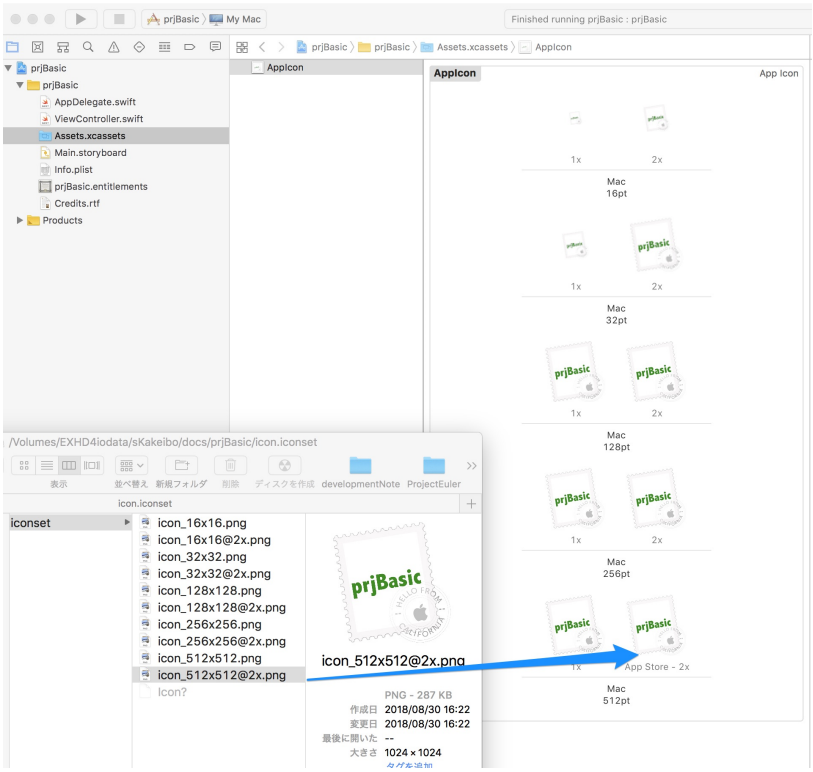
図のファイル保存画面で、矢印（１）にファイル名 Credits.rtf を指定し、Group が黄色のフォルダ prjBasic になっていることを確認してください。その欄の右端のピッカーを押せば青と黄色の２個が表示されるはずです。矢印（３）の Create をクリックしてナビゲーター領域をみてください。Credits.rtf があります。





これをクリックしてエディタ領域で About Box で表示することを書き込みます。外部のエディタで作成してこのプロジェクト内に保存しても同じです。書き込んだものが About prjBasic で表示されます。図は Credits.rtf に書き込みをしたのち、アプリを実行しメニューで About prjBasic を表示させたものです。

次に、AppIcon です。プロジェクトの Assets.xcassetss をクリックして下さい。AppIcon の領域に Mac16pt、32pt、128pt、256pt、512pt とそれぞれの X2 のアイコンが必要となっています。ここに用意したアイコンファイルをドラッグ・ドロップします。ここでは、Image2Icon.app を使って作った iconset を使います。図は `icon_512x512@x2.png` をドロップし終わった状態です。これで設定したアイコンが表示されま

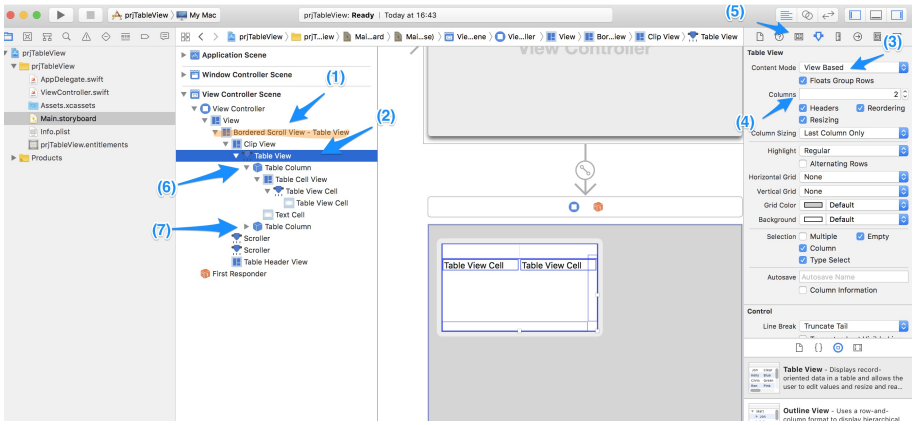


2　Table View

正直なところ、この Table View については、よくわからなくて、なんとか必要な機能だけは実装できたのかな？、という状態です。私がやってきたことを順に書いていきます。参考になれば幸いです。

2.1　テーブル・ビューの準備

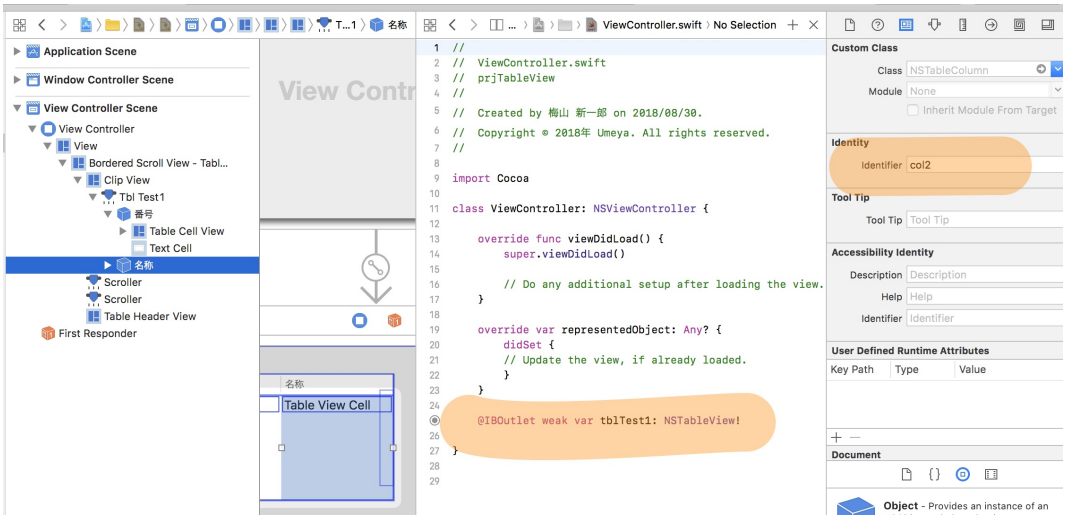
prjTableView という名前で新たに macOS のプロジェクトファイルを作ります。ナビゲータ領域の Main.storyboard をクリックして、インターフェイス・ビルダーを表示。ライブラリから Tavle View を選んで適当な位置に配置して下さい。



図は Table View 配置後、ドキュメント構成図を広げ、矢印（１）のドロップ・マークをクリックして、最下位の Table View Cell までを表示させ、矢印（２）の Table View をクリックして画面右上の属性インスペクターを表示させたところです。矢印（３）の Content Mode が View Based になっています。Cell Based もありますが、View Based が標準[*3]ようなのでこのままにします。矢印（４）でテーブルの列数が指定できます。その下の方にある Alternating Rows にチェックを入れると、行が縞模様にになります。

アシスタント・エディタに ViewController.swift を表示し、そこへ矢印（２）をコントロール・ドラッグして tblTest1 の名前で、outlet 接続します。２列のままにして、列のヘッダーと名前 (identifier) を決めます。第１列目は、上の図の（６）クリックして Title の欄に「番号」とでもタイトルを、矢印（５）をクリックしアイデンティティ・インスペクタを表示し identifier の欄に名前「col1」をつけてみます。矢印（７）で第２列目を指定し、第一列と同様に、タイトルは「名称」、identifier は「col2」にします。

*3 こちらの方が良いのではと思っているだけです？



2.2　テーブル・ビューのプロトコル

データを取り入れることとテーブルへの行・列の表示との２つのプロトコルを Table View は持っています。このプロトコル NSTableViewDataSource と NSTableViewDelegate を ViewController クラスに下のように付け加えます。テーブル・ビューのデータ・ソースとデリゲート[*4]として ViewController クラス自身を割り当てます。これは以下のコードのように viewDidLoad 関数の中に書きます。

```
import Cocoa

class ViewController: NSViewController, NSTableViewDataSource, NSTableViewDelegate{

    override func viewDidLoad() {
        super.viewDidLoad()

        tblTest1.dataSource = self
        tblTest1.delegate = self
    }
```

このままで実行してみて下さい。ウィンドウにからの表が表示され、デバッグ領域に以下のようなメッセージが出ます。

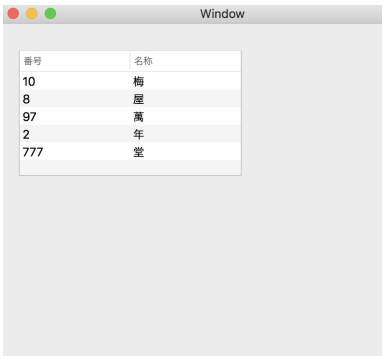
```
2018-09-03 17:52:37.389472+0900 prjTableView[1385:295031] *** Illegal NSTableView data source
(<prjTableView.ViewController: 0x6000000c2990>).
  Must implement numberOfRowsInTableView: and tableView:objectValueForTableColumn:row:
```

ここで示されている２つの関数を書かなければなりません。では、以下のコードを ViewController クラスに書き込みます。

[*4] delegate:委譲。クラスからクラスに処理を任せる。

```
func numberOfRows(in tableView: NSTableView) -> Int {
        return 4
    }

func tableView(_ tableView: NSTableView, viewFor tableColumn: NSTableColumn?
             , row: Int) -> NSView? {
        guard let vw = tableView.makeView(withIdentifier: tableColumn!.identifier
             , owner: self) as? NSTableCellView else {return nil}

        if tableColumn?.identifier.rawValue == "col1"{
            vw.textField?.stringValue = String(row) + "行  " + "１列"        }
        else if tableColumn?.identifier.rawValue == "col2"{
            vw.textField?.stringValue = String(row) + "行  " + "２列"        }
        else{
            return nil
        }

        return vw
    }
```

関数 numberOfRows(はテーブル・ビューで取り扱われるデータの行数を返しています。関数 tableView は Xcode の自動補完で引数のタイプによりたくさん出てきますが、このコード[*5]のものを選んで下さい。変数 vw は各列の有効な NSTableCellView がセットされます。次に、列につけた identifier で列を指定し、vw の内容を（その row 行の）列ごとにデータをセットします。実行してみて下さい。各コードの役割がわかるはずです。詳しくは、P.Hudson 氏のテキストを読んで下さい。



何かテストデータを用意してテーブルに表示してみましょう。testData1 は、タプル (no: Int, name:String) を要素にもつ配列にし関数 numberOfRows では、その要素数が表示する行数になるので、その数を返します。関数 tableView では各列ごとに、第 1 列 (col1) では配列要素タプルの no を、第 2 列（col2) には name をテーブルの列のテキスト・フィールドに代入しています。（下のコードを参照して下さい）このように testData1 のような変数に表示させたいデータを入れておけば、それが表示されます。

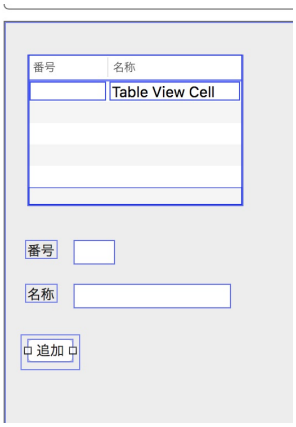
[*5] viewFor method

```
import Cocoa

class ViewController: NSViewController, NSTableViewDataSource, NSTableViewDelegate{

    var testData1:[(no: Int, name:String)] = [(10,"梅"), (8,"屋"), (97,"萬")
                              , (2,"年"), (777,"堂")]

    ........

    @IBOutlet weak var tblTest1: NSTableView!

    func numberOfRows(in tableView: NSTableView) -> Int {
        return testData1.count
    }

    func tableView(_ tableView: NSTableView, viewFor tableColumn: NSTableColumn?
                , row: Int) -> NSView? {
        guard let vw = tableView.makeView(withIdentifier: tableColumn!.identifier
                    , owner: self) as? NSTableCellView else {return nil}

        if tableColumn?.identifier.rawValue == "col1"{
            vw.textField?.integerValue = testData1[row].no
        }else if tableColumn?.identifier.rawValue == "col2"{
            vw.textField?.stringValue = testData1[row].name
        }else{   return nil  }
        return vw
    }
}
```

2.3 テーブルの行の追加、削除、編集

テーブルに行を追加するのは、testData1 のような変数にデータを追加し、テーブルをリロードするだけです。

実験してみます。図のように２つのラベルと２つのテキスト・フィールドを追加します。さらにボタンを一つ付け加えます。ラベルとボタンのタイトルは、「番号」、「名称」、「追加」にします。それぞれ Outlet と Action 接続しておきます。




```
@IBOutlet weak var txtNoTest1: NSTextField!
@IBOutlet weak var txtNameTest1: NSTextField!
@IBAction func btnAddTest1(_ sender: NSButton) {
    }
```

btnAddTest1 ボタンを押すと、txtNoTest の番号と txtNameTest1 の名称を表に追加するようにコードを準備します。

```
@IBAction func btnAddTest1(_ sender: NSButton) {
         //------------------------------------------------------ data check
        let fldNo = txtNoTest1.stringValue.trimmingCharacters(in: .whitespacesAndNewlines)
        if fldNo.isEmpty || !isStringContainsOnlyNumbers(string: fldNo){
            someMessageWithAlert("番号は数字です")
            return
        }
        let fldName = txtNameTest1.stringValue.trimmingCharacters(in: .whitespacesAndNewlines)
        if fldName.isEmpty{
            someMessageWithAlert("名称が空白です")
            return
        }
        //------------------------------------------------------

        testData1.append((Int(fldNo)!, fldName ))   //---(1)

        tblTest1.reloadData()                              //....(2)
    }

    //-------------for data check
      func someMessageWithAlert(_ message: String, _ textInfo:String = "information") {
        let alert = NSAlert()
        alert.messageText = message
```

```
        alert.informativeText = textInfo
        alert.runModal()
    }

    func isStringContainsOnlyNumbers(string: String) -> Bool {
        return string.trimmingCharacters(in: .decimalDigits).count <= 0
    }
```

data check はごちゃごちゃしてますが Alert を使用してメッセージを表示することと、数字だけのチェックの関数を入れているだけです。中心は（１）のデータの変数への追加と、（２）のテーブルのリロードだけです。実行しデータを追加してみて下さい。

では、テーブルの行を選択して（クリックして)、その行を削除する処理を追加してみます。タイトルを「削除」としたボタンを追加し、Action 接続します。そして、以下のようコードを準備します。下の関数 tableViewSelectionIsChanging を用意して下さい。

```
func tableViewSelectionIsChanging(_ notification: Notification) {
      let selRowNo = tblTest1.selectedRow //...(1)
      guard selRowNo != -1 else{return}       //...(2)
      txtNoTest1.integerValue = testData1[selRowNo].no  //...(3)
      txtNameTest1.stringValue = testData1[selRowNo].name  //...(4)
  }
```

この関数はテーブルの選択行が変わった時に起動されます。コードの（１）は選択行の番号[*6]を変数 selRowNo に代入しています。テーブルのヘッダーにクリックした時など選択されてない時には行番号が－１になるので、その下の guard 文（２）はそのテェックしています。（２）と（３）で選択秘された行のデータをテキスト・フィールドに表示しています。（３）と（４）で選択された行のデータをテキスト・フィールドに表示し削除の確認の用意をします。

関数 btnDeleteTest1 のコードについてです。(5）はデータ行が選択されてない時に－１になるのを使ってテェックしています。データ削除の確認を（6）で行います。追加の時と同様に、中心は（7）の変数からデータを削除し、(8）でそのデータを reload し、テーブルに再表示しています。

```
    @IBAction func btnDeleteTest1(_ sender: NSButton) {
        guard tblTest1.selectedRow != -1 else{          //...(5)
            someMessageWithAlert("データ行を選択してから削除してください。")
            return}
        guard yesNoMessageWithAlert("選択した行のデータを削除しますか？",
                      "削除する", "しない")else{return}         //...(6)
        testData1.remove(at: tblTest1.selectedRow)   //...(7)
        tblTest1.reloadData()            //...(8)
    }
```

[*6] 行番号は、0,1,2,...,testDat1.count-1

（6）で削除の確認のために関数 yesNoMessageWithAlert は Alert を使っています。この関数の中の１０００とか１００１については、また後で触れることにします。実行して下さい。

この削除の方法を使って、選択された行の編集をしてみます。タイトルが編集更新のボタンを追加し、btnEditTest1 の名前で Action 接続して下さい。関数 btnEditTest1 も関数 btnDeleteTest1 とほぼ同じです。（9）でデータチェック、（10）で確認、（11）でデータの更新、（12）でテーブルの再表示。

```
@IBAction func btnEditTest1(_ sender: NSButton) {
    guard tblTest1.selectedRow != -1 else{          //...(9)
        someMessageWithAlert("データ行を選択してから編集更新してください。")
        return}
    guard yesNoMessageWithAlert(
         "選択した行のデータをテキスト・フィールドのデータで更新しますか？"
          , "更新する", "しない")else{return}         //...(10)
    testData1[tblTest1.selectedRow] = (txtNoTest1.integerValue, txtNameTest1.stringValue)   //..
    tblTest1.reloadData()            //...(12)
}

//------------------------------------------------------ for btnDeleteTest1, btnEditTest1
func yesNoMessageWithAlert(_ message:String, _ yesTitle:String, _ noTitle:String) -> Bool{
    //Yes: return true, No:return false
    let alert = NSAlert()
    alert.messageText = message
    alert.addButton(withTitle: yesTitle)// = 1000
    alert.addButton(withTitle: noTitle)   // = 1001
    if alert.runModal().rawValue == 1000{
        return true
    }else{
        return false
    }
}
```

2.4　複数個ののテーブルの取り扱い

もう一つ TableView を追加してみます。tblTest2 の名前で Outlet 接続します。tblTest2 の属性インスペクタで列数を 1 にします。複数個のテーブルビューを区別するのは Tag を使用します。属性インスペクタの View のところに Tag があります。その値を 1 にして下さい。 2 個以上のテーブルビューを取り扱うときは、この Tag で区別します。テストデータ testData2 を辞書形式で準備し、tblTest2 のデータは vTestData2 に testData1 の no をキーにして取り出した値を使います。tblTest1 と同じに、tblTest2 の dataSource と delegate を自身 (ViewController クラス）にします。

```
var testData1:[(no: Int, name:String)] = [(10,"梅"), (8,"屋"), (97,"萬"), (2,"年"), (777,"
堂")]
var testData2 = [10:[1,2,3], 8:[11,2], 97:[7,6,9], 2:[20,21,22,23,24], 777:[1000]]
var vTestData2 = [Int]()

override func viewDidLoad() {
        super.viewDidLoad()
        ...
        tblTest2.dataSource = self
        tblTest2.delegate = self
    }

    @IBOutlet weak var tblTest1: NSTableView!
    @IBOutlet weak var tblTest2: NSTableView!
```

関数 numberOfRows では、この Tag で区別して、tblTest2 では vTestData2 の要素数を行数とします。関数 tableView でも同じで Tag で区別して、それぞれのテーブルの列のデータを行ごとにセットしています。

```
func numberOfRows(in tableView: NSTableView) -> Int {
        if tableView.tag == 0{ // tblTest1
            return testData1.count
        } else {                // tblTest2
            return vTestData2.count
        }

    }

func tableView(_ tableView: NSTableView, viewFor tableColumn: NSTableColumn?
                          , row: Int) -> NSView? {
        guard let vw = tableView.makeView(withIdentifier: tableColumn!.identifier,
                         owner: self) as? NSTableCellView else {return nil}
```

```
        if tableView.tag == 0 {  //tblTest1
            if tableColumn?.identifier.rawValue == "col1"{
                vw.textField?.integerValue = testData1[row].no
            }else if tableColumn?.identifier.rawValue == "col2"{
                vw.textField?.stringValue = testData1[row].name
            }else{   return nil }
        }else if tableView.tag == 1 {   //tblTest2
            if tableColumn?.identifier.rawValue == "colSuuryou"{
                vw.textField?.integerValue = vTestData2[row]
            }else{  return nil s }
        }
        return vw
    }
```

関数 tableViewSelectionIsChanging では引数 notification からの NSTableView を変数 tvw に代入しています。これを使って 2 つの TableView を区別し使っています。このテスト用のアプリでは tblTest1 で選択された番号 (no) の数量を tblTest2 に表示するようにしています。そのため、下のコード（1）で番号をキーにして testData2 からの数量を vTestData2 に代入し、tbTest2 を再表示 (reloadData()) しています。実行してみて下さい。tblTest1 テーブルの選択行を変えると tblTest2 には対応したデータが表示されます。

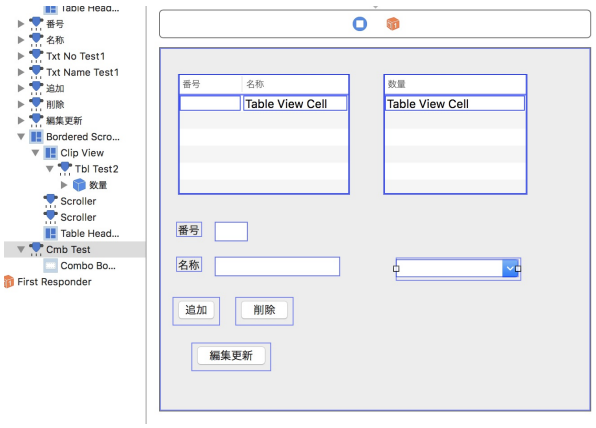
```
func tableViewSelectionIsChanging(_ notification: Notification) {
        guard let tvw = notification.object as? NSTableView else{return}
        let selRowNo = tvw.selectedRow
        guard selRowNo != -1 else {return}

        if tvw.tag == 0{           // tblTest
            txtNoTest1.integerValue = testData1[selRowNo].no
            txtNameTest1.stringValue = testData1[selRowNo].name
            vTestData2 = testData2[testData1[selRowNo].no]!  //...(1)
            tblTest2.reloadData()                                          //...(2)
        } else if tvw.tag == 1{    // tblTest2
            someMessageWithAlert("tblTest2 で、数量" + String(vTestData2[selRowNo])
               + "が選択されました。")
        }
    }
```

2.5 Combo Box

Combo Box は Table View と似たような処理をします。

Combo Box を追加し、cmbTest の名前で Outlet 接続します。

```
@IBOutlet weak var cmbTest: NSComboBox!
```



ViewController クラスに 2 個のプロトコルを下のコードのように追加します。関数 viewDidLoad に 3 行のコードを書きます。

```
class ViewController: NSViewController, NSTableViewDataSource, NSTableViewDelegate,
                                NSComboBoxDataSource, NSComboBoxDelegate{
    ...
override func viewDidLoad() {
        super.viewDidLoad()
        ....
        cmbTest.usesDataSource = true  //...(1)
        cmbTest.dataSource = self          //...(2)
        cmbTest.delegate = self             //...(3)
}
```

用意する関数は 2 個あり numberOfItems と comboBox です。tblTest2 で使用したデータを、この cmbTest でも使うことにして関数 numberOfItems では、その要素数をリストの行数としています。これで準備できました。実行してみて下さい。tblTest1 のテーブルビューで選択行を変えて cmbTest のリストが対応しているか確かめて下さい。cmbTest でリストの項目を選択するとメッセージが表示されます。これは、関数 comboBoxSelectionDidChange で処理しています。

```
 func numberOfItems(in comboBox: NSComboBox) -> Int {
        return vTestData2.count   }

func comboBox(_ comboBox: NSComboBox, objectValueForItemAt index: Int) -> Any? {
        return vTestData2[index]
    }

func comboBoxSelectionDidChange(_ notification: Notification) {
```

```
someMessageWithAlert("cmbTest で"
+ String(vTestData2[cmbTest.indexOfSelectedItem]) + " が選択されました。")      }
```

3　SQLite データベースと Swift

Swift で SQLite を取り扱うには、組み込みの SQLite3 ライブラリーが使えます。いくつかのサンプル[*7]を読めば利用できるようになります。SQLite そのもの解説書は少ないのですが、SQL 命令の解説書があれば、SQLite3 のサンプルを見ながら使うことができます。私が持っている解説書は次の 3 冊です。「SQLite 入門、西沢直木著、翔泳社」、「SQLite ポケットリファレンス、五十嵐貴之著、技術評論社」、「SQL 入門、中山清喬／飯田理恵子著、インプレスジャパン」

3.1　sqlite.org より

sKakeibo.app で使用した SQLite3 のコンストラクタ、デコンストラクタ、メソッドや定数などは以下の通りです。sqlite.org の Documentation[*8]からのものを列記します。

- `int sqlite3_open(`
  ```
    const char *filename,   /* Database filename (UTF-8) */
    sqlite3 **ppDb          /* OUT: SQLite db handle */
  );
  ```
  データベースファイルを開く（接続する）
- `int sqlite3_close(sqlite3*);`
  データベースファイルを閉じる（接続を解除する）
- `int sqlite3_prepare(`
  ```
    sqlite3 *db,            /* Database handle */
    const char *zSql,       /* SQL statement, UTF-8 encoded */
    int nByte,              /* Maximum length of zSql in bytes. */
    sqlite3_stmt **ppStmt,  /* OUT: Statement handle */
    const char **pzTail     /* OUT: Pointer to unused portion of zSql */
  );
  ```
  SQL 命令の実行前にバイトコードにコンパイルする
- `int sqlite3_step(sqlite3_stmt*);`
  prepare 処理後、SQL 命令を実行する
- `int sqlite3_finalize(sqlite3_stmt *pStmt);`
  prepare で作成した SQL 命令のバイトコードを削除します
- `int sqlite3_column_int(sqlite3_stmt*, int iCol);`
  iCol 番目のフィールドの数値から 3 2 ビット整数 (Int32) をかえします
- `const unsigned char *sqlite3_column_text(sqlite3_stmt*, int iCol);`
  iCol 番目のフィールドの文字データから UTF8 文字列をかえします
- `int sqlite3_bind_int(sqlite3_stmt*, int, int);`

[*7] Swift SQLite Tutorial for Beginners,by Belal Khan など
[*8] SQLite C Interface を転記しています

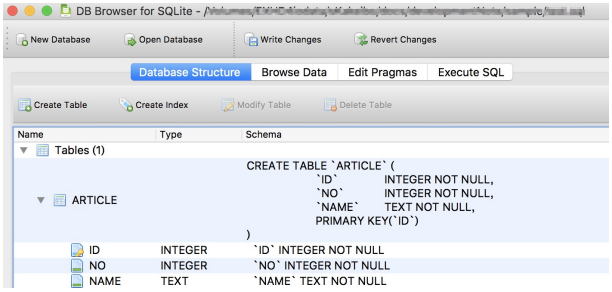
引数３番目の整数値を、SQL 命令のパラメータの引数２番目のにセットします。

- `int sqlite3_bind_text(sqlite3_stmt*,int,const char*,int,void(*)(void*));`
  引数３番目の文字列データを、SQL 命令のパラメータの引数２番目のにセットします
- `sqlite3_int64 sqlite3_last_insert_rowid(sqlite3*);`
  直近の SQL の INSERT 命令で作られたデータの ID(rowid) をかえします
- `const char *sqlite3_errmsg(sqlite3*);`
  SQLite3 処理が失敗した時に、そのエラーメッセージをかえします
- 定数
  - `#define SQLITE_OK           0   /* Successful result */`
  - `#define SQLITE_ROW         100  /* sqlite3_step() has another row ready */`
  - `#define SQLITE_DONE        101  /* sqlite3_step() has finished executing */`
- `sqlite3_prepare_v2()`、`sqlite3_step()`、`sqlite3_finalize()` のラッパー (wrapper) です

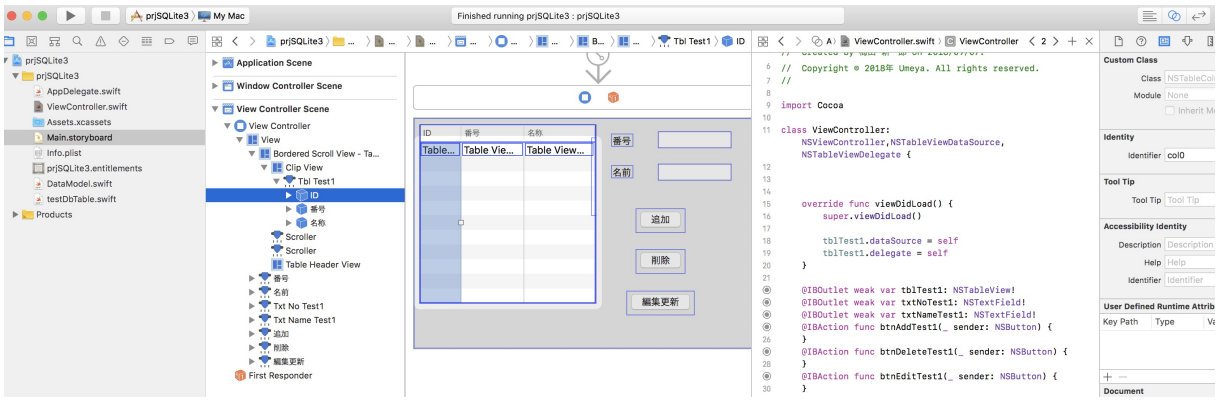
```
int sqlite3_exec(
  sqlite3*,                                  /* An open database */
  const char *sql,                           /* SQL to be evaluated */
  int (*callback)(void*,int,char**,char**),  /* Callback function */
  void *,                                    /* 1st argument to callback */
  char **errmsg                              /* Error msg written here */
);
```

3.2　SQLite3 を使う

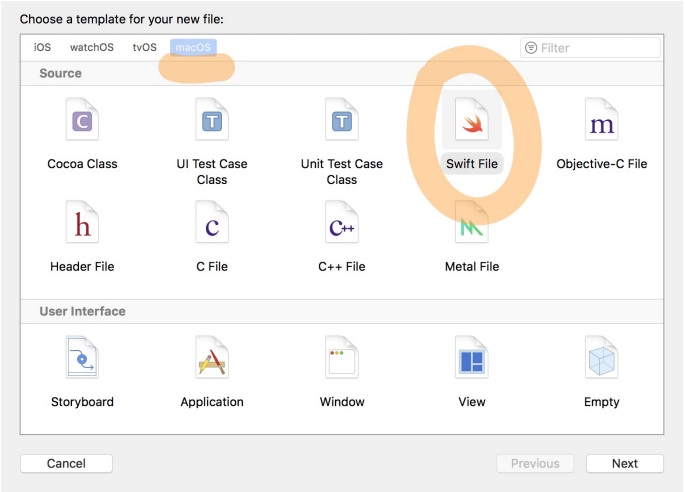
例として一つのテーブル ARTICLE をもつ testDb.sql ファイルを使います。テーブル内のフィールドには図のように 3 個のフィールドがあります。これは、DB Browser for SQLite を使って作ったものです。これにデータの読み込み (SELECT)、追加 (INSERT)、削除 (DELETE)、更新（UPDATE）をします。Table View を使ってデータを表示することにします。



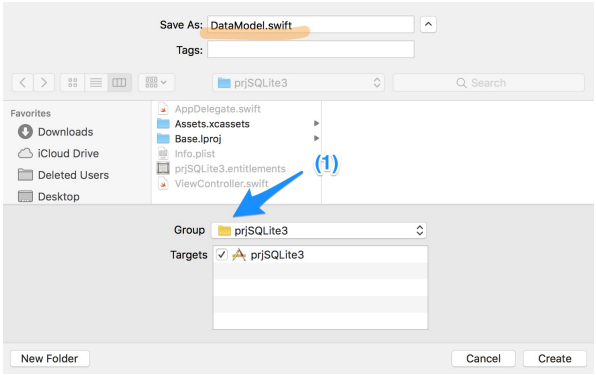
この例のために新しいプロジェクト prjSQLite3 を作ります。View Controller Scene に、TableView（列数は 3 列)、ラベル 2 個テキストフィールド 2 個、ボタン 3 個を図のように準備します。前節の prjTableView と同じにようにして、class ViewController のコードを参考にしながら作ります。テーブルの新しい列は、タイトル「ID」、identifire を「col0」とします。



次に、SQLite3 のコードを着て行きますが、このようなデータの処理部分をデータモデルとしてひとつのクラスに書き込みます。新規 Swift ファイル DataModel.swift を追加します。メニュー `File->New->File` からでも、Navigator インスペクタの prjSQLite3（黄色のフォルダー）右クリックでも良いので、画面の Choose a template for your new file:で Swift File を選び Next をクリック。



次のファイル保存ダイアログで名前を DataModel.swift にします。矢印（１）にあるように黄色のフォルダーの prjSQLite3 の Group になっていることを確認し、Create ボタンを押します。



同様の方法で、testDbTable.swift ファイルをプロジェクトに加えます。

先に、testDbTable.swift のコードを書きましょう。このなかに、testDb.sql データベース内のテーブル ARTICLE に合わせたデータ構造のクラスを作ります。TestDbTable.swift をメイン・エディタで開き、import する Foundation の部分を Cocoa に変えて、以下のコードを書き込みます。

```
import Cocoa
class Article{
    var Id: Int; var No: Int; var Name: String
    init(Id: Int, No: Int, Name: String){
        self.Id = Id; self.No = No; self.Name = Name   }
}
```

次に DataModel.swift ファイルの中にクラスを作ります。import の Foundation を Cocoa に変え、さらに SQLite3 も import します。まずはこのコードを準備します。

```
import Cocoa
import SQLite3
class DataModel{
    static let sharedInstance = DataModel()  //...(1)
    var testData1 = [Article]()              //...(2)
    fileprivate var db: OpaquePointer?       //...(3)
    fileprivate var stmt: OpaquePointer?     //...(4)
    fileprivate var strDbPath = Bundle.main.resourcePath! + "/testDb.sql""          //...(5)
    private init() {}
}
```

（１）で、このクラスの静的プロパティに sharedInstance インスタンスを用意しました。これをクラス ViewController から参照します。（２）はデータを保持する変数で、ここに全てのデータが入ります。（３）はデーベース・ファイルの、（４）は SQL 命令文の OpaquePointe[*9] （５）で testDb.sql ファイルを prjSQLite3.app/Contents/Resource に作ります。init() のコードも入れながら SQLite3 の例を見て行きます。

[*9] Swift で表現できない C ポインタを表すためのもの？よくわからない

3.3 データベースファイルの新規作成、オープン

sKakeibo.app では、データベースファイル sKakeibo.sql の雛形となるファイル sKakeibo.app/Contents/Resource/ においてあるので、ファイルやテーブルの新規作成はしていません。ここの例ではデータベースファイル testDb.sql がない時にファイルやテーブルの新規作成する処理を加えることにします。ファイルを開く openDB を書きます。これは、ファイルがない場合新規作成し ARTICLE テーブルを作成します。ここでは、sKakeibo では使わなかった `sqlite3_exec` を使っています。

```
func openDB(){
        if sqlite3_open(strDbPath, &db) != SQLITE_OK{
            let errmsg = String(cString: sqlite3_errmsg(db)!)
            someMessageWithAlert("error open database : \(errmsg)" , "stop the app")
            return
        }
        // create table ARTICLE if not exists
        let sql = "CREATE TABLE IF NOT EXISTS ARTICLE
           (Id INTEGER NOT NULL PRIMARY KEY,No INTEGER NOT NULL,Name TEXT NOT NULL)"
        if sqlite3_exec(db, sql, nil, nil, nil) != SQLITE_OK{
            let errmsg = String(cString: sqlite3_errmsg(db)!)
            someMessageWithAlert("error when create table, stop the app:\(errmsg)")
        }
    }
```

実行してみてください。プロジェクトファイルのあるところから
SQLite3/DerivedData/prjSQLite3/Build/Product/Debug/prjSQLite3.app
で、prjSQLite3.app を右クリックし「パッケージの内容を表示」。Contents/Resource/testDb.sql があります。DB Browser for SQLite などで開き、テーブル ARTICLE を確認してください。

3.4 データの読み込み　SELECT

下のコードのように読み込み用の関数 readDB を用意します。（1）で、データを保持する testData1 配列の要素をクリアします。（2）は SQL 命令の SELECT 文です。このように文字列で準備します。（3）は、この SQL 文をバイトコードにします。この準備 (prepare) が必要です。（4）は `sqlite3_step` で `SQLITE_ROW` が返される時、データ（レコード）があるので読み込みを繰り返します。（5）、（6）、（7）はデータ・フィールドの値を変数に代入しています。`sqlite3_column` の第 2 引数がその位置を示しています。（8）はそれぞれの値を testDat1 の要素として代入しています。（9）は `sqlite3_prepare` で処理された stmt を解除してしています。

```
func readDB() -> Bool{
        testData1.removeAll()                                          //...(1)
        let queryString = "SELECT * FROM ARTICLE"       //...(2)
        //prepare the query
        if sqlite3_prepare(db, queryString, -1, &stmt, nil) != SQLITE_OK {    //...(3)
            let errmsg = String(cString: sqlite3_errmsg(db)!)
            someMessageWithAlert("error prepare select from ARTICLE: \(errmsg)" , "stop the app")
            return false
        }
        //traversing through all records
        while (sqlite3_step(stmt) == SQLITE_ROW) {                          //...(4)
            let Id = sqlite3_column_int(stmt, 0)                                  //...(5)
            let No = sqlite3_column_int(stmt, 1)                                 //...(6)
            let Name = String(cString: sqlite3_column_text(stmt, 2))     //...(7)
            //adding values to list
            testData1.append(Article(Id:Int(Id), No:Int(No)
                                     ,Name:String(describing: Name)))  //...(8)
        }
        sqlite3_finalize(stmt)                              //...(9)
        return true
    }
```

まだ testDb.sql はレコードがありませんから、読み込みのテストができませんので、データの追加の処理を付け加えます。

3.5　データの追加　INSERT

DataModel クラスの中にレコード追加のメソッドを書き、ViewController でテキストフィールドに入力されたデータを引数にしてメソッドを呼び出してデータ追加の処理を行います。DataModel 内の関数は

```
func insertDB(no:Int, name:String) -> Bool {
        //the insert quary
        let queryString = "INSERT INTO ARTICLE (No,Name) VALUES (?,?)"    //...(1)
        //preparing the query
        if sqlite3_prepare(db, queryString, -1, &stmt, nil) != SQLITE_OK{        //...(2)
            let errmsg = String(cString: sqlite3_errmsg(db)!)
            someMessageWithAlert("error preparing  insert: \(errmsg)")
            return false
        }
        if sqlite3_bind_int(stmt, Int32(1), Int32(no)) != SQLITE_OK{              //...(3)
            let errmsg = String(cString: sqlite3_errmsg(db)!)
            someMessageWithAlert("failure binding int no: \(errmsg)")
            return false
        }
        let u8txt = (name as NSString).utf8String                                  //...(4)
        if sqlite3_bind_text(stmt, Int32(2), u8txt, -1, nil) != SQLITE_OK{       //...(5)
            let errmsg = String(cString: sqlite3_errmsg(db)!)
            someMessageWithAlert("failure binding text name: \(errmsg)")
            return false
        }
        //execute the query to insert values
        if sqlite3_step(stmt) != SQLITE_DONE {                                       //...(6)
            let errmsg = String(cString: sqlite3_errmsg(db)!)
            someMessageWithAlert("failure inserting ARTICLE : \(errmsg)")
            return false
        }

        sqlite3_finalize(stmt)                                                    //...(7)

        return readDB()                                                          //...(8)
    }
```

（1）は INSERT の SQL 命令文です。(No,Name) と並べられた順に 1 番目、2 番目と区別され（3）（5）でデータがバインドされます。int タイプの方は問題ないのですが、text の方はなかなかうまく動いてくれませんでした。ネットからの情報で（4）のように utf8 文字列で扱うと動きました。まだ、よくわかりません？

（８）でデータを testData1 に読み込んでいまし。

ViewController クラスのコードを書きます。まずは（１）で DataModel の静的インスタンスを名前 DM で参照できるようにします。

```
class ViewController: NSViewController,NSTableViewDataSource, NSTableViewDelegate {
    let DM = DataModel.sharedInstance          //...(1)
```

TableView のコードは prjTableView のものとほぼ同じですが、データ変数の参照を DM.testData1 としているところが異なるところです。

```
override func viewDidLoad() {
        super.viewDidLoad()
        tblTest1.dataSource = self
        tblTest1.delegate = self
    }
@IBOutlet weak var tblTest1: NSTableView!
@IBOutlet weak var txtNoTest1: NSTextField!
@IBOutlet weak var txtNameTest1: NSTextField!
@IBAction func btnAddTest1(_ sender: NSButton) {
    }
@IBAction func btnDeleteTest1(_ sender: NSButton) {
    }
@IBAction func btnEditTest1(_ sender: NSButton) {
    }

// tableView ----------------------------------------
 func numberOfRows(in tableView: NSTableView) -> Int {
        return DM.testData1.count
    }
func tableView(_ tableView: NSTableView, viewFor tableColumn: NSTableColumn?
                  , row: Int) -> NSView? {
        guard let vw = tableView.makeView(withIdentifier: tableColumn!.identifier
                     , owner: self) as? NSTableCellView else {return nil}
        if tableColumn!.identifier.rawValue == "col0"{
            vw.textField?.integerValue = DM.testData1[row].Id
        }else if tableColumn!.identifier.rawValue == "col1"{
            vw.textField?.integerValue = DM.testData1[row].No
        }else if tableColumn!.identifier.rawValue == "col2"{
            vw.textField?.stringValue = DM.testData1[row].Name
        }else{  return nil }
        return vw
```

```
    }
```

「追加」ボタン (btnAddTest1) を押すと、テキストフィールド txtNoTest1 と txtNameTest1 のデータが追加されるようにコードを書きます。[*10]（１）で DataModel の insertDB を使いデータベースへのレコードの挿入、さらに readDB で tsetData1 へのデータの追加をしています。（２）は testData1 にデータが追加されたのでそれをテーブルの反映させるようにリロードし、テクストフィールドをクリアしています。

```
@IBAction func btnAddTest1(_ sender: NSButton) {
        guard DM.insertDB(no: txtNoTest1.integerValue, name: txtNameTest1.stringValue)
             else{ return }                                                    //...(1)
        reloadDataAndCleartext(){                                  //...(2)
    }
...
func reloadDataAndCleartext(){
        tblTest1.reloadData()
        txtNoTest1.stringValue = ""
        txtNameTest1.stringValue = ""
    }
```

では、実行していくつかデータを追加してください。データテェックしていないので「番号」の入力は半角数字のみに注意してください。いくつかのデータを追加したら一度停止し、再度実行してみてください。追加したデータが起動画面のテーブルに表示されているはずです。DB Browser for SQLite で testDb.sql を見ても良いでしょう。

[*10] テキストフィールドのデータテェックしていません。テェックするには、prjTavleView を参考にしください。

3.6　データの削除 (DELETE) と編集更新 (UPDATE)

テーブルで処理対象となるデータ行の選択結果をテキストフィールドに表示します。prjTableView のテーブルとほぼ同じです。変数 selRowNo はグローバルスコープに定義され、選択されたデータの行番号を保持しています。選択されていない時は-1 になっています。これから作る関数（データの削除 (btnDeleteTest1) と編集更新 (btnEditTest1)）から参照されます。

```
func tableViewSelectionIsChanging(_ notification: Notification) {
        selRowNo = tblTest1.selectedRow
        guard selRowNo != -1 else {return}
        txtNoTest1.integerValue = DM.testData1[selRowNo].No
        txtNameTest1.stringValue = DM.testData1[selRowNo].Name
}
```

DataModel の中に deleteDB と updateDB を作ります。コードは（６）以外はこれまでのものと同じはたらきをしています。（６）も SQL 命令文の文字列を作っているのですが、フィールドに SET する時、テキストデータはクォーテーションマークで囲む必要があり、文字列のダブル・クォーテーション内なのでシングル・クォーテーションで囲んでいます。ダブル・クォーテーションマークを D、シングル・クォーテーションマークを S で書くと（６）の該当する部分は

```
,Name= '" + name + "'  WHERE
,Name= SD + name + DS  WHERE
```

と name の文字列をシングル・クォーテーションマークで囲った形になります。

ViewController クラス内の btnDeleteTest1 と btnEditTest1 のコードを書きます。

```
func deleteDB(id:Int) -> Bool{
        let queryString = "DELETE FROM ARTICLE WHERE Id=" + String(id)  //...(1)

        //preparing the query
        if sqlite3_prepare(db, queryString, -1, &stmt, nil) != SQLITE_OK{       //...(2)
            let errmsg = String(cString: sqlite3_errmsg(db)!)
            someMessageWithAlert("error preparing delete: \(errmsg)")
            return false
        }
        //execute the query to insert values
        if sqlite3_step(stmt) != SQLITE_DONE {                                          //(3)
            let errmsg = String(cString: sqlite3_errmsg(db)!)
            someMessageWithAlert("failure delete from ARTICLE: \(errmsg)")
            return false
        }
```

```
        sqlite3_finalize(stmt)                                          //...(4)
        return readDB()                                                 //...(5)
    }

func updateDB(id:Int, no:Int, name:String)->Bool{
        let queryString = "UPDATE ARTICLE SET no= " + String(no)
                          +  " ,Name= '" + name + "'  WHERE Id=" + String(id)    //...(6)
        if sqlite3_prepare(db, queryString, -1, &stmt, nil) != SQLITE_OK {
            let errmsg = String(cString: sqlite3_errmsg(db)!)                          //...(7)
            someMessageWithAlert("error prepare update ARTICLE: \(errmsg)" )
            return false
        }
        //execute the query to insert values
        if sqlite3_step(stmt) != SQLITE_DONE {                                  //...(8)
            let errmsg = String(cString: sqlite3_errmsg(db)!)
            someMessageWithAlert("failure update: \(errmsg)")
            return false
        }
        sqlite3_finalize(stmt)                                          //...(9)
        return readDB()                                                 //...(10)
    }
```

それぞれ選択行が有効かテェックし、削除 (deleteDB) または更新 (updateDB) で処理をして、データのリロード、テキストフィールドのクリア、選択行の解除をしています。実行してください。

```
 @IBAction func btnDeleteTest1(_ sender: NSButton) {
        guard selRowNo != -1 else{return}
        guard DM.deleteDB(id: DM.testData1[selRowNo].Id) else{return}
        reloadDataAndCleartext()
        deselectRow()
    }
    @IBAction func btnEditTest1(_ sender: NSButton) {
        guard selRowNo != -1 else{return}
        guard DM.updateDB(id: DM.testData1[selRowNo].Id,
                  no: txtNoTest1.integerValue, name: txtNameTest1.stringValue) else{return}
        reloadDataAndCleartext()
        deselectRow()
    }
.....
func deselectRow(){
        tblTest1.deselectRow(selRowNo)
```

```
        selRowNo = -1
    }
```

以上で大体の SQL 命令は書けると思います。

4　複数の View

メインとなるウィンドウに別のウィンドウ（メッセージモーダル、タブ、ビューなど）からのデータを受けたり、そのウィンドウにデータを渡したりすることが多くあります。

4.1　NSAlert の使用

NSAlert は主に、メッセージを表示するものですが、そのダイアログ上でのボタン選択に応じたデータを受け取ることもできます。

4.1.1　メッセージの表示

まずは、メッセージを表示するだけのものを書いてみます。タイトルが「MessageAlert」のボタンを用意し、名前「btnMessageAlert」で Action 接続します。次のコードを書きます。

(1) で NSAlert のインスタンス alert を作ります。（2）と（3）は表示するメッセジです。実行結果をみてください。（4）はあとで、説明します。

```
@IBAction func btnMessageAlert(_ sender: NSButton) {
        let alert = NSAlert()                                   //...(1)
        alert.messageText = "メッセージです"              //...(2)
        alert.informativeText = "情報テキストです"    //...(3)
        //alert.alertStyle = NSAlert.Style.critical      //...(4)
        alert.runModal()
    }
```

実行します。コードの（1）（2）の部分を確認してください。表示されるアイコンは、図の左側のものが AppIcon をセットしてない状態でシステムの規定画像があります。右はアイコンを設定したものです。

NSAlert にはプロパティがいくつかあります。（4）のコメントを外して実行してみてください。アイコンが誓うものになっているはずです。alertStyle はこの critical の他に warning と informatinal があります。[*11] そこで icon プロパティががあるので、そこにアイコンのファルを設定してみました。コード（4）の部分を次のもの変えて実行してみました。

```
alert.icon = NSImage(named: NSImage.Name(rawValue: "warning.png"))
```

[*11] warrning と informational は変化がありません。どこかで設定するのか？わかりません。

下の図 (f1) は NSAlert.Style.critical のもの、(f2) はアイコン warning.png、(f3) は informational.png を使ったものです。なお、２つの png ファイルはこのプロジェクトに追加したものです。

4.1.2　メッセージへの応答

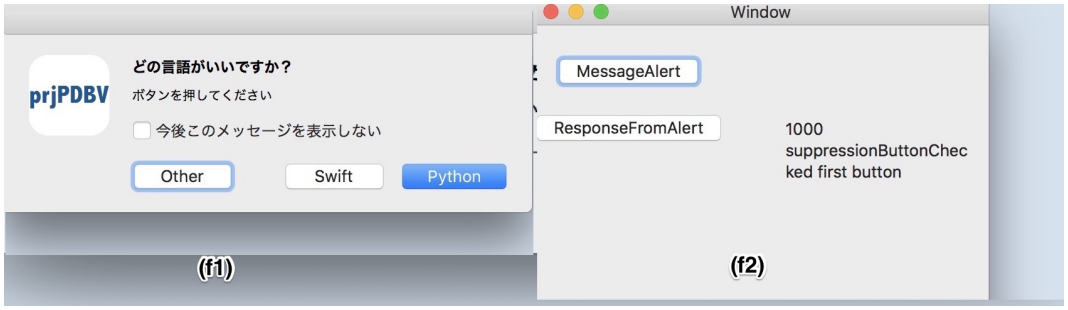
表示するダイアログにいくつかボタンを用意し、どのボタンが押されたかの応答をうる方法についてです。タイトルが「ResponseFromAlert」のボタンを追加し、「btnResponseFromAlert」の名前で Action 接続します。返された値を表示するためにラベルを追加し「lblResponseFromAlert」の名前で Outlet 接続しておきます。次のコードを見てください。（１）、（２）、（３）のように必要なボタンをタイトルをつけてアラート画面に追加します。コメントにある「Magic Number」はそれぞれのボタンが押された時の戻り値です。つまり、1000、1001、... と追加した順に値が割り当てられます。この値は（５）のように表示することもできます。Xcode のコード自動保管を利用してどんなものがあるか調べてください。

```
@IBOutlet weak var lblResponseFromAlert: NSTextField!
@IBAction func btnResponceFromAlert(_ sender: NSButton) {
        let alert = NSAlert()
        alert.messageText = "どの言語がいいですか？"
        alert.informativeText = "ボタンを押してください"
        alert.addButton(withTitle: "Python") // 1000     ...(1)
        alert.addButton(withTitle: "Swift")   // 1001     ...(2)
        alert.addButton(withTitle: "Other")  // 1002    ...(3)
        let res = alert.runModal()
        lblResponseFromAlert.stringValue = String(res.rawValue)      //....(4)
        if res == NSApplication.ModalResponse.alertFirstButtonReturn{                //...(5)
            lblResponseFromAlert.stringValue = lblResponseFromAlert.stringValue + "first button"
        }
    }
```

もう一つ、「今後このメッセージを表示しない」というテェックボタン付きのメッセージの取扱方法についてです。まず、これをアラート画面に表示するのは、「showSuppressionButton」を「true」にします。テェックされているかどうかは、「.suppressionButton.state」が「.on」かどうかで調べます。次のコードは、前の「@IBAction func btnResponceFromAlert」の（３）の下に（６）、（４）の下に（７）を挿入したものです。

```
alert.showsSuppressionButton = true                    //...(6)
....
if alert.suppressionButton?.state == .on{
            lblResponseFromAlert.stringValue = lblResponseFromAlert.stringValue
                        + " suppressionButtonChecked"}                              //...(7)
```

図は、アラート画面 (f1) と、そこでテェックを入れ「python」をクリックした状態 (f2) のものです。[*12]



(f1) の「今後このメッセージを表示しない」が英語表記になっている時には、Xcode のメニュー `View->Navigators->show Project Navigator` で「青」のプロジェクタアイコンをクリックし「info」を選択します。「Localization native development region」の欄を「Japan」に変えます。そしてそのまま右クリックを押し「Japanese」を選びます。こちらの方法で Xcode アプリの標準言語を日本語に変えることができます。

[*12] ラベルのサイズを大きくしています。

4.2 NSTabView と NSTabViewController

4.2.1 NSTabView

NSTabView について Apple Developer Documentation に以下のように書いてあります。

An NSTabView object provides a convenient way to present information in multiple pages. The view contains a row of tabs that give the appearance of folder tabs, as shown in the following figure. The user selects the desired page by clicking the appropriate tab or using the arrow keys to move between pages. Each page displays a view hierarchy provided by your application.

その通りで、今まで使った ViewController 上に一つのコントロールとして配置して、複数のページを簡単に実現できます。新しいプロジェクト prjTabView を作り NSTabView の動きを見たいと思います。View Controller 上に Tab View をおきます。この属性インスペクタで Tabs の数を 2 から 3 に変えてページ数を 3 ページにします。サイズは適当な大きさにしてください。この Tab View を「tbvTest」の名前で Outlet 接続します。ドキュメント構造図でページを指定して、3 つのページのタイトルと identifier をそれぞれ「本店、idHonten」、「支店 A、idSitenA」、「支店 B、idSitenB」としてきます。それぞれのページに Button と Labe と TextField を一つずつおきます。Button は Action 接続で、Label と TextField は」Outlet 接続で下のように準備しておきます。

```
@IBOutlet weak var tbvTest: NSTabView!
@IBOutlet weak var lblHonten: NSTextField!
@IBOutlet weak var txtHonten: NSTextField!
@IBOutlet weak var lblSitenA: NSTextField!
@IBOutlet weak var txtSitenA: NSTextField!
@IBOutlet weak var lblSitenB: NSTextField!
@IBOutlet weak var txtSitenB: NSTextField!
@IBAction func btnHonten(_ sender: NSButton) {
}
@IBAction func btnSitanA(_ sender: NSButton) {
}
@IBAction func btnSitenB(_ sender: NSButton) {
}
```

ボタンのタイトルを、本店のページのは「支店 A と B へ」、支店 A は「支店 B から」、支店 B は「本店へ」とします。それぞれのボタンを押した時に、タイトルにある方向で、テキストフィールのデータを移動先のラベルに表示させるようにコードを書きます。

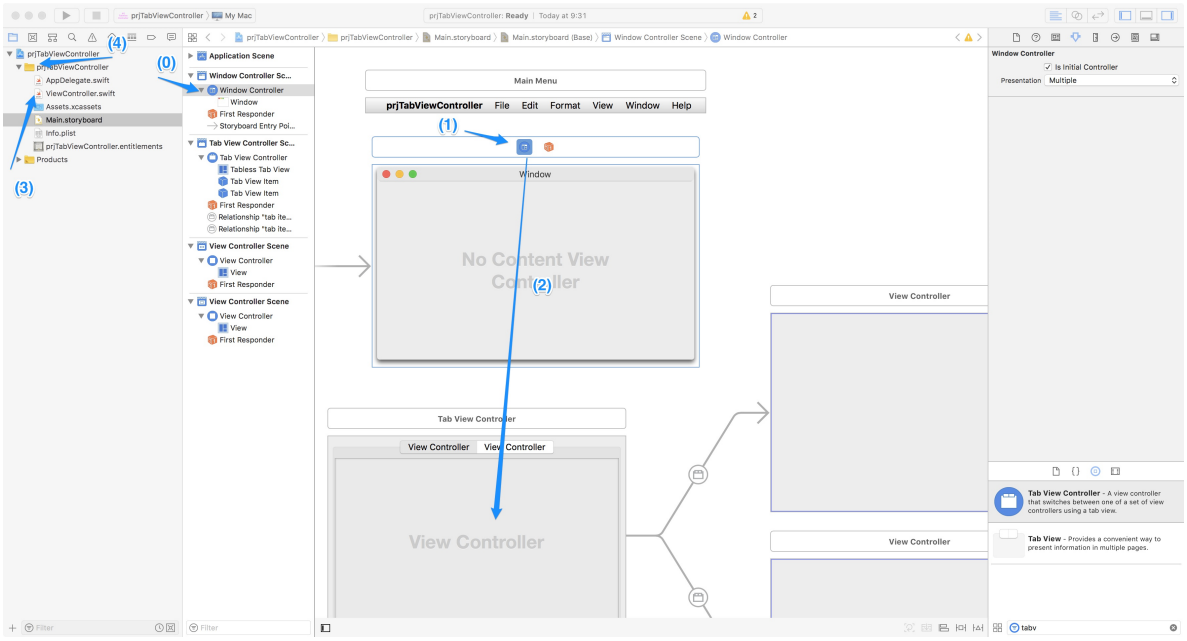
```
@IBAction func btnHonten(_ sender: NSButton) {
    // to SitenA SitenB
    lblSitenA.stringValue = txtHonten.stringValue     //...(1)
    lblSitenB.stringValue = txtHonten.stringValue     //...(1')
}
```

```
@IBAction func btnSitanA(_ sender: NSButton) {
    // from SitenB
    lblSitenA.stringValue = txtSitenB.stringValue     //...(2)
}
@IBAction func btnSitenB(_ sender: NSButton) {
    // to Henten
    lblHonten.stringValue = txtSitenB.stringValue    //...(3)
    tbvTest.selectTabViewItem(withIdentifier: "idHonten")      //...(4)
}
```

コード (1)(1')(2)(3) はデータをページ間で送っているものです。「見た目」がだけですがデータのページ間での受け渡しができています。(4) は支店 B から本店のページを表示しています。図は、起動直後の画面です。各ページでテキストフィールドにいろいろな値を入れて動かしてみてください。

4.2.2 NSTabViewController

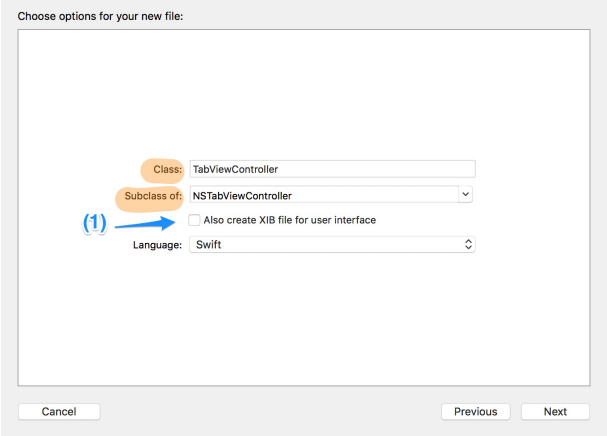
これまで使っていた ViewController の代わりに TabViewController を使います。プロジェクト prjTabViewController を新規作成します。はじめに View Controller Scene を削除します。ドキュメント構成図（Document Outline）で View Controller Scene を選択し、BackSpace キーを押して削除します。Window Controller Scene の画面には、「No Content View Controller」と表示されています。ライブラリーの検索キーに tabv とでも打つと Tab View Controller と Tab View の 2 個が出ます。青い円形の背景がある Tab View Controller の方を DragPaste します。図は、この直後に Window Controller（矢印（0））をクリックしたものです。



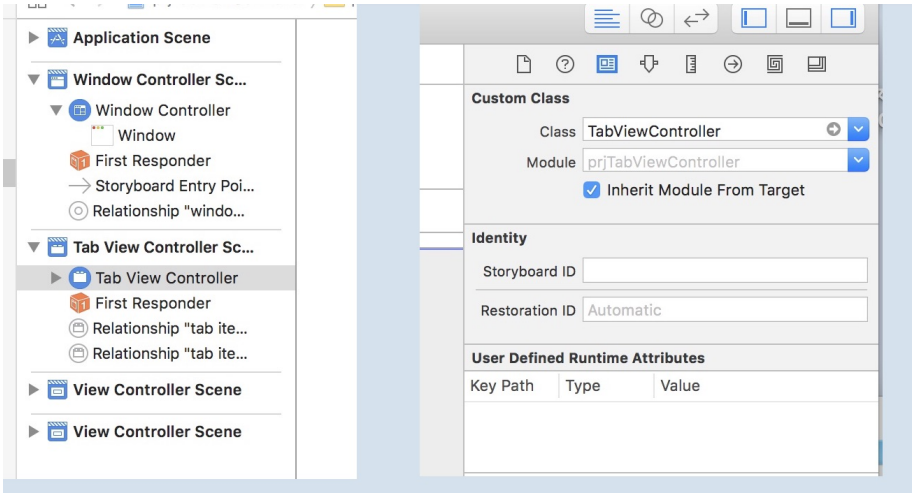
矢印（1）のアイコンをクリックし矢印（2）の方向にしたの ViewController までドラッグすると ViewController 部分が青色に変わります。そこでマウスを話してください。ポップアップ・メニューに「Relationship Segue windou Content」が出ます、クリックしてください。これで Window Controller に Tab View Controller がリンクされました。Storyboard で複数の画面の行き来（画面遷移）を実現する方法としての Segue[*13]を使っています。Tab View Controller を使ったアプリではそれぞれの Tab View をコントロールするクラスを持ちます。NavigatorPanel の矢印（3）ViewController.swift をクリックしバックスペースキーで削除します。「ゴミ箱に捨てるか？どうか」のメッセージには「ゴミ箱へ」を選択し、完全に削除します。次にクラスを新たに追加します。矢印（4）を右クリックし、「New File」を選択します。

[*13] セグエとでも読むのでしょうか

「Choose a template for your new file」で「macOS、Cocoa Class」を選択し、「Next」をクリックします。図のオプション選択画面で Class を TabViewController、SubclassOf で NSTabViewController を書き込みます。矢印（１）の XIB[*14] のところのチェック入れない。



ここで作ったクラスを TabViewController の Custom Class とします。ドキュメント構成図の TabViewControllerScene の中の TabViewController を選択し、インスペクタでアイデンティティ・インスペクタを開きます。その Custom Class にクラス TabViewController を指定します。



この TabViewController にはすでに２つの ViewController が用意されています。３つ必要となった場合は、View Controller を追加します。ライブラリで View Controller（青の円形の背景に白の正方形のアイコン）を選びインターフェイス・ビルダー（Storyboard）にドラッグしてきます。ドキュメント構成図の TabViewControllerScene の中の TabViewController から追加した ViewControll の画面までコントロール・ドラッグします。不要な View がある場合はドキュメント構成図の該当する TabViewControllerScene を選択し（バックスペースキーを使って）削除します。これで３つの View ができたので、それぞれのタイトルを「本店」、「支店 A」、「支店 B」とします。これは、それぞれの ViewControll を選択し、属性インスペクタで指定できます。これで一度実行します。タブを押すとそれぞれのビューが表示されます。（いまは、何も配置していないので、なにもないのですが）３つの ViewController にクラスを用意します。先に TabViewController のときと同様の方法で Custom Class に指定します。

[*14] XML Interface Builder の略で、コードを使って部品の整理をするものらしい。 Storyboard での開発を主にしているので、いつか機会があったら勉強します。

1. プロジェクトへの新規ファイル追加で Cocoa Class ファイルを追加します。
   それぞれ、HontenViewController、SitanAViewController、SitenBViewController とします。NSTab-ViewController の Subclass で XIB のテェックは必ず外す。
2. ドキュメント構成図で本店 Scene の ViewController(本店と表示されている) を選択し、アイデンティティ・インスペクタで Custom Class の名前を HontenViewController とします。残り 2 つにも同様にしてそれぞれのクラス名を入れます。

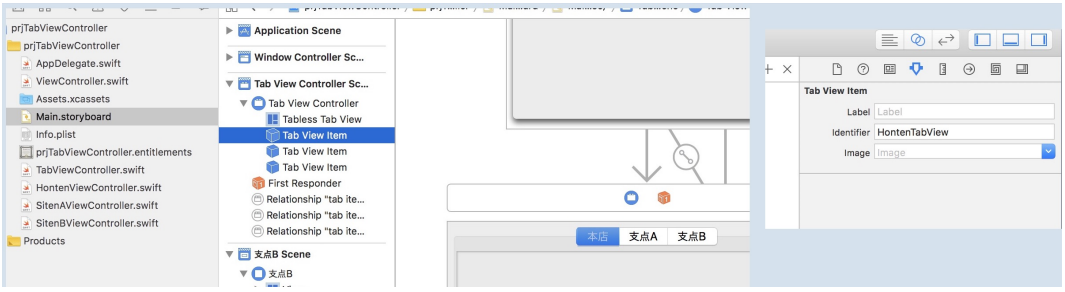
3 つの TabView が用意できたので、それぞれのタブが選択された時の処理のコードを書いて見ます。TabViewController.swift のファイルをメインでもアシスタントでも良いのでエディタ領域で編集します。次のコードのように tabViw 関数の didSelect と willSelecte の 2 つの関数を用意します。実行してデバッグ領域を観察してください。タブを選択した時の動きがわかります。

```
override func tabView(_ tabView: NSTabView
                              , didSelect tabViewItem: NSTabViewItem?) {
       print("tabView didSelect")       //...(1)
       }
override func tabView(_ tabView: NSTabView
                             , willSelect tabViewItem: NSTabViewItem?) {
       print("tabView willSelect")      //...(2)
       }
```

コード (1) を次のように書き換えます。オプショナル型の `tabViewItem: NSTabViewItem?`から選択された TabView を取り出し変数 tabItem に代入しています。実行してみます。

```
if let tabItem = tabViewItem{   print(tabItem.label)  }
```

それぞれの TabView にアイデンテティ (idendifire) が設定を設定して、それぞれの View 上での処理を書いてみます。それぞれの TabView のアイデンテティを HontenTabView、SitenAtabView、SitenBTabView とします。これはドキュメント構成図で TabViewControllerScene を開いて TabViewItem を選択し、属性インスペクタの identifier の欄に書き込みます。



3 つの TabView に一つずつラベルを「lblThisTab」という名前で Outlet 接続してください。前のコードの（2）の部分を以下のように書き換えます。（1）のコードは削除するかコメント・アウトしてください。

```
       if let tItem = tabViewItem {
           let strID = tItem.identifier as! String
```

```
            switch strID {
            case "HontenTabView":
                let VC = tItem.viewController as! HontenViewController
                VC.lblTHisTabView.stringValue = "本店です"

            case "SitenATabView":
                let VC = tItem.viewController as! SitenAViewController
                VC.lblThisTabView.stringValue = "支店 A"
            case "SitenBTabView":
                let VC = tItem.viewController as! SitenBViewController
                VC.lblThisTabView.stringValue = "支店 B"
            default:
                // debug print
                print("default ")
                break
            }
        }
    }
```

このように TabViewController の中では、それぞれの TabView が設定した identifier で識別されます。では、TabView 間でのデータの受け渡しを書いてみます。本店 TabView にボタンとテキスト・フィールドを配置します。ボタンのタイトルを「支店 AB へ」とでもして HontenViewController に Action 接続をします。名前は「btnSendAB」にします。残り 2 つの TabView にテキスト・フィールドを一つ配置します。3 つのテキスト・フィールドをそれぞれの ViewController に Outlet 接続します。名前は「txtOnTab」と全部同じにしておきます。別々でも構いません。

```
@IBAction func btnSendAB(_ sender: NSButton) {
        guard let parentTabVC = parent as? NSTabViewController
          else { return }                                                    //...(1)

        if let target = parentTabVC.childViewControllers[1] as?
             SitenAViewController {                                  //...(2)
            target.txtOnTab.stringValue = txtOnTab.stringValue + " 支店 A"}
        if let target = parentTabVC.childViewControllers[2] as?
           SitenBViewController {
            target.txtOnTab.stringValue = txtOnTab.stringValue + " 支店 B"}
    }
    @IBOutlet weak var txtOnTab: NSTextField!
```

コード (1) で本店 TabView の親 TabView (ここでは、TabViewController) を取得し、(2) で子 TabView を

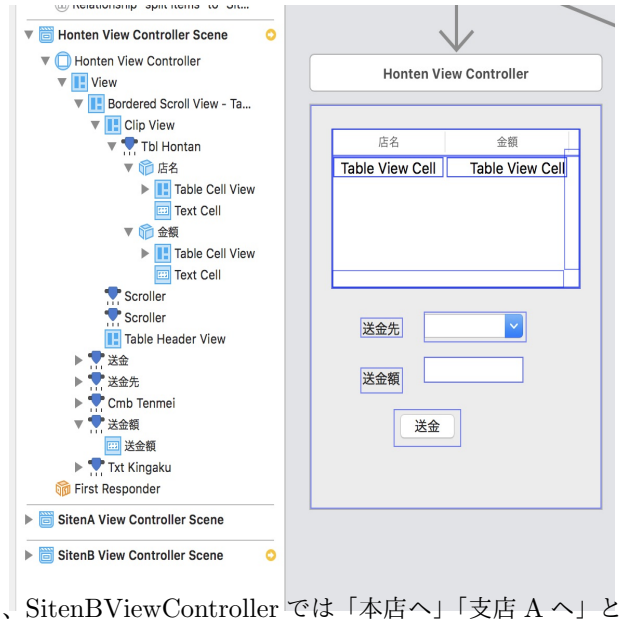
`childViewControllers[1] as? SitenAViewController` で変数 target に代入して、それを使ってデータを渡しています。この例では、`childViewControllers[0,1,2]` は順に本店、支店 A、支店 B の TabView になっています。実行して動作を確認してください。この部分のコードは P.Hudoson 氏のものを参考にしました。

4.3 NSSplitViewController

ウィンドウに３個のビューを横並びしてみます。準備することは、NSTabViewController のときとよく似ています。

1. ドキュメント構成図で ViewControllerScene を選択、バックスペース・キーで削除。ナビゲータ領域 (NvigatorPlane) で ViewController.swift を選択、バックスペース・キーで削除。
2. ライブラリ・インスペクタで VerticalSplitViewController を選択、StoryBoard にドラッグ・ペースト。
3. WindowContoroller（青い四角のアイコン）か、ドキュメント構成図の WindowControllerScene の WindowController から、今入れた SplitViewController へコントロール・ドラッグ。（TabViewController の説明を見てください）
4. ３つ目のビューを追加する。ライブラリー・インスペクタで ViewController を選択し StoryBoard に追加。SplitViewControllerScene の SplitViewController から追加した ViewController へコントロール・ドラッグ。この操作のような「接続 (Conection)」のときには、ドキュメント構成図から Storyboard 中のアイテムにコントロール・ドラッグするのが良いでしょう。
5. プロジェクトへの新規ファイル追加で Cocoa Class ファイルを追加します。
   それぞれ、HontenViewController、SitenAViewController、SitenBViewController とします。NSTabViewController の Subclass で XIB のテェックは必ず外す。
6. ドキュメント構成図で本店 Scene の ViewController(本店と表示されている) を選択し、アイデンティティ・インスペクタで Custom Class の名前を HontenViewController とします。残り２つにも同様にしてそれぞれのクラス名を入れます。

３つのビューに同じようにコントロールを配置します。HontenViewController で説明します。tableView、comboBox、textField、button を一つずつ、label を２個配置します。（図のように適当に配置してください）TableView は２列で、それぞれの TableColumn のタイトルを「店名」と「金額」、identifier を「tenmei」と「kingaku」とします。アシスタント・エディタで HontenViewController。swift を開き、そこに名前「tblHonten」でに Outlet 接続します。NSTableViewDataSource と NSTableViewDelegate のプロトコルを付け加えます。comboBox は単純にリストからの項目選択だけなので、 名前「cmbTenmei」で Outlet 接続ののち、その属性インスペクタの Items の欄で図のように「支店 A へ」「支店 B へ」としておきます。



SitenAViewController では「本店へ」「支店 B へ」、SitenBViewController では「本店へ」「支店 A へ」と自身への送金先はないものとしておきます。

TextField は、名前「txtKingaku」で Outlet 接続します。Button は、名前「btnSoukin」で Action 接続し、タイトルを送金とします。label はそぞれのタイトルを「送金額」「送金先」とします。残り 2 つのビューへも同じコントロールを配置します。[*15]

前にやったように、DataModel.swift を追加し、そこに DataModel クラスを作ります。そのコードです。コード（1）を利用して 3 つの SplitViewController でデータを共有します。

```
import Cocoa
class DataModel{
    static let sharedInstance = DataModel()                  //...(1)
    var hontenList = [(namae:String, kingaku:Int)]()
    var sitenAList = [(namae:String, kingaku:Int)]()
    var sitenBList = [(namae:String, kingaku:Int)]()
    private init(){
        hontenList.append(("初期資金", 1000))
        sitenAList.append(("初期資金", 500))
        sitenBList.append(("初期資金", 300))
    }
    func soukin(sendFrom:String, sendTo:String, kingaku:Int){
        switch sendFrom {
        case "本店":
            hontenList.append((sendTo + "へ", kingaku))
        case "支店 A":
            sitenAList.append((sendTo + "へ", kingaku))
        case "支店 B":
            sitenBList.append((sendTo + "へ", kingaku))
        default:
            return
        }
        switch sendTo {
        case "本店":
            hontenList.append((sendFrom + "から", kingaku))
        case "支店 A":
            sitenAList.append((sendFrom + "から", kingaku))
        case "支店 B":
            sitenBList.append((sendFrom + "から", kingaku))
        default:
            return
        }
```

[*15] Xcode の Edit メニューの Duplicate を使ってコントロールをコピーすることもできます

```
    }
}
```

では、HontenViewController のコードを書きます。TableView の部分は以前見たものとほとんど同じです。（１）で comboBox での送金先の選択アイテムのインデックスを得ています。0 が支店 A、1 が支店 B です。SitenAViewController では 0 が本店、1 が支店 B、SitenBViewController では 0 が本店、1 が支店 A となります。インデックスが-1 の場合は選択されていないので、（２）で処理を中断しています。（３）（５）は以前にも使いましたが、（３）で「親 View」を取得し、その「子 View」を children[i][*16]で変数に代入しています。children[0] が HontenViewController、children[1] が SitenAViewController、children[2] が SitenBViewController となり、この変数 toVC をかいして、それぞれのテーブルを reload するメソッド（関数）を（５）で呼び出しています。（４）は DataModel の soukin メソッド（関数）を呼び出し、データの更新をしています。（６）は自身のテーブルの reload で、（７）は他の ViewController から呼び出されます。実行してみてください。

```
class HontenViewController: NSViewController,NSTableViewDataSource, NSTableViewDelegate {
    let DM = DataModel.sharedInstance
    override func viewDidLoad() {
        super.viewDidLoad()
        tblHontan.dataSource = self
        tblHontan.delegate = self
    }
    @IBOutlet weak var cmbTenmei: NSComboBox!
    @IBOutlet weak var txtKingaku: NSTextField!
    @IBAction func btnSoukin(_ sender: Any) {
        let idxTenmei = cmbTenmei.indexOfSelectedItem                          //...(1)
        guard idxTenmei != -1 else{return}                                                  //...(2)
        guard let spvc = parent as? NSSplitViewController else{return}  //...(3)
        if idxTenmei == 0 { // from honten to sitenA
            DM.soukin(sendFrom: "本店", sendTo: "支店 A", kingaku: txtKingaku.integerValue) //...(4)
            if let toVC = spvc.children[1] as? SitenAViewController {
                toVC.refreshTable()                    //...(5)
            }
        }else if idxTenmei == 1 { // honten to sitenB
            DM.soukin(sendFrom: "本店", sendTo: "支店 B", kingaku: txtKingaku.integerValue)
            if let toVC = spvc.children[2] as? SitenBViewController {
                toVC.refreshTable()
            }
        }else{
            return
```

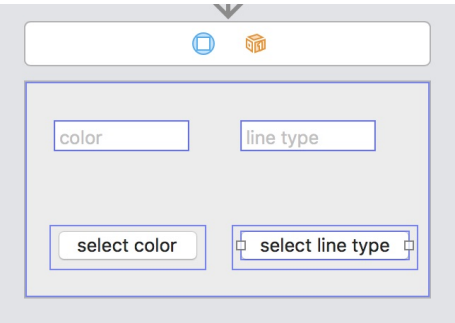
[*16] 2018/9/23 現在、Xcode10.0, Swift4.2 になっていました。この children も前は、childViewController でした。

```
        }
        tblHontan.reloadData()     //...(6)
    }
    func refreshTable(){             //...(7)
        tblHontan.reloadData()
    }
    //----------------------------------------------------------------------table view
    @IBOutlet weak var tblHontan: NSTableView!
    func numberOfRows(in tableView: NSTableView) -> Int {
        return DM.hontenList.count
    }
    func tableView(_ tableView: NSTableView,
                viewFor tableColumn: NSTableColumn?, row: Int) -> NSView? {
        guard let vw = tableView.makeView(withIdentifier: tableColumn!.identifier,
                  owner: self) as? NSTableCellView else {return nil}

        switch tableColumn!.identifier.rawValue {
        case "tenmei":
            vw.textField?.stringValue = DM.hontenList[row].namae
        case "kingaku":
            vw.textField?.integerValue = DM.hontenList[row].kingaku

        default:
            return nil
        }
        return vw
    }
}
```

4.4　主となるウィンドウと補助ウィンドウ

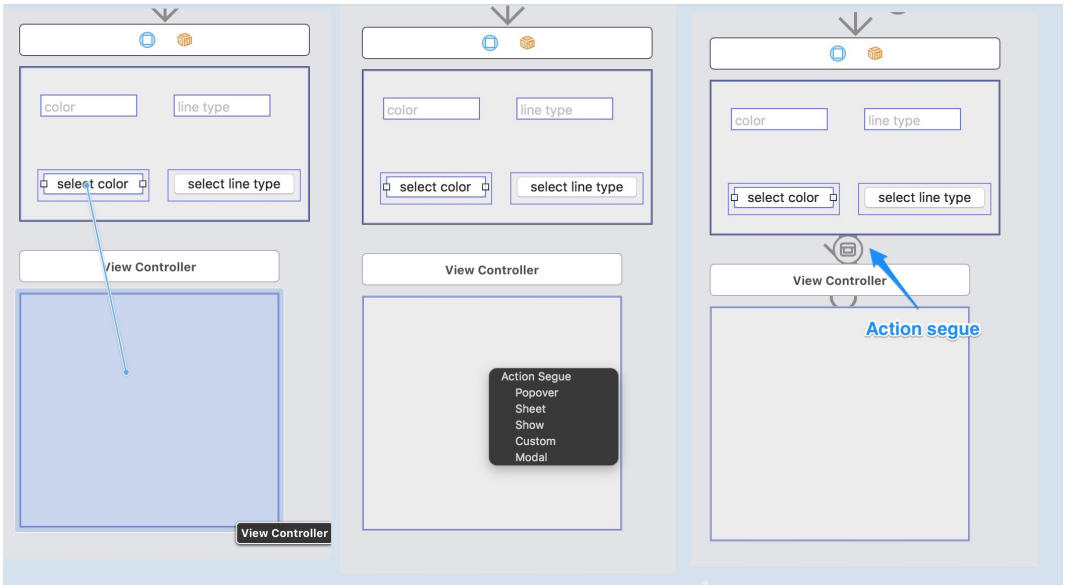
複数のウィンドウを扱うアプリで、補助的なウィンドウを使用する場面は多くあります。むしろ、これまでの TabViewController や SplitViewController より多いかもしれません。その取り扱いを書いていきます。方法は TabViewController や SplitViewController での View の追加でとの時と同じですが、主ウィンドウ (View）での Action から追加 View への接続（Segue）の方法をみてください。

主ウィンドウ上のボタンを押すことで補助ウィンドウが開かれ、そこへデータを送り、そこでで設定されたデータを受け取るという処理を作ってみます。prjAuxiliaryView を新規作成します。主ウィンドウとなる ViewController 上に 2 つのテキスト・フィールドとボタンを配置します。それぞれ Outlet 接続、Action 接続します。

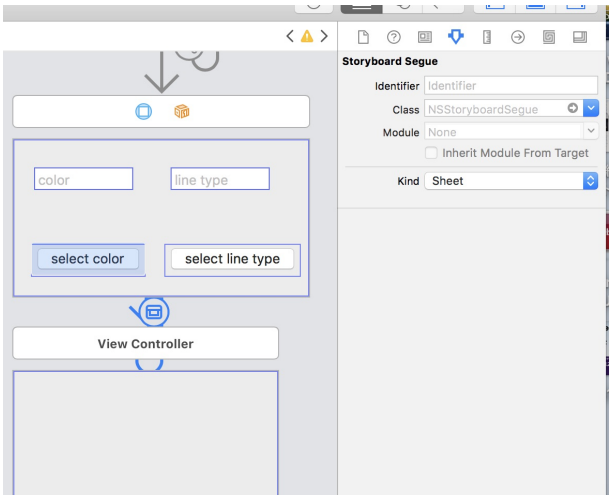



```
@IBOutlet weak var txtColor: NSTextField!
@IBOutlet weak var txtLineType: NSTextField!
@IBAction func btnSelectColor(_ sender: Any) {   }
@IBAction func btnSelectLineType(_ sender: Any) {  }
```

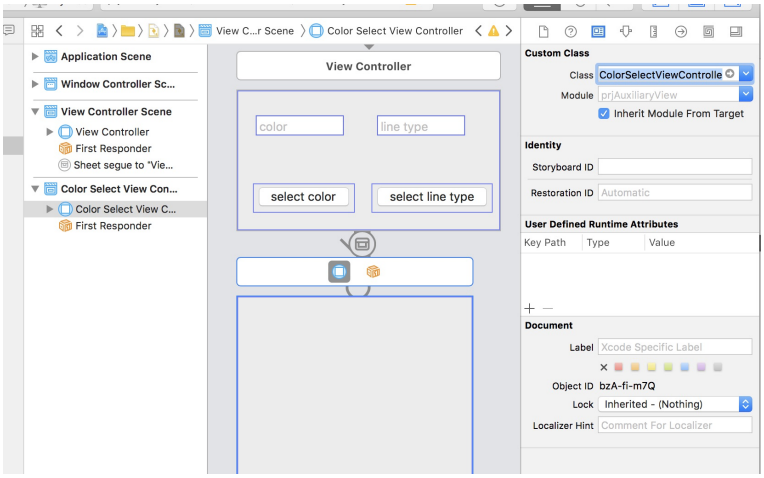
新しい View を Storyboad 上におきます。btnSelctColor ボタンからこの追加した View へコンロール・ドラッグします。左の図のようになります。マウスを離したところで中央の図のように ActionSegue のいくつかの選択が出ます。ここでは Sheet を選んでおきます。右の図は操作終了後の図で 2 つのビューが繋がれた状態を表しています。



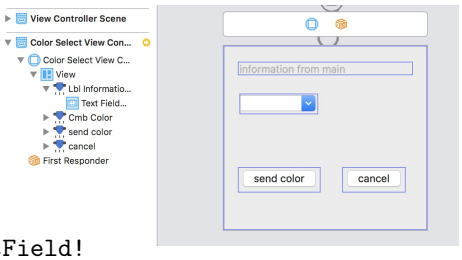
ボタンのインスペクタで、接続インスペクタ (ConectionsInspector) を見ると、TriggeredSegues のところに設定したものが表示されています。実行してみます。ボタンを押すとモーダルモードで補助画面が表示されます。ViewController（主ウィンドウ）から繋がれた補助ウィンドウの View への矢印の途中のアイコンを選択します。属性インスペクタの Storyboard に identifier の欄があるので、ColorSelect としておきます。



追加した View のクラス、コントロールの配置を行います。ColorSelectViewContoroller.swift の名前で Cocoa クラスファイルを作成します。SubClass:は NSViewController で XIB ファイルのテェックは外します。ColorSelectViewContoroller クラスをを追加した ViewController の CustomView にします。



ColorSelectViewContoroller 上にラベルと comboBox、２つのボタンをおきます。comboBox の属性インスペクタの item を「red」「blue」「white」とでも書き換えます。２つのボタンの Action 接続された関数に次のコードのように dismiss(self) を書き入れます。




```
@IBOutlet weak var lblInformationFromMain: NSTextField!
@IBOutlet weak var cmbColor: NSComboBox!
@IBAction func btnSendColor(_ sender: Any) {
    dismiss(self)     }
@IBAction func btnCancel(_ sender: Any) {
    dismiss(self)     }
```

実行します。select color ボタンを押すと color 選択の補助ウィンドウが表示されます。send color または cancel ボタンで主ウィンドウに戻ります。

では、主ウィンドウの color テキストフィールドにある文字列を補助ウィンドウに送って、補助ウィンドウ上のラベルに表示してみます。ViewContoroller に関数 prepare を書きます。

```
override func prepare(for segue: NSStoryboardSegue, sender: Any?) {   //...(1)
        if let strSegue = segue.identifier {                                    //...(2)
            if  strSegue == "ColorSelect" {                                     //...(3)
                (segue.destinationController as!
                ColorSelectViewController).representedObject
                 = ("color",txtColor.stringValue)                           //...(4)
            }        }     }
```

ColorSelectViewController に representedObjec と viewWillAppear を書きます。

```
var informationFromMain = ("", "")                                  //...(5)
..........
override var representedObject: Any?{
        didSet{
            informationFromMain = representedObject as! (String, String)     //---(6)
        }    }
override func viewWillAppear() {                                            //...(7)
        lblInformationFromMain.stringValue =
         "color:" + informationFromMain.1 + " from:" + informationFromMain.0    }
```

コード（１）は ViewContoroller で設定された ActionSegue、select color ボタンが押されたときに起動される関数 prepare です。ActionSegue による画面遷移を実現するために用意する関数です。（２）（３）は segue が複数ある場合に identifier で区別しています。（４）は、segue 対象コントローラーを ColorSelectViwController にして、そこに用意されている変数 representedObjec 渡したいデータをタプルで代入しています。representedObjec は外部クラスからアクセス可能なプロパティです。

コード（５）は受け取ったデータを保持しておくための変数で、（６）でデータを受け取っています。（７）は補助画面が表示される前にラベルにデータをセットしています。color のテキストフィールドに何か文字を入れて、select color ボタンを押してみます。補助ウィンドウに渡されたデータが表示されます。

補助ウィンドウからデータを受け取ります。ColorSelectViwControlle の「親 ViewController」である ViewController を presentingViewController[*17]プロパティを使って取得し、ViewController の関数の引数に返すデータを入れます。データを返してもらう親 View の ViewController に関数 passBackData を用意します。

```
func passBackData(color:String){
        txtColor.stringValue = color     }
```

データを戻す ColorSelectViewController の関数 btnSendColor にコードを付け加え次のようにします。コード（１）は以前の parent プロパティを使ったものと似ています。

[*17] swift4 では presenting でした

```
@IBAction func btnSendColor(_ sender: Any) {
       let mainVC = presentingViewController as! ViewController      //...(1)
       mainVC.passBackData(color: cmbColor.itemObjectValue(at: Color.indexOfSelectedItem)
              as! String )
       dismiss(self)
   }
```

実行して、データをやりとりします。

もう一つ補助ウィンドウを作って select line type の ActionSegue を実装してみます。手順などは ColorSelectViwControll のものほぼ同じで、以下の通りです。

1. 新しい View を Storyboad 上におき、selectLineType ボタンからこの追加した View へコンロール・ドラッグ。ActionSegue の Sheet を選択。矢印の途中のアイコンをクリック、属性インスペクタの StoryboardSegue に identifier の欄に LineTypeSelect を入力
2. LineTypeSelectViewContoroller.swift の名前で Cocoa クラスファイルを作成, LineTypeSelectViewContoroller クラスをを追加した ViewController の CustomView にする。
3. LineTypeSelectViewContoroller 上にラベルと comboBox、２つのボタンをおく。（図を参照）comboBox の属性インスペクタの item を「solid」「dotted line」「double line」にする。Action 接続、Outlet 接続は次の通り。

```
@IBOutlet weak var lblInformationFromMain: NSTextField!
@IBOutlet weak var cmbColor: NSComboBox!
@IBAction func btnSendLineType(_ sender: Any) {
    dismiss(self)     }
@IBAction func btnCancel(_ sender: Any) {
    dismiss(self)     }
```

4. ViewContoroller に関数 prepare にコードを追加。２つの ActionSegue を判別する。

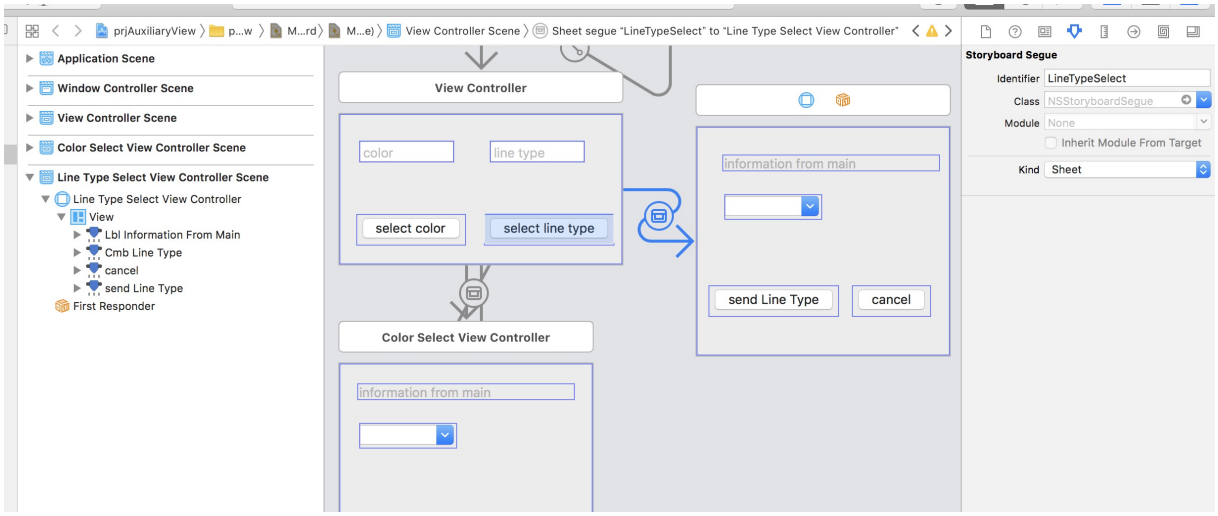
```
override func prepare(for segue: NSStoryboardSegue, sender: Any?) {
      if let strSegue = segue.identifier {
          if  strSegue == "ColorSelect" {
        ..................

          }else if strSegue == "LineTypeSelect"{                    //以下５行追
加
              (segue.destinationController as! LineTypeSelectViewController).representedObjec
          }                                                                          //
ここまで
      }
   }
```

5. LineTypeSelectViewController に representedObjec と viewWillAppear を書く。ColorSelectView-Controller との違いは、viewWillAppea 内の文字列で、「”color”」を「”line type”」としているだけ。
6. 親 View の ViewController に関数 passBackData を用意。（引数の名前が違うだけ）

```
func passBackData(line_type:String){
        txtLineType.stringValue = line_type }
```

7. LineTypeSelectViewController の関数 btnSendLine のコードを次のようにします。（passBackData の引数がことなるだけ）

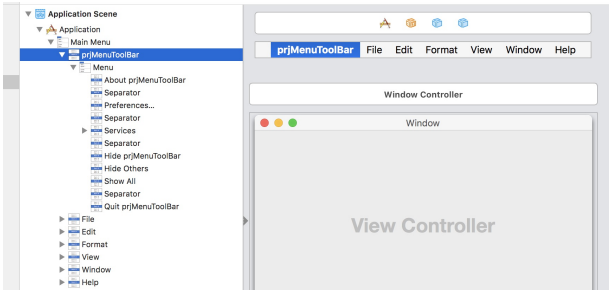
```
@IBAction func btnSendColor(_ sender: Any) {
        let mainVC = presentingViewController as! ViewController      //...(1)
        mainVC.passBackData(line_type: cmbColor.itemObjectValue(at: Color.indexOfSelectedItem)
                as! String )
        dismiss(self)
    }
```



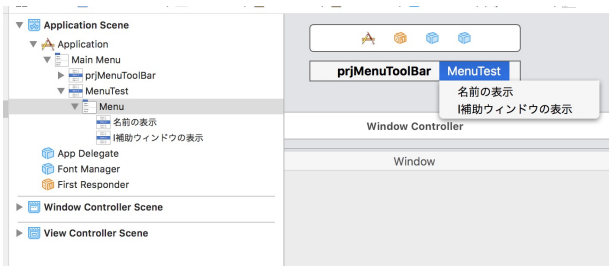
5　メニューとツールバー

5.1　メニュー

5.1.1　メニューの削除と追加



メニュー・アイテムの削除は、削除したいアイテムを選択し BackSpace キーで削除されます。ドキュメント構成図の ApplicationScene にある prjMenuToolBar を選択し（Storyboard の上部にあるメニューの中でも良い）、その中の「About prjMenuToolBar」と「QuitprjMenuToolBar」以外のアイテムを全部削除してください。このメニューの右にあるメニューも削除してください。サブメニューを開かずに選択・バックスペースで全部のアイテムが削除されます。

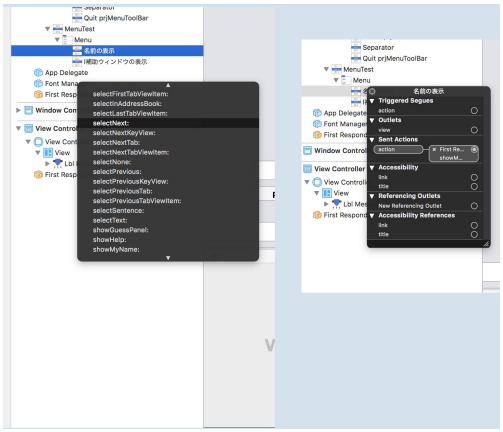


では、メニューアイテムを追加します。オブジェクト・ライブラリーから SubMenuItem 選んで MainMenu にドラッグ追加します。追加直後は Menu と表示され、Item を 1 個持っています。この Menu に Item を追加します。オブジェクト・ライブラリーから MenuItem 選んでドラッグ追加します。追加したメニューのタイトルは、それぞれの属性インスペクタの Title 欄に禁輪できます。「MenuTest」とし、その下の 2 つの Item のタイトルは、「名前の表示」、「補助ウィンドウの表示」としておきます

5.1.2　メニュー・アクション

ViewContorller にラベルを追加し、名前 lblMessaage で Outlet 接続します。このラベルに、メニュー「名前の表示」のクリックで、「梅屋」とでも表示させることにします。

図の (a) か (b) のオレンジ色のキュービック・アイコン (FirstResponder) をクリックします。Xcode 画面右端のインスペクタに FirstResponder が現れるので UserDefined で接続する関数を記入します。「＋」をクリックし追加された行の action をクリック、編集します。あとで、ViewController 内に関数 showMyName を作りますが、ここで「showMyName:」としておくことで接続の準備ができます。ここの名前には末尾にコロンがついています。id のところはそのままにしておきます。

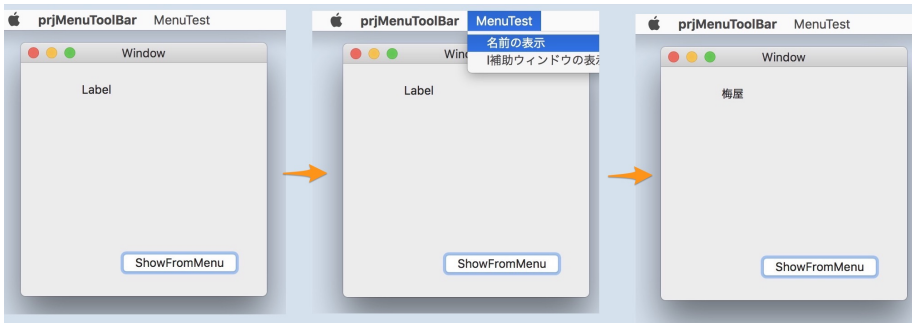


準備ができたので、図の（1）の矢印部分か（2）かどちらでも良いのですが、メニュ・アイテム「名前の表示」から「FirstResponder」にコントロール・ドラッグします。関数名が表示されるので、そのなかか showMyName:を探し選択します。このののちメニュ・アイテム「名前の表示」を右クリックすると SendAction のところに、今の設定がみられます。

ViewController 内に showMyName のコードを記入します。

```
@IBAction func showMyName(_ sender:Any){
        lblMessage.stringValue = "梅屋"    }
```

実行します。「MenuTest」をクリックし、「名前の表示」をクリックするとラベルに「梅屋」が表示されます。



5.1.3 Menu と ActionSegue

前の節での ActionSegue で補助ウィンドウの表示と同じようにやってみます。新しい ViewController を Storyboard に追加します。メニュー・アイテム「補助ウィンドウの表示」から追加した View へコントロール・ドラッグ。ActionSegue の Show を選択。矢印の途中のアイコンをクリック、属性インスペクタの StoryboardSegue に identifier の欄に fromMenu を入力。

Cocoa クラスファイルを ShowViewController.swift の名前で、プロジェクトに追加します。このクラスを追加した View の CustomView にします。これでメニュから補助ウィンドウは表示できおるのですが、データの受け渡しができません。ここがよくわからないところなのですが、ボタンからの ActionSegue はデータの受け渡しもできたので、メニューからボタンからの（を介して）Segue で補助ウィンドウとのやりとりを実現しました。(^_;)

では、その手順です。主ウィンドウの View 上にボタンを一つ配置し、その属性インスペクタでタイトルは「ShowFromMenu」、下の View の項目の「Hidden」にチェクを入れて隠すようにします。ViewController クラスに Action 接続しておきます。前に作ったメニュー・アイテム「補助ウィンドウの表示」からの ActionSegue は削除し、このボタンから ShowViewController へ ActionSegue を作ります。StoryboardSegue の identifier は「fromMenu」とし、kind は「Sheet」とします。ViewController クラスのコードは次のようになります。

```
override func prepare(for segue: NSStoryboardSegue, sender: Any?) {
        if let strSegue = segue.identifier {
            if  strSegue == "fromMenu" {
                (segue.destinationController as! ShowViewController).representedObject = ("fromMenu"
             }         }     }
@IBAction func btnShowFromMenu(_ sender: Any) {
    performSegue(withIdentifier: "fromMenu", sender: nil)}
func passBackData(data:String){
        lblMessage.stringValue = data    }
```

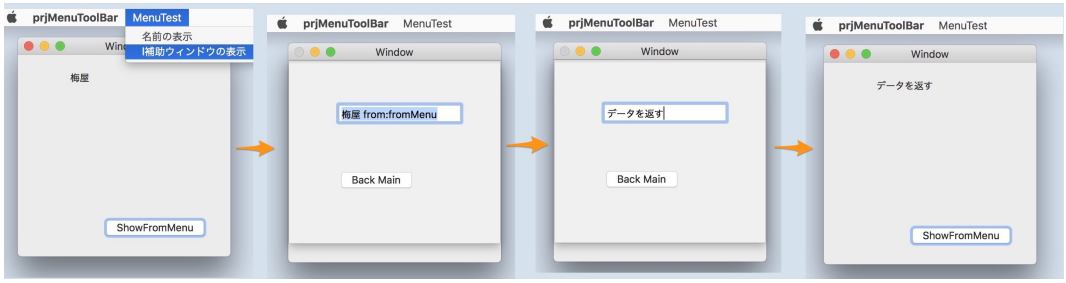
FirstResponder に btnShowFromMenu:を UserDefined の欄に追加します。メニュー・アイテム「補助ウィンドウの表示」から FirstResponder にコントロール・ドラッグし表示されたものの中から btnShowFromMenu:を選択します。

新しい View（ShowViewController）上に TextField と Button を一つずつ配置します。ViewController のコードを次のようにします。

```
 var informationFromMain = ("", "")
override func viewDidLoad() {
        super.viewDidLoad()    }
override var representedObject: Any?{
        didSet{
            informationFromMain = representedObject as! (String, String)        }    }
override func viewWillAppear() {
        txtMessage.stringValue =   informationFromMain.1 + " from:" + informationFromMain.0
```

```
    }

@IBOutlet weak var txtMessage: NSTextField!
@IBAction func btnBackMain(_ sender: Any) {
        let mainVC = presentingViewController as! ViewController
        mainVC.passBackData(data: txtMessage.stringValue)
        dismiss(self)    }
```

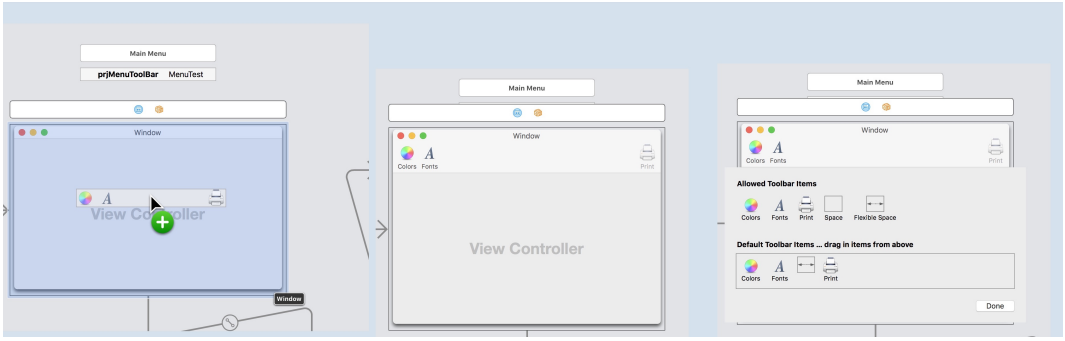
実行して、メニュからの補助ウィンドウとのデータのやり取りを確認してください。



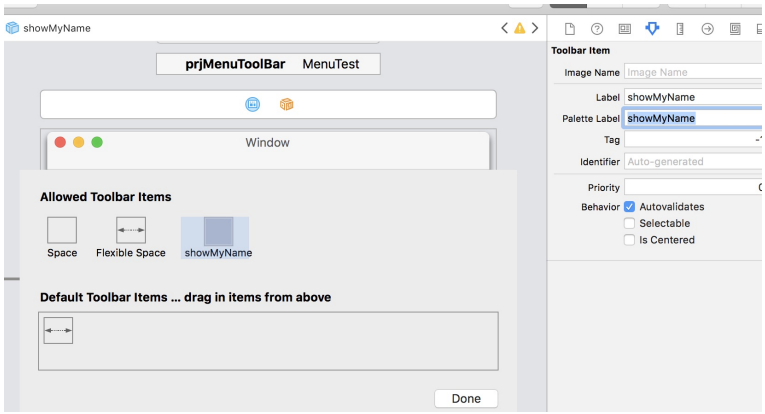
5.2　ツール・バー

5.2.1　ツール・バーの追加と削除

prjMenuToolBar を使って、ツール・バーについて書いていきます。ツール・バーは、はじめにライブラリーから追加しなければなりません。ライブラリーから Toolbar をドラッグします。WindowControllerScene のほうにもってきます。ViewControllerScene ではありません。左の図のようになり、中央の図のようにツール・バーが追加されました。右の図はツール・バーをクリックして、すでにある（用意されている）アイテム（上部の AllowedToolbarItems）と現在セットされているアイテム（下部の DefaultToolbatItems...）を表示させたものです。せっとされているアイテムを削除するには下部のアイコンを選択してバックスペースで削除できます。用意されているアイテム自体の削除は上部のアイコンの削除で可能です。上部にある用意されているアイテムで「Color」「Fonts」「Print」を削除してください。

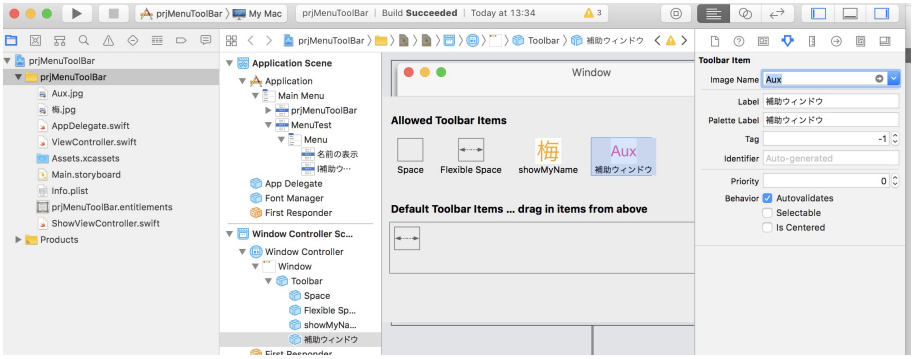


アイテムの追加は、ライブラリから ImageToolbarItem を用意されているアイテムがある上部にドラッグします。その属性インスペクタで Label と PaletteLabel を「showMyName」にします。Image も設定できます。



ImageName の欄で選択できます。ここにあるものではなく、自分独自のものを使いたいときは、プロジェクトにイメージファイルを追加すれば良いのです。

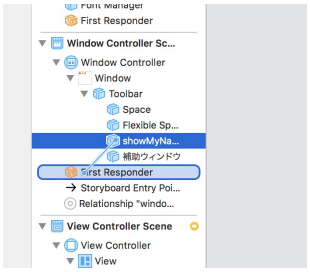
もう一つアイテムを追加して、Label と PaletteLabel を「補助ウィンドウ」にします。次の図は、プロジェクトにイメージファイル「梅.jpg」と「Aux.jpg」を加え、それぞれのツール・バーアイテムのイメージを設定したものです。



新たに用意したアイテムをセットするため下部の DefaultToolbatItems.. にドラッグします。左から右に表示順になるますが、ドラッグ・ドロップで位置を変えられます。実行すると、ツール・バーアイコンが表示されます。

5.2.2　ツール・バー　アクション

WindowControllerScene の中を開いて、Toolbar の中の showMyName から FirstResponders にコントロール・ドラッグします。表示されたものの中から showMyName を選びます。これでメニュー「名前の表示」と同様の処理になります。



ツール・バーから「補助ウィンドウ」の表示もメニューと同じように「隠ボタン」を介して処理します。ViewController 上にボタンを、タイトルは「ShowFromToolbar」、「Hidden」にチェックを入れ、ViewController クラス内に次のコードを記入した Action 接続しておきます。このボタンから ShowViewController へ ActionSegue を設定し、StoryboardSegue の identifier は「fromToolbar」とします。FirstResponders に btnShowFromToolbar:を追加します。ツール・バー　アイテム「補助ウィンドウ」から FirstResponders にコントロール・ドラッグし、showFromToolbar を選びます。ViewCotroller クラスのコードは次のようになります。

```
 override func prepare(for segue: NSStoryboardSegue, sender: Any?) {
        if let strSegue = segue.identifier {
            if  strSegue == "fromMenu" {
                (segue.destinationController as! ShowViewController).representedObject = ("fromMenu"
            }else if strSegue == "fromToolbar"{
                (segue.destinationController as! ShowViewController).representedObject = ("fromToolb
@IBAction func btnShowFromMenu(_ sender: Any) {
        performSegue(withIdentifier: "fromMenu", sender: nil)    }
@IBAction func btnShowFromToolbar(_ sender: Any) {
        performSegue(withIdentifier: "fromToolbar", sender: nil)    }
```

6　印刷処理

Swift もよくわかってないので、印刷処理に関するドキュメントやサンプル・コードを読んでもよくわかりません。ここでは、例によって「印刷処理で、とにかく動いた」ものを書いていきたいと思います。

6.1　ページ処理のない印刷処理

表示しているものを単に印刷するだけなら、次のコードのように適当な View を渡していけばできます。

```
let printInfo = NSPrintInfo.shared                                  //...(1)
let op = NSPrintOperation(view: PrintingView, printInfo: printInfo) //...(2)
op.run()                                                            //...(3)
```

コード（1）でプリンターのデフォルト設定を受け取ります。このコードの次に

```
printInfo.isHorizontallyCentered = true
```

などのように設定をすることができます。（2）の引数 PrintingView で印刷する NSView を指定すれば準備終了です。（3）で いつもの印刷処理のウィンドウが表示されます。ここでの指定項目を追加したければ、（2）と（3）の間に

```
op.printPanel.options = NSPrintPanel.Options.showsPaperSize
op.printPanel.options = NSPrintPanel.Options.showsOrientation
```

とでも入れれば良いのです。

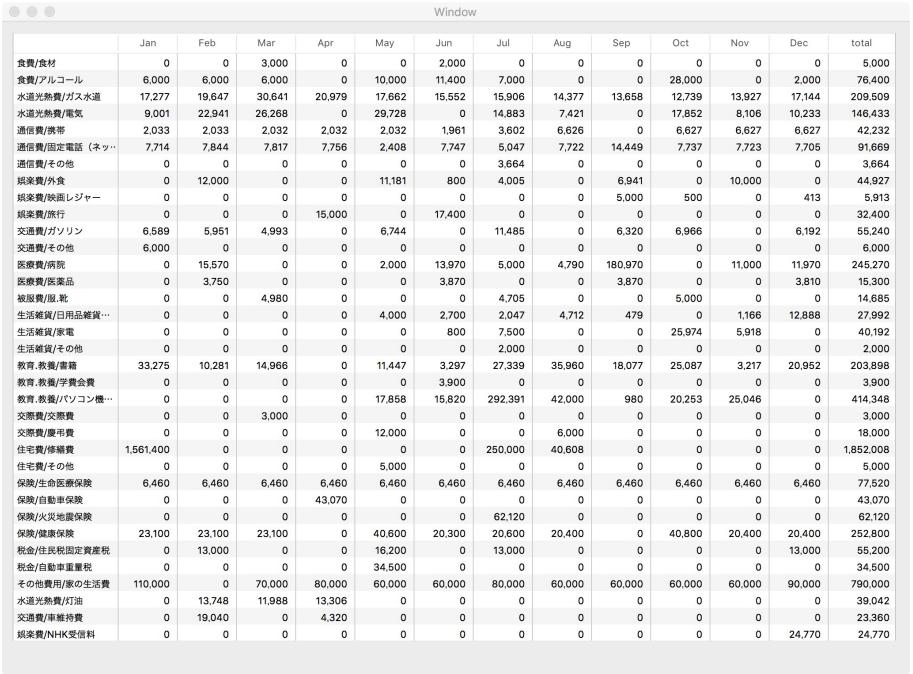
ではこの方法での印刷処理のためのプロジェクト prjPrintingWithView を作ります。テストデータを CSV ファイルで用意して、class DataModel の中で以下のように準備します。

```
import Cocoa



class DataModel{
    static let sharedInstance = DataModel()
    var testDataName:[String] = []
    var testDataAmount:[[Int]] = []
    private init(){
        readTestData()
    }
    func readTestData(){
        let cvsPath = Bundle.main.path(forResource: "testData", ofType: "csv")
        do{
            let csvStrData = try String(contentsOfFile: cvsPath!, encoding:String.Encoding.utf8)
            for line in csvStrData.split(separator: "\n"){
```

```
                var elm = line.split(separator: ",")
                testDataName.append(String(elm[0]))
                var X:[Int] = []
                for i in 1...13{
                    X.append(Int(String(elm[i].trimmingCharacters(in: .whitespaces)))!)
                }
                testDataAmount.append(X)
            }
        } catch let error {
            print(error)
        }  } }
```

TableView を ViewControllerScene 上において、テストデータを表示させます。この TableView を前のコード（２）の PrintingView にして印刷処理をします。これはテーブルの一部が印刷されるだけでダメでした。

そこで、Custom View をまず ViewControllerScene 上において、その中に TableView をおきます。これら３つの View のサイズは調整しながら決めます。これでテストデータが全て表示できるサイズにしただけです。



	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	total
食費/食材	0	0	3,000	0	0	2,000	0	0	0	0	0	0	5,000
食費/アルコール	6,000	6,000	6,000	0	10,000	11,400	7,000	0	0	28,000	0	2,000	76,400
水道光熱費/ガス水道	17,277	19,647	30,641	20,979	17,662	15,552	15,906	14,377	13,658	12,739	13,927	17,144	209,509
水道光熱費/電気	9,001	22,941	26,268	0	29,728	0	14,883	7,421	0	17,852	8,106	10,233	146,433
通信費/携帯	2,033	2,033	2,032	2,032	2,032	1,961	3,602	6,626	0	6,627	6,627	6,627	42,232
通信費/固定電話（ネッ…	7,714	7,844	7,817	7,756	2,408	7,747	5,047	7,722	14,449	7,737	7,723	7,705	91,669
通信費/その他	0	0	0	0	0	0	3,664	0	0	0	0	0	3,664
娯楽費/外食	0	12,000	0	0	11,181	800	4,005	0	6,941	0	10,000	0	44,927
娯楽費/映画レジャー	0	0	0	0	0	0	0	0	5,000	500	0	413	5,913
娯楽費/旅行	0	0	0	15,000	0	17,400	0	0	0	0	0	0	32,400
交通費/ガソリン	6,589	5,951	4,993	0	6,744	0	11,485	0	6,320	6,966	0	6,192	55,240
交通費/その他	6,000	0	0	0	0	0	0	0	0	0	0	0	6,000
医療費/病院	0	15,570	0	0	2,000	13,970	5,000	4,790	180,970	0	11,000	11,970	245,270
医療費/医薬品	0	3,750	0	0	0	3,870	0	0	3,870	0	0	3,810	15,300
被服費/服,靴	0	0	4,980	0	0	0	4,705	0	0	5,000	0	0	14,685
生活雑貨/日用品雑貨…	0	0	0	0	4,000	2,700	2,047	4,712	479	0	1,166	12,888	27,992
生活雑貨/家電	0	0	0	0	0	800	7,500	0	0	25,974	5,918	0	40,192
生活雑貨/その他	0	0	0	0	0	0	2,000	0	0	0	0	0	2,000
教育.教養/書籍	33,275	10,281	14,966	0	11,447	3,297	27,339	35,960	18,077	25,087	3,217	20,952	203,898
教育.教養/学費会費	0	0	0	0	0	3,900	0	0	0	0	0	0	3,900
教育.教養/パソコン機…	0	0	0	0	17,858	15,820	292,391	42,000	980	20,253	25,046	0	414,348
交際費/交際費	0	0	3,000	0	0	0	0	0	0	0	0	0	3,000
交際費/慶弔費	0	0	0	0	12,000	0	0	6,000	0	0	0	0	18,000
住宅費/修繕費	1,561,400	0	0	0	0	0	250,000	40,608	0	0	0	0	1,852,008
住宅費/その他	0	0	0	0	5,000	0	0	0	0	0	0	0	5,000
保険/生命医療保険	6,460	6,460	6,460	6,460	6,460	6,460	6,460	6,460	6,460	6,460	6,460	6,460	77,520
保険/自動車保険	0	0	0	43,070	0	0	0	0	0	0	0	0	43,070
保険/火災地震保険	0	0	0	0	0	0	62,120	0	0	0	0	0	62,120
保険/健康保険	23,100	23,100	23,100	0	40,600	20,300	20,600	20,400	0	40,800	20,400	20,400	252,800
税金/住民税固定資産税	0	13,000	0	0	16,200	0	13,000	0	0	0	0	13,000	55,200
税金/自動車重量税	0	0	0	0	34,500	0	0	0	0	0	0	0	34,500
その他費用/家の生活費	110,000	0	70,000	80,000	60,000	60,000	80,000	60,000	60,000	60,000	60,000	90,000	790,000
水道光熱費/灯油	0	13,748	11,988	13,306	0	0	0	0	0	0	0	0	39,042
交通費/車維持費	0	19,040	0	4,320	0	0	0	0	0	0	0	0	23,360
娯楽費/NHK受信料	0	0	0	0	0	0	0	0	0	0	0	24,770	24,770

メニューバーに「印刷処理」、そのサブメニュに「TableView の印刷」作ります。次の printOutTableView を FirstResponder に追加して、このメニュから起動できるようにします。

```
@IBAction func printOutTableView(_ sender:Any){
        let printInfo = NSPrintInfo.shared
        printInfo.scalingFactor = CGFloat(0.55)                                    //...(1)
```

```
        printInfo.orientation = NSPrintInfo.PaperOrientation(rawValue: 1)!   //...(2)
        printInfo.paperName = NSPrinter.PaperName("A4")
        let op = NSPrintOperation(view: cvwTestData, printInfo: printInfo)
        op.printPanel.options = NSPrintPanel.Options.showsPaperSize
        op.printPanel.options = NSPrintPanel.Options.showsPreview
        op.run()
    }
```

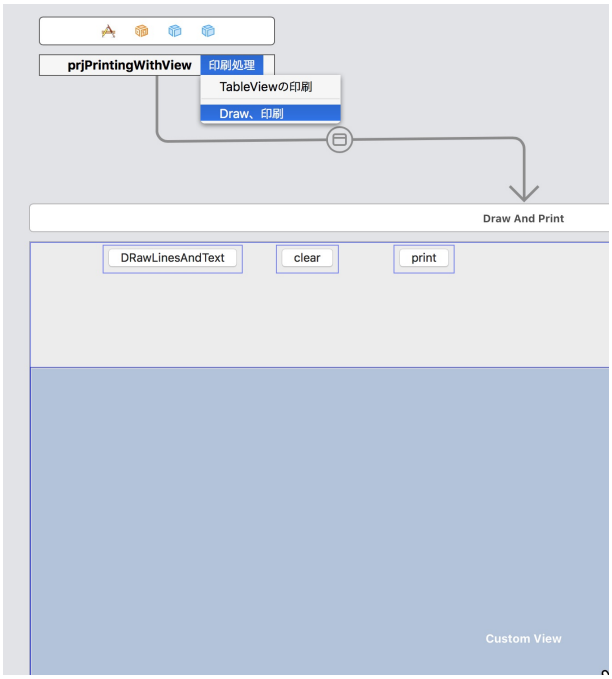
なんとか A4 横に入るようにコード（１）で縮小５５％、コード（２）で Lanscape の設定をしています。

表示印刷するものが固定されているものならばページ処理もしないで”無理やり”でも、こんなふうに処理できるのかもしれません？

では、Custom View に TableVew をおかないで、書き込む方法はどうでしょうか。次の節で実験してみます。

6.2 Custom View に書き込む

prjPrintingWithView に ViewController を追加し、そこに CustomView をおき、文字と直線など図形を書き込みます。図のようにメニューに「Draw、印刷」を追加します。追加した View 上にボタンを 3 個配置します。それぞれのサイズは適当に調整します。 3 個のボタンの Action 接続などは次のコードを見てください。このコードは、DrawAndPrint.swift の名前の Cocoa Class ファイルです。 追加した ViewController の Custom Class にします。コードは次の通り。




```
import Cocoa
class DrawAndPrint: NSViewController {
    let DM = DataModel.sharedInstance                                   //...(0)
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do view setup here.
    }
    @IBOutlet weak var cvwDrawPrint: NSView!                            //...(1)
    @IBAction func btnDrawLinesAndTexts(_ sender: Any) {          //...(2)
```

```
        cvwDrawPrint.addSubview(DrawLinesTexts(frame: NSRect(x:0,  y:0,
        width: cvwDrawPrint.bounds.width, height: cvwDrawPrint.bounds.height),
        dataName: DM.testDataName, dataAmount: DM.testDataAmount))
    }
    @IBAction func btnClear(_ sender: Any) {                              //...(3)
        for subvw in cvwDrawPrint.subviews {
            subvw.removeFromSuperview()
        }
    }
    @IBAction func btnPrint(_ sender: Any) {                               //...(4)
        let printInfo = NSPrintInfo.shared
        printInfo.scalingFactor = CGFloat(0.55)
        printInfo.orientation = NSPrintInfo.PaperOrientation(rawValue: 1)!
        printInfo.paperName = NSPrinter.PaperName("A4")
        let op = NSPrintOperation(view: cvwDrawPrint, printInfo: printInfo)
        op.printPanel.options = NSPrintPanel.Options.showsPaperSize
        op.printPanel.options = NSPrintPanel.Options.showsPreview
        op.run()
    } }

class DrawLinesTexts:NSView{                                               //...(5)
    var dataName:[String] = []
    var dataAmount:[[Int]] = []
    init(frame:NSRect, dataName:[String], dataAmount:[[Int]]){
        super.init(frame: frame)
        self.dataName = dataName
        self.dataAmount = dataAmount
    }
    required init?(coder decoder: NSCoder) {
        fatalError("init(coder:) has not been implemented") }

    override func draw(_ r: NSRect) {                                    //...(5.1)
        //draw lines
        let path = NSBezierPath()                                         //...(5.2)
        path.lineWidth = 3.0                                               //...(5.3)
        NSColor.black.setStroke()                                        //...(5.4)
        let x_right = r.width - CGFloat(10)
        let x_left = CGFloat(10)
        let name_width = CGFloat(160)
        let total_width = CGFloat(100)
```

```
let amount_count = 9
let x_width = CGFloat((x_right - x_left - name_width - total_width) / 12.0)
let y_height = CGFloat((r.height - 50) / 36.0)

path.move(to: NSPoint(x: 5, y: r.height-5)              //...(5.5)
path.line(to: NSPoint(x: r.width-5, y: r.height-5))    //...(5.6)
path.line(to: NSPoint(x: r.width-5, y: 5))
path.line(to: NSPoint(x: 5, y: 5))
path.line(to: NSPoint(x: 5, y: r.height - 5))
path.move(to: NSPoint(x: 5, y: r.height - 2 - y_height * 2))
path.line(to: NSPoint(x: r.width-5, y: r.height - 2 - y_height * 2 ))
path.stroke()                                                    //...(5.7)
path.lineWidth = 1.0

var yy = r.height - y_height * 2
for row in 1...7{
    let y = yy - y_height * CGFloat(row * 5)
    path.move(to: NSPoint(x: x_left, y:  y))
    path.line(to: NSPoint(x: x_right, y: CGFloat(y)))
    path.stroke()
}

let dashes:[CGFloat] = [3,2]                                                    //...(5.8)
path.setLineDash(dashes, count: dashes.count, phase: 0)     //...(5.9)
for row in 1...35{
    let y = yy - y_height * CGFloat(row)
    path.move(to: NSPoint(x: x_left, y:  y))
    path.line(to: NSPoint(x: x_right, y: CGFloat(y)))
    path.stroke()
}

//draw texts
yy = r.height - 10 - y_height * 1
"   name".draw(at:CGPoint(x:x_left, y:yy),
           withAttributes:[NSAttributedString.Key.foregroundColor:NSColor.blue,
           NSAttributedString.Key.font: NSFont.systemFont(ofSize: 10),])       //...(5.10)
var xx = x_left + name_width + 6
for i in 1...12{
    let s = String(format:"%3d 月",i)
    s.draw(at:CGPoint(x:xx, y:yy),
```

```
             withAttributes:[NSAttributedString.Key.foregroundColor: NSColor.blue,
            NSAttributedString.Key.font: NSFont.systemFont(ofSize: 10),])
            xx += x_width
        }
        "   合計".draw(at:CGPoint(x:xx, y:yy),
         withAttributes:NSAttributedString.Key.foregroundColor: NSColor.blue,
        NSAttributedString.Key.font: NSFont.systemFont(ofSize: 10),])
        yy = r.height - y_height * 2
        for (row, amounts) in dataAmount.enumerated(){
            let y = yy - y_height * CGFloat(row + 1)
            String(dataName[row]).draw(at:CGPoint(x:x_left, y:y),
             withAttributes:[NSAttributedString.Key.foregroundColor: NSColor.blue,
             NSAttributedString.Key.font: NSFont.systemFont(ofSize: 10),])
            var x = x_left + name_width
            for amount in amounts{
                let s = ("          " + amount.withComma).suffix(amount_count)
                String(s).draw(at:CGPoint(x:x, y:y),
                 withAttributes:[NSAttributedString.Key.foregroundColor: NSColor.blue,
                 NSAttributedString.Key.font: NSFont.systemFont(ofSize: 10),])
                x += x_width
            } } }
    func int2stringWithComma(integer:Int, digits:Int  = 10) -> String{      //...(6)
        var s = ""
        for _ in 1...digits{ s += " " }
        let formatter = NumberFormatter()
        formatter.numberStyle = .decimal
        s += formatter.string(from: integer as NSNumber) ?? "error"
        return String(s.suffix(digits))
    }

}
```

(0) はテストデータを参照するためのものです。

(1) は CustonView の Outlet 接続したものです。

(2) は「DrawLinesAndText」を押すと起動され、DrawLinesAndTexts クラスで描画されたものを CustomView に Subview として追加します。ここ描画クラスに書き込む 2 つのデータを渡しています。

(3) は CustomView にある Subview を削除して「画面をクリア」しています。

(4) は前にも使用したコードで、印刷処理をします。

(5) が実践や点線を引き、タイトルや渡されたテキストデータを CustomView に書き込むクラスです。

クラスの初期化 init ののち、このコードの下にある (5.1) 以下のコードで書き込みをします。

(5.1)、draw を override して書き込み処理を記述します。

(5.2) で NSBezierPath クラスのオブジェクトを設定し、これで図形描画をします。

(5.3) は線の太さを指定し、(5.4) では色を指定しています。

(5.5) で描画始点を設定し、(5.6) の点まで線を引きます。

(5.7) でこれまで描画された図形を表示します。

(5.8) は点線のタイプの指定で、ここでは「塗り 3、空白 2」のパターンで点線を指定しています。(5.9) でこのパターンを設定しています。「phase」はよくわかりません。

(5.10) でテキスト（文字列）を指定した位置、色、フォントサイズなどで書き込みます。

(6) は 3 桁ごとのコンマと桁数を指定して数値の文字列への変換するものです。

実行すると次の画面が表示され、「print」ボタンを押すと印刷処理ができます。

6.3　PDF の作成

印刷処理には、やはりページ処理が必要なのですがよくわかりません。ネットで情報を集めると「PDF」処理が多くありました[*18]。できるかどうかわかりませんが、これで PDF ファイルが作れれば実際の印刷は PDFreader などに”任せればいい”のですから・・・・・・

prjCreatePDF を作り実験します。テストデータは前のものと同じものを使うことにします。図のようなレイアウトで、PDF ファイルを作ることを目標にします。用意したデータで２ページになるようにして、ページ処理を作ってみました。

このレイアウトでは１ページに２０個のデータを書き込みます。２０個でページ１、次の２０個でページ２、という具合に処理するのがページ処理にいなります。

このレーアウトの１ページ分を描画するクラスを PDFPage をスーパークラスとして作成します。コードは”ごちゃごちゃ”しますが（印刷処理はこんな歩になることが多い？）、処理の構造は簡単です。

名称	１月	２月	３月	４月	５月	６月
食費/食材	0	0	3,000	0	0	2,000
食費/アルコール	6,000	6,000	6,000	0	10,000	11,400
水道光熱費/ガス水道	17,277	19,647	30,641	20,979	17,662	15,552
水道光熱費/電気	9,001	22,941	26,268	0	29,728	0
通信費/携帯	2,033	2,033	2,032	2,032	2,032	1,961
通信費/固定電話（ネット）	7,714	7,844	7,817	7,756	2,408	7,747
通信費/その他	0	0	0	0	0	0
娯楽費/外食	0	12,000	0	0	11,181	800
娯楽費/映画レジャー	0	0	0	0	0	0
娯楽費/旅行	0	0	0	15,000	0	17,400
交通費/ガソリン	6,589	5,951	4,993	0	6,744	0
交通費/その他	6,000	0	0	0	0	0
医療費/病院	0	15,570	0	0	2,000	13,970
医療費/医薬品	0	3,750	0	0	0	3,870
被服費/服.靴	0	0	4,980	0	0	0
生活雑貨/日用品雑貨　衣料品	0	0	0	0	4,000	2,700
生活雑貨/家電	0	0	0	0	0	800
生活雑貨/その他	0	0	0	0	0	0
教育.教養/書籍	33,275	10,281	14,966	0	11,447	3,297
教育.教養/学費会費	0	0	0	0	0	3,900

７月	８月	９月	１０月	１１月	１２月	合計
0	0	0	0	0	0	5,000
7,000	0	0	28,000	0	2,000	76,400
15,906	14,377	13,658	12,739	13,927	17,144	209,509
14,883	7,421	0	17,852	8,106	10,233	146,433
3,602	6,626	0	6,627	6,627	6,627	42,232
5,047	7,722	14,449	7,737	7,723	7,705	91,669
3,664	0	0	0	0	0	3,664
4,005	0	6,941	0	10,000	0	44,927
0	0	5,000	500	0	413	5,913
0	0	0	0	0	0	32,400
11,485	0	6,320	6,966	0	6,192	55,240
0	0	0	0	0	0	6,000
5,000	4,790	180,970	0	11,000	11,970	245,270
0	0	3,870	0	0	3,810	15,300
4,705	0	0	5,000	0	0	14,685
2,047	4,712	479	0	1,166	12,888	27,992
7,500	0	0	25,974	5,918	0	40,192
2,000	0	0	0	0	0	2,000
27,339	35,960	18,077	25,087	3,217	20,952	203,898
0	0	0	0	0	0	3,900

[*18] Generating a PDF Document in Swift, Created By: Debasis Das (31-Jan-2016) このコードを参考にしました。

6.3.1 PDFPage をスーパークラスとする 1 ページ分のクラス

ViewController クラス上で TableView を使いテストデータを表示しています。PDF の作成はボタン「createPDF」から起動します。その Action 接続されたコードはは次のようになります。

```
@IBAction func btnCreatePDF(_ sender: Any) {
        let aPdfDoc = PDFDocument()                                  //...(1)
        var numberOfPages = DM.testDataName.count / numberOfDatasRowsPerPage //...(2)
        if DM.testDataName.count % numberOfDatasRowsPerPage > 0{
            numberOfPages += 1
        }
        for i in 0 ..< numberOfPages{                                                  //...(2.1)
            let startIndex = i * numberOfDatasRowsPerPage          //...(3)
            var endIndex = i * numberOfDatasRowsPerPage + numberOfDatasRowsPerPage
            if endIndex >  DM.testDataName.count{
                endIndex = DM.testDataName.count
            }

            let pdfDataName:[String] = Array(DM.testDataName[startIndex..<endIndex]) //...(4)
            let pdfDataAmount:[[Int]] = Array(DM.testDataAmount[startIndex..<endIndex])//...(5)
            let aPage = MyPDFPage(numberOfDatasRowsPerPage: numberOfDatasRowsPerPage,
                 numberOfRowsPerPage: numberOfRowsPerPage,
                  dataname: pdfDataName, dataamount: pdfDataAmount)     //...(6)
            aPdfDoc.insert(aPage, at: i)                      //...(7)

        }
        let FS = NSSavePanel()                                  //...(8)
        FS.canCreateDirectories = true
        FS.allowedFileTypes = ["pdf", "PDF"]
        if FS.runModal() == NSApplication.ModalResponse.OK{
            let fileURL = FS.url
            let strFS = (fileURL?.path)!
            print(strFS)
            aPdfDoc.write(toFile: strFS)   //...(9)
        }  }
```

- (1) で PDFDocument のインスタンスを pPdfDoc にしています。ここに (7) のコードでページ単位に追加していきます。
- (2) とその下の 3 行でページ数を計算しています。(2.1) のループで必要なページ数だけの処理を行っています。
- 1 ページ２０個のデータなので (3) 部分で 1 ページぶんのデータをスライスするインデックスを計算し

ています。

- (4)(5) で 1 ページ分をスライスしています。
- (6) の MyPDFPage が 1 ページ分の描画を行うところです。
- (7) は出来上がった 1 ページ分を追加しています。
- (8) から (9) までで保存ファ入りの指定と書き込みをしています。

では、1 ページ分の処理をする MyPDFPage クラスはどのようなものになるのでしょう。

```
import Cocoa
import Quartz        //...(1)

class MyPDFPage: PDFPage {
    ..................    //...(2)

    init(numberOfDatasRowsPerPage:Int, numberOfRowsPerPage:Int,
     dataname:[String], dataamount:[[Int]]){
        super.init()
        ..................  //...(3)
    }

    override func draw(with box: PDFDisplayBox) {    //...(4)
        super.draw(with: box)
        self.drawTableLines()
        self.drawTableData()
```

(2) では変数の定義、(3) では初期化がなされています。中心となるのは (1) の Quarts の import と (4) の draw の override[*19]です。(4) の中に描画する k ー度を書いていけば良いのです。ここでは罫線とデータの二つに分けています。罫線は次の関数を用意しておいて、これで書きます。

```
func drawLine(linewidth: CGFloat, from:NSPoint, to:NSPoint){
        let path = NSBezierPath()
        NSColor.darkGray.set()
        path.move(to: to)
        path.line(to: from)
        path.lineWidth = linewidth
        path.stroke()
    }
```

データの方は次のコードのようになります。

[*19] draw(with:) was deprecated in OS X 10.12 と警告が出されますが、動くので後で調べるとにします

```
let titleFont = NSFont(name:"Helvetica", size: 11.0)      //...(1)
var p0 = NSMakePoint(leftMargin + textInset + textInset,
                  self.pageHeight - verticalPadding - topMargin - rowHeight)   //...(2)
var s0Rect = NSMakeRect(p0.x, p0.y, name_totalWidth, rowHeight)       //...(3)
"  名称".draw(in: s0Rect,
              withAttributes: [NSAttributedString.Key.font: titleFont!])      //...(4)
```

(1) でフォントの属性設定、(2) で書き込み開始位置を決めます。(3) でその位置から幅、高さの長方形を (4) で書き込むのですが
文字列.draw(書き込む長方形エリア、フォントなどの属性)
となっています。これを繰り返すことになります。

マージンなどの書式定数は一つのファイルにまとめておきます。あとの細かなところは、次のコード全文を見てください。

6.3.2 class MyPDFPage:PDFPage

```
import Cocoa
import Quartz
class MyPDFPage: PDFPage {
    var dataName:[String] = []
    var dataAmount:[[Int]] = []
    var numberOfDatasRowsPerPage:Int = 0
    var numberOfRowsPerPage:Int  = 0
    var pageWidth = CGFloat(0)
    var pageHeight = CGFloat(0)

    init(numberOfDatasRowsPerPage:Int, numberOfRowsPerPage:Int,
           dataname:[String], dataamount:[[Int]]){
        super.init()
        self.numberOfDatasRowsPerPage = numberOfDatasRowsPerPage
        self.numberOfRowsPerPage = numberOfRowsPerPage
        self.dataName = dataname
        self.dataAmount = dataamount
        self.numberOfDatasRowsPerPage = numberOfDatasRowsPerPage
        self.numberOfRowsPerPage = numberOfRowsPerPage
        self.pageWidth = leftMargin + tableWidth + rightMargin
        self.pageHeight = topMargin + rowHeight * CGFloat( self.numberOfRowsPerPage)
                   + bottomMargin
    }
    override func draw(with box: PDFDisplayBox) {
        super.draw(with: box)
```

```
    self.drawTableLines()
    self.drawTableData()
}
func drawTableLines(){
    var p00 = NSMakePoint(leftMargin , self.pageHeight - topMargin)
    var p01 = NSMakePoint(leftMargin + tableWidth, self.pageHeight - topMargin)
    var p10 = NSMakePoint(leftMargin , self.pageHeight - topMargin)
    var p11 = NSMakePoint(leftMargin ,  self.pageHeight - topMargin
                      - rowHeight * CGFloat(numberOfDatasRowsPerPage + 1))
    drawLine(linewidth: CGFloat(2), from: p10, to: p11)
    p10.x += name_totalWidth; p11.x += name_totalWidth
    for _ in 1...2{
        drawLine(linewidth: CGFloat(2), from: p00, to: p01)
        for _ in 1...6{
            drawLine(linewidth: CGFloat(0.5), from: p10, to: p11)
            p10.x += columnWidth; p11.x += columnWidth
        }
        p00.y -= rowHeight; p01.y -= rowHeight
        drawLine(linewidth: CGFloat(2), from: p00, to: p01)
        for _ in 1...3{
            p00.y -= rowHeight * 5; p01.y -= rowHeight * 5
            drawLine(linewidth: CGFloat(0.5), from: p00, to: p01)
        }
        p00.y -= rowHeight * 5; p01.y -= rowHeight * 5
        drawLine(linewidth: CGFloat(2), from: p00, to: p01)
        p00.y -= rowHeight * CGFloat(spaceTables);
        p01.y -= rowHeight * CGFloat(spaceTables)
        p10.x = leftMargin + columnWidth; p11.x = leftMargin + columnWidth
        p10.y = self.pageHeight - topMargin
             - rowHeight *CGFloat(numberOfDatasRowsPerPage
             + 1 + spaceTables)
        p11.y = self.pageHeight - topMargin
                 - rowHeight * CGFloat((numberOfDatasRowsPerPage + 1) * 2
                   + spaceTables)
    }
    p10.x  = leftMargin + tableWidth; p11.x = leftMargin + tableWidth
    p10.y = self.pageHeight - topMargin
               - rowHeight * CGFloat(numberOfDatasRowsPerPage + 1 + spaceTables)
    p11.y = self.pageHeight - topMargin
         - rowHeight * CGFloat((numberOfDatasRowsPerPage + 1) * 2 + spaceTables)
```

```
    drawLine(linewidth: CGFloat(2), from: p10, to: p11)
}

func drawTableData(){
    //-----------------------------------------------------column title
    let titleFont = NSFont(name:"Helvetica", size: 11.0)

    var p0 = NSMakePoint(leftMargin + textInset + textInset,
                self.pageHeight - verticalPadding - topMargin - rowHeight)
    var s0Rect = NSMakeRect(p0.x, p0.y, name_totalWidth, rowHeight)
    "  名称".draw(in: s0Rect, withAttributes: [NSAttributedString.Key.font: titleFont!])
    p0.x += name_totalWidth
    var p1 = NSMakePoint(leftMargin + textInset + textInset,
               self.pageHeight - verticalPadding - topMargin
                - rowHeight * CGFloat( numberOfDatasRowsPerPage + 2 + spaceTables))
    for m in 1...6{
        let s1Rect = NSMakeRect(p0.x, p0.y, columnWidth, rowHeight)
        let s2Rect = NSMakeRect(p1.x, p1.y, columnWidth, rowHeight)
        let s1 = String(format: "%4d 月", m);let s2 = String(format: "%4d 月", m + 6)
        s1.draw(in: s1Rect, withAttributes: [NSAttributedString.Key.font: titleFont!])
        s2.draw(in: s2Rect, withAttributes: [NSAttributedString.Key.font: titleFont!])
        p0.x += columnWidth; p1.x += columnWidth
    }
    s0Rect = NSMakeRect(p1.x, p1.y, name_totalWidth, rowHeight)
    "       合計".draw(in: s0Rect, withAttributes: [NSAttributedString.Key.font: titleFont!])

    //----------------------------------------- name
     p0 = NSMakePoint(leftMargin + textInset,
                self.pageHeight  - verticalPadding - topMargin - rowHeight - rowHeight)
     p1 = NSMakePoint(leftMargin + textInset,
              self.pageHeight - topMargin
              - rowHeight * CGFloat( numberOfDatasRowsPerPage + spaceTables))
    let nameFont = NSFont(name:"Helvetica", size: 10.0)
    for s in dataName{
        let nameRect = NSMakeRect(p0.x, p0.y,  name_totalWidth,  rowHeight)
        s.draw(in: nameRect, withAttributes: [NSAttributedString.Key.font: nameFont!])
        p0.y -= rowHeight
    }
    //----------------------------------------- each data
    let amountFont = NSFont(name: "Helvetica", size: 8.0)
```

```
        p0 = NSMakePoint(leftMargin + name_totalWidth + textInset + textInset,
                self.pageHeight - verticalPadding - topMargin - rowHeight - rowHeight)
        p1 = NSMakePoint(leftMargin + textInset + textInset,
                    self.pageHeight - verticalPadding - topMargin
                    - rowHeight * CGFloat( numberOfDatasRowsPerPage + 2 + spaceTables + 1))
        for line in dataAmount{
            for i in 0...5{
                let a1Rect = NSMakeRect(p0.x, p0.y, columnWidth, rowHeight)
                let a2Rect = NSMakeRect(p1.x, p1.y , columnWidth, rowHeight)
                int2stringWithComma(integer: line[i]).draw(in: a1Rect,
                              withAttributes: [NSAttributedString.Key.font: amountFont!])
                int2stringWithComma(integer: line[i + 6]).draw(in: a2Rect,
                            withAttributes: [NSAttributedString.Key.font: amountFont!])
                p0.x += columnWidth; p1.x += columnWidth
            }
            let a0Rect = NSMakeRect(p1.x, p1.y, name_totalWidth, rowHeight)
            int2stringWithComma(integer: line[12]).draw(in: a0Rect,
                         withAttributes: [NSAttributedString.Key.font: amountFont!])
            p0.x = leftMargin + name_totalWidth + textInset + textInset
            p1.x = leftMargin + textInset + textInset
            p0.y -= rowHeight; p1.y -= rowHeight
        }
    }
    func drawLine(linewidth: CGFloat, from:NSPoint, to:NSPoint){
        let path = NSBezierPath()
        NSColor.darkGray.set()
        path.move(to: to)
        path.line(to: from)
        path.lineWidth = linewidth
        path.stroke()
    }
    func int2stringWithComma(integer:Int, digits:Int  = 10) -> String{
        var s = ""
        for _ in 1...digits{ s += " " }
        let formatter = NumberFormatter()
        formatter.numberStyle = .decimal
        s += formatter.string(from: integer as NSNumber) ?? "error"
        return String(s.suffix(digits))
    }
}
```

6.3.3 PDF レイアウト定数

```
import Foundation
let topMargin = CGFloat(40.0)
let leftMargin = CGFloat(20.0)
let rightMargin = CGFloat(20.0)
let bottomMargin = CGFloat (40.0)
let textInset = CGFloat(5.0)
let verticalPadding = CGFloat (10.0)
let rowHeight = CGFloat(25.0)
let columnWidth = CGFloat(70.0)
let name_totalWidth = CGFloat(160)
let tableWidth = name_totalWidth + columnWidth * 6

let numberOfDatasRowsPerPage = 20
let spaceTables = 5
let numberOfRowsPerPage = (numberOfDatasRowsPerPage + 1) * 2 + spaceTables
```

7　おわりに　sKakeibo

XOJO で書いていた家計簿ソフト uKakeibo.app を Swift4 で書き直すために色々と調べたりしてきました。mac の DeskTop アプリの参考書が少なかったので、ここまでに学んできたことをまとめてみました。他の言語でのプログラミングも同じですが、一通りの文法的なことを学んでも、実際に動くコードはなかなかかけません。「なになに　入門」とか「これでわかる　なんとか」と言ったものを読んでもです。ライブラリなどの知識、情報が必ず必要となるからです。このノートでは、そんなところに注意しながら書いてきたつもりです。一番役に立つものは”サンプルコード”だと思います。

uKakeibo を sKakeibo と改名して swift で書きました。”良い”サンプルコードではないかもしれませんが、このプロジェクトもアップします。（Vector の予定です）

2018/11/2 梅屋萬年堂 http:umeyamann.web.fc2.com　umeya.mann@gmail.com