```
//
//  EclipseState.swift
//  EclipseTimes
//
//  Created by Rob Hawley on 10/22/20.
//
// This struct will hold the state generated by a prediction.  That will avoid
 the
// display views having to dig into the EWPredict objects themselves

import UIKit
import SwiftUI
// When abs(v) less than this we are within 6 nm (11km) of the centerline
let gCloseEnough = 0.001

let gEclipseSummary = EclipseSummary()

let gTestEclipseSummary = EclipseSummary(test: true)

struct EventCirc{
    var eVis: EventVis = EventVis.disregard
    var utcDate = Date.distantPast // UTC Date of event
    var Z : Angle = -1.0
    var Zclock : Angle = -1.0 // Angle of moon entering {leaving} sun as seen
     by observer
                        // Meeus p 27
    var azi : Angle = -1.0 // azimuth to event
    var alt : Angle = -1.0 // alitude of event
}

class EclipseSummary: NSObject, ObservableObject{
    @Published var predictLocation =  ObsConditions(lat: 38.8976805, long:
     -77.0387185, height: 17.0) // default to White House

    @Published  var be : Bessel = EW2017_08_21  // elements for this eclipse

    @Published var magnitude: String = "-1.0" // magnitude of eclipse at this
     location
    @Published var duration: String =  "-1.0" // duration of eclipse in seconds

    @Published  var eclipseType = EventType.none0 // type of eclipse at this
     location

    @Published var v : Double = -1.0 // if positive observer is south of
     centerline


    @Published  var c1 = EventCirc()
    @Published var c2 = EventCirc()
    @Published  var mid = EventCirc()
    @Published  var c3 = EventCirc()
```

```
@Published  var c4 = EventCirc()

@Published  var sunrise = Date.distantFuture
@Published  var sunset = Date.distantFuture

private  let  whenFormatter = DateFormatter()

override init(){
    super.init()
}

/* create a test version of the eclipse summary*/
init (test: Bool){
    let  whenFormatter = DateFormatter()
    whenFormatter.timeZone = TimeZone(abbreviation: "UTC")
    whenFormatter.dateFormat = gFormatEventDate


    super.init()
    let thisTestLat = -31.5122
    let thislong = -68.5295
    let thisheight = 0.0  // real height 650.0

    predictLocation = ObsConditions(lat: thisTestLat, long: thislong,
     height: thisheight)
    be = EW2019_07_02

    magnitude = "1.001"
    duration = "234.4"
    eclipseType = EventType.total

    // v is initialized

    c1.eVis = EventVis.aboveHorizon
    c1.utcDate = whenFormatter.date(from: "07/02/19 19:25:59.5")!
    c1.alt = 22.3
    c1.azi = 318.3
    c1.Zclock = 25.0

    c2.eVis = EventVis.aboveHorizon
    c2.utcDate = whenFormatter.date(from: "07/02/19 20:40:15.1")!
    c2.alt = 10.5
    c2.azi = 305.4

    mid.eVis = EventVis.aboveHorizon
    mid.utcDate = whenFormatter.date(from: "07/02/19 20:40:36.0")!
    mid.alt = 10.4
    mid.azi = 305.4

    c3.eVis = EventVis.aboveHorizon
    c3.utcDate = whenFormatter.date(from: "07/02/19 20:40:56.8")!
    c3.alt = 10.4
    c3.azi = 305.3

    c4.eVis = EventVis.belowHorizon

}

// create an eclipse summary where only the be is set
func newBessel(whichEclipse: Bessel){
    let thisTestLat = 0.0
    let thislong = 0.0
    let thisheight = 0.0

    predictLocation = ObsConditions(lat: thisTestLat, long: thislong,
     height: thisheight)
    be = whichEclipse

    magnitude = "-1"
    duration = "-1"
    eclipseType = EventType.total

    // v is initialized

    c1.eVis = EventVis.disregard
    c1.utcDate = Date.distantFuture
    c1.alt = -1.0
    c1.azi = -1.0
    c1.Zclock = -1.0

    c2.eVis = EventVis.disregard
    c2.utcDate = Date.distantFuture
    c2.alt = -1.0
    c2.azi = -1.0

    mid.eVis = EventVis.disregard
    mid.utcDate = Date.distantFuture
    mid.alt = -1.0
    mid.azi = -1.0

    c3.eVis = EventVis.aboveHorizon
    c3.utcDate = Date.distantFuture
    c3.alt = -1.0
    c3.azi = -1.0

    c4.eVis = EventVis.disregard
}


func copyCirc(from: Circumstances,   to: inout EventCirc){
    to.eVis = from.eVis
    to.utcDate = from.utcDate
    to.alt = deg(from.alt)
    to.azi = deg(from.azi)
    to.Z = from.Z
    to.Zclock = from.Zclock


 }

// call after a prediction to populate this table
func populate(_ predict: EWPredict){
    be = predict.be
    predictLocation = predict.obsC

    sunrise = predict.sunriseDate
    sunset = predict.sunsetDate


    if predict.mid.eType != EventType.none0{
        magnitude =  String(format:"%1.3f", predict.mid.magnitude)
    } else {
        magnitude = NSLocalizedString("N/A", value:"N/A", comment: "Not
         applicable")
    }

    eclipseType = predict.mid.eType

    v = predict.mid.v // indicates wheither we are north or south

    // duration is more complicated
    if predict.c2.eVis != EventVis.aboveHorizon || predict.c3.eVis !=
     EventVis.aboveHorizon {
        duration =  NSLocalizedString("N/A", value:"N/A", comment: "what
         to show when nothing to show")
    } else {

        // OK both C2 and C3 have a valid utcDate

        let secs = predict.c3.utcDate.timeIntervalSince(predict.c2.utcDate)
        duration = String(format:"%1.0f",secs)
    }

    copyCirc(from: predict.c1, to: &c1)
    copyCirc(from: predict.c2, to: &c2)
    copyCirc(from: predict.mid, to: &mid)
    copyCirc(from: predict.c3, to: &c3)
    copyCirc(from: predict.c4, to: &c4)

    whenFormatter.timeZone = TimeZone(abbreviation: "UTC")
    let dateFormat = gFormatDate

    whenFormatter.dateFormat = dateFormat
```

```
}

 // return a string of this location relative to the centerline.
 // The normal case for tracks that run west to east is for this to return
 // north == < 0 , south == >0
 // for some eclipses (eg. polar eclipses like 2003 and 2008 ) a more
  accurate
 // description is west and east.
 // From there is gets complicated.

 // For the early part of the eclipse
 //     east == +
 //     west == -

 // for the later part of the eclipse
 //     east == -
 //     west == +

func northSouth()->Text{

    return Text(northSouthStr())

}
func northSouthStr() -> String{
    if eclipseType == EventType.none0 {
        return ""
    }

    // handle the simple case of close enough
    if abs(v) < gCloseEnough{
        return NSLocalizedString("near centerline", value:"Location is
         within 6 nm (11 km) of centerline", comment: "on center")
    }

    // now we have to consult the exceptions table to see if this is
    // one of the special cases
    let ewAnswer = ewSpecialCase()
    if ewAnswer != .notSpecial{
        return nsString(ewAnswer)
    }

    // we have to special case the eclipses of 2003, 2021, and 2033
    // In these cases the sun is shining over the south pole.
    // This causes v to behave differently than described in Meeus p 29
    if be.T0Date == EW2021_12_04.T0Date || be.T0Date ==
     EW2003_11_23.T0Date ||  be.T0Date == EW2033_03_30.T0Date {
        if (v < 0){
            return nsString(.southCenter)
        } else {
            return nsString(.northCenter)
        }

    }

    // Not a special case
    if (v > 0){
        return nsString(.southCenter)
    } else {
        return nsString(.northCenter)

    }
}

struct NSExceptionEntry{
    var date: Date // Date for which the exeption applies
    var long: Angle // Long rule applies  -west
    var ew: Bool // which direction is the rule applied true is apply west
    var resultForMinus: NSStringCase
    var resultForPlus: NSStringCase
}

private var nsExceptionTable: [NSExceptionEntry] =
            [
                NSExceptionEntry(date: NASA2008_08_01.T0Date,
                                 long: -64,
                                 ew: true,  // apply west
                                 resultForMinus: .westCenter,
                                 resultForPlus: .eastCenter
                ),
                NSExceptionEntry(date: NASA2008_08_01.T0Date,
                                 long: 45,
                                 ew: false,  // apply east
                                 resultForMinus: .eastCenter,
                                 resultForPlus: .westCenter
                ),
                NSExceptionEntry(date: EW2003_11_23.T0Date,
                                 long: 86,
                                 ew: false,  // apply east
                                 resultForMinus: .eastCenter,
                                 resultForPlus: .westCenter
                ),
                NSExceptionEntry(date: EW2033_03_30.T0Date,
                                 long: -127,
                                 ew: true,  // apply west  // 180 will
                                  stop comparison which is what we want
                                 resultForMinus: .westCenter,
                                 resultForPlus: .eastCenter
                )
            ]
```

```
// Handle the special case processing for northSouth() described above
// returns .notSpecial if default processing of v should happen
func ewSpecialCase() -> NSStringCase{

    for exception in nsExceptionTable{
        // examine an exception
        if be.T0Date != exception.date{
            continue // not interesting
        }

        // now we have 4 cases that can result.  For ease of understanding
         compute value for a switch
        var theCase = 0
        if exception.ew {
            theCase = 4
        } // else case of test east does not change thCase

        if predictLocation.longitudeAsDeg() < exception.long{
            theCase += 1 // we are west of the test point
        } // case of east does not change result

        switch theCase{

        // test west and iswest -> special case
        case 5,
        // test east and iseast -> special case
             0:

            if v > 0{
                return exception.resultForPlus
            } else {
                return exception.resultForMinus
            }
          default:
            continue
        }

    }

    return .notSpecial
}

func eventTime(row: Int ) -> ModifiedContent<Text,
 AccessibilityAttachmentModifier> {
    let  whenFormatter = DateFormatter()
    whenFormatter.timeZone = TimeZone(abbreviation: "UTC")
    whenFormatter.dateFormat = gFormatEventDate

    var retStr = ""
    var thisCirc: EventCirc
```

```
        switch row{
            case 1:
                thisCirc = c1
            case 2:
                thisCirc = c2
            case 3:
                thisCirc = mid
            case 4:
                thisCirc = c3
            case 5:
                thisCirc = c4
            default:
                thisCirc = c1
        }
        if thisCirc.eVis != EventVis.aboveHorizon{
            let retText =
             Text(notVis()).foregroundColor(Color.gray)
             .accessibility(identifier: "eventTime" + String(row))
            return retText
        } else {
            retStr = whenFormatter.string(from: thisCirc.utcDate)
            return Text(retStr).accessibility(identifier: "eventTime" +
             String(row))
        }

    }
    func eventAz(_ row: Int ) -> ModifiedContent<Text,
     AccessibilityAttachmentModifier>  {

        var retStr = ""
        var thisCirc: EventCirc
        switch row{
            case 1:
                thisCirc = c1
            case 2:
                thisCirc = c2
            case 3:
                thisCirc = mid
            case 4:
                thisCirc = c3
            case 5:
                thisCirc = c4
            default:
                thisCirc = c1
        }
        if thisCirc.eVis != EventVis.aboveHorizon{
            return Text("").accessibility(identifier: "eventAz" + String(row))
        } else {
            retStr = String(format:"%1.0f",thisCirc.azi)
            return Text(retStr).accessibility(identifier: "eventAz" +
             String(row))
        }

    }
    func eventAlt(_ row: Int ) -> ModifiedContent<Text,
     AccessibilityAttachmentModifier>  {

        var retStr = ""
        var thisCirc: EventCirc
        switch row{
            case 1:
                thisCirc = c1
            case 2:
                thisCirc = c2
            case 3:
                thisCirc = mid
            case 4:
                thisCirc = c3
            case 5:
                thisCirc = c4
            default:
                thisCirc = c1
        }
        if thisCirc.eVis != EventVis.aboveHorizon{
            return Text("").accessibility(identifier: "eventAlt" + String(row))
        } else {
            retStr = String(format:"%1.0f",thisCirc.alt)
            return Text(retStr).accessibility(identifier: "eventAlt" +
             String(row))
        }

    }
}



struct EclipseState_Previews: PreviewProvider {
    static var previews: some View {
        Text("Hello, World!")
    }
}
```