```
//
//  EWPredict.swift
//  Predict
//
//  Created by Rob Hawley on 10/3/20.
//  This file is Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)
//  Copyright © 2020 Rob Hawley. Some rights reserved.
//
// This is based on  Solar Eclipse Circumstances Calculator
//
// That code was being released under the terms of the GNU General Public
// License (http://www.gnu.org/copyleft/gpl.html) which IMHO is a compatible

// with the request that if
// you do improve on it or use it in your own site, please let me know at
// http://eclipsewise.com/main/contact.html
// Thank you.  - Fred Espenak
//
//http://www.eclipsewise.com/solar/index.html
//http://www.eclipsewise.com/solar/SEcirc/2001-2100/SE2017Aug21Tcirc
 .html#section3
//
/*
Solar Eclipse Circumstances Calculator
Version 1.0 by Bill Kramer and Fred Espenak - 2016 Apr 26.
(based on "Eclipse Calculator" by Deirdre O'Byrne and Stephen McCann - 2003)

Modified 2016 April for tabular display of local circumstances given
a list of cities. Cities are grouped by geographic regions.

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.
*/

// This alogorithm closes follows the procedure layed out in
// "Elements of Solar Eclipses" by Jean Meeus and implemented
// in a different object with at least one BIG exception.
// The Meeus algorithm as implemented directly from the book
// caused C2 and C3 to reverse under some circumstances.
// This contains a sign flop that fixes that.

// I have made some changes to the Javascript algorithm to prevent
// some computational errors and, at least in one case, where I disagree with
// a design decision
```

```
    // The Javascript subsitude sunmoon ratio for magnitude if the eclipse is
     total or annualar
    // I have retained the computation from Meeus p 27.  This means my
     magitude will differ
    // from the calculator.  It does agree with the implementations in the
     Google Maps pages

    // I protected the sunrise calculations from periods of no darkness

    // My calculation of the moon contact angle differs from the Javascript
     since my
    // calculation makes it more useful to observers.  The Javascript
     calculation is not exposed
    //

    // The prediction for north and south of the centerline discussed on Meeus
     p 29 is exposed.
    // I determined by experiment how large of value of v is represented 0.1°
     (6 nm)

import UIKit


// (0) Event type (C1=-2, C2=-1, Mid=0, C3=1, C4=2)
enum CircumType : Int{
    case C1 = -2
    case C2 = -1
    case Mid = 0
    case C3 = 1
    case C4 = 2
}

public enum EventType :Int{
    case none0 = 0
    case partial = 1
    case annular = 2
    case total = 3

}

enum EventVis : Int{
    case aboveHorizon = 0
    case belowHorizon = 1
    case sunrise
    case sunset
    case disregard

}

typealias Angle = Double
```

```
typealias RAngle = Double // Angle in radians

let noValue = -12345678.9012345


// I have organized the variables similiar to the Javascript to keep this
 closer to the reference code

class ObsConditions :NSObject{
  /*0*/  var latitude : RAngle
  /*1*/  var longitude : RAngle
  /*2*/  var altitude: Double /*meters*/
  /*3*/  /* The reference code makes a provision for time zones.  This code
   does everything in UTC
             since I cannot be guarenteed internet access to determine the
              Time Zone myself*/
    /*4*/  var ρSinϕP : Double /*rho sin O'*/
    /*5*/  var ρCosϕP : Double /*rho cos O'*/

    override init(){

        latitude = noValue
        longitude = noValue
        altitude = noValue
        ρSinϕP = noValue
        ρCosϕP = noValue

    }

    init( lat: Angle, long: Angle, height: Double /*meters*/){

        latitude = rad(lat)
        longitude = -rad(long) /* west is now +*/
        altitude = height
        let tmp = atan(0.99664719*tan(latitude))
        ρSinϕP = 0.99664719*sin(tmp) + height/6378140.0 * sin(latitude)

        ρCosϕP = cos(tmp) + height/6378140.0*cos(latitude)
    }

    func longitudeAsDeg() -> Angle {
        return -deg(longitude)
    }

    func latitudeAsDeg() -> Angle {
        return deg(latitude)
    }
    func stringValue() -> String{
       var ret = "-->Lat      " + String(format:"%1.6f", latitudeAsDeg()) + "\n"
       ret += "-->Long     " + String(format:"%1.6f", longitudeAsDeg()) + "\n"
       ret += "-->Height  " + String(Int(altitude)) + "\n"
```

```
        return ret

    }
}

struct Circumstances{
    /*0*/  var type : CircumType = CircumType.Mid
    /*1*/  var t : TDTrel  = noValue    /* Time relative to T0 in hours*/

    // -- time-only dependent circumstances (and their per-hour derivatives)
     follow --

    /* 2*/  var X : Double   = noValue    /* sum of X Bessel*/
    /* 3*/  var Y : Double   = noValue    /* sum of Y Bessel*/
    /* 4*/  var d : RAngle   = noValue    /* sum of d */
    /* 5*/  var sinD : Double = noValue
    /* 6*/  var cosD : Double = noValue
    /* 7*/  var M : RAngle   = noValue   /* aka μ */
    /* 8*/  var l1 : Double  = noValue
    /* 9*/  var l2 : Double  = noValue
    /*10*/  var dx : Double  = noValue   /* X' in Meeus*/
    /*11*/  var dy : Double  = noValue   /* Y' in Meeus*/
    /*12*/  var dd : RAngle  = noValue  // not in meeus
    /*13*/  var dmu : RAngle = noValue
    /*14*/  var dl1 : Double = noValue
    /*15*/  var dl2 : Double = noValue

    // -- time and location dependent circumstances follow --
    /*16*/  var h: RAngle     = noValue   //
    /*17*/  var sinh: Double = noValue
    /*18*/  var cosh: Double = noValue
    /*19*/  var xi : Double  = noValue   // ξ
    /*20*/  var eta : Double  = noValue  // η
    /*21*/  var zeta : Double = noValue  // ζ
    /*22*/  var dxi : Double  = noValue  // ξ'
    /*23*/  var deta : Double = noValue  // η'
    /*24*/  var u : Double    = noValue
    /*25*/  var v : Double    = noValue
    /*26*/  var a : Double    = noValue
    /*27*/  var b : Double    = noValue
    /*28*/  var l1p : Double  = noValue  // l1'
    /*29*/  var l2p : Double  = noValue  // l2'
    /*30*/  var nsqrd : Double = noValue // n^2

    // -- observational circumstances follow --

    /*31*/  var ρ : Double    = noValue
    /*32*/  var alt : RAngle  = noValue
    /*33*/  var q : RAngle     = noValue
    /*34*/  var vv : Double   = noValue
    /*35*/  var azi : RAngle   = noValue
```

```
    /*36*/  var correct :Double = noValue //m (mid eclipse only) or limb
     correction applied (where available!)
    /*37*/  var magnitude : Double  = noValue // mid eclipse
    /*38*/  var moonSun : Double = noValue
    /*39*/  var eType = EventType.none0
    /*40*/  var eVis: EventVis = EventVis.disregard

    var tanP : RAngle = -1.0
    var P : Angle = -1.0 // Position angle measured from North Point of the
     solar limb to the east
    var Z: Angle = -1.0  // Position angle of point of contact measured from
                         // zenith with east positive
                         // Meeus p 27
    var Zclock: Angle = -1.0  // angle relative to sun as viewed.  = Z in
     southern hem  360-Z in north

    var utcTime :UTCHours  = noValue // converted time of event (may be
     negative or > 24.0
    var utcDate  = Date()  // will be filled in with a Date Object
     corresponding to the UTC time of event

}


import Foundation

public class EWPredict: NSObject{
    let be : Bessel
    var c1 = Circumstances()
    var c2 = Circumstances()
    var mid = Circumstances()
    var c3 = Circumstances()
    var c4 = Circumstances()
    var obsC = ObsConditions()
    var sunriseDate = Date.distantFuture
    var sunsetDate = Date.distantFuture

    var dateT0 = Date()  // Mac Date object corresponding to T0 TDT
    var  whenFormatter = DateFormatter()


    override init (){

        be = NASA1994_05_10 // placeholder
        super.init()
     }

    init(whichEclipse: Bessel,obsC location: ObsConditions){
        be = whichEclipse
        obsC = location
        c1.type = CircumType.C1
```

```
        c2.type = CircumType.C2
        mid.type = CircumType.Mid
        c3.type = CircumType.C3
        c4.type = CircumType.C4

        dateT0 = be.T0Date + be.T0*3600.00

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

     func predict(  ) /*->EventRecord*/{

        // get the mid point for the eclipse
        getMid()
        // calculated some useful stuff like magnitude
        midObservational()

        // depending on the type of eclipse calc other parameters

        // If we are really eclipsed
        if mid.magnitude > 0 {
            getc1c4()

            if mid.correct < mid.l2p || mid.correct < -mid.l2p{
                // either annular or total
                getc2c3()

                if mid.l2p < 0.0{
                    mid.eType = EventType.total
                } else {
                    mid.eType = EventType.annular
                }
                // Fill in the observation circumstances
                observational(vars: &c1, isC2C3: false)
                observational(vars: &c2, isC2C3: true)
                observational(vars: &c3, isC2C3: true)
                observational(vars: &c4, isC2C3: false)

                c2.correct = 999.9
                c3.correct = 999.9

                // The javascript does some calculations at this point
                // with the sunset/rise calculations; however, I have
                // found these completely unreliable.  So, instead, we
                // will just check to make sure that C2 and C3 are
                // both visible.  Otherwise we will declare the eclipse type
                 as none
                if c2.eVis == .belowHorizon || c3.eVis == .belowHorizon{
                    mid.eType = .none0
```

```
            // Ah but if I can see either C1 or C4 convert it to a
             partial
            if c1.eVis == .aboveHorizon || c4.eVis == .aboveHorizon{
                mid.eType = .partial
            }
        }



        // Earlier C2 and C3 were corrected to the opposite side of
        // the moon assuming a total eclipse.  If this is an annular
        // then it is more interesting to display the side where the
         moon
        // last touched the sun (i.e. 180 degrees)
        if mid.eType == EventType.annular{
            if c2.eVis == EventVis.aboveHorizon{
                if c2.Zclock > 180.0 {
                    c2.Zclock -= 180.0
                } else {
                    c2.Zclock += 180.0

                }
            }
            if c3.eVis == EventVis.aboveHorizon{
                if c3.Zclock > 180.0 {
                    c3.Zclock -= 180.0
                } else {
                    c3.Zclock += 180.0

                }
            }

        }

    } else {
        mid.eType = EventType.partial
        observational(vars: &c1, isC2C3: false)
        observational(vars: &c4, isC2C3: false)

        // If the beginning. middle, and end of the eclipse are all
         saying below horizon then
        // the partial declaration is wrong
        if c1.eVis == .belowHorizon && mid.eVis == .belowHorizon &&
         c4.eVis == .belowHorizon{
            mid.eType = .none0
        }

        c2.eVis = EventVis.disregard
        c3.eVis = EventVis.disregard
```

```
        }
    } else {
        mid.eType = EventType.none0
    }

    // The EclipseWise code at this point promotes moon sun to magnitude
     in the case of
    // total and annular eclipses

    // I decided to retain the definition of Meeus from page 25.  This is
     the result
    // displayed on the current Google Map pages.

   // if mid.eType == EventType.annular || mid.eType == EventType.total{
   //     mid.magnitude = mid.moonSun
  //  }

//     NSLog ("C1=" + fmtHours(c1.t + be.T0 - be.ΔT/3600) + " alt=" +
 prt5(c1.alt) + " az=" + prt5(c1.azi))

    // glue to hide the call to the NOAA sunset/rise calcs
    noaaglue()

}
func noaaglue(){

    // calculate approximate noon time
    let localNoon = 12.0 + obsC.longitudeAsDeg()/15.0

    #if DEBUG
    //NSLog("local noon = " + fmtHours(localNoon) )
    #endif

    //Now make the NOAA prediction
    let noaaCalcs = SolarCalcs(loc: obsC, atDay: be.T0Date, atTime:
     localNoon)
    noaaCalcs.sunsetRise() // calc the hour times
    sunriseDate = noaaCalcs.sunriseDate
    sunsetDate = noaaCalcs.sunsetDate
}

// Populate the circumstances array with the time-only dependent
 circumstances (x, y, d, m, ...)

func timeDependent( vars: inout Circumstances){

    let t : TDTrel = vars.t

    vars.X = be.x[0] + t*(be.x[1] + t*(be.x[2] + t*be.x[3]))
```

```
vars.Y =  be.y[0] + t*(be.y[1] + t*(be.y[2] + t*be.y[3]))

// These are X' and Y' in Meeus
vars.dx = be.x[1] + 2.0*t*be.x[2] + 3.0*t*t*(be.x[3])

// ans = 3.0 * elements[13+index] * t + 2.0 * elements[12+index]
// dy = ans * t + elements[11+index]

vars.dy = be.y[1] + t*(3.0*be.y[3]*t + 2.0*be.y[2])

vars.d = rad(be.d[0] + t*(be.d[1] + t*(be.d[2])))

vars.sinD = sin(vars.d)
vars.cosD = cos(vars.d)

// ans = 2.0 * elements[16+index] * t + elements[15+index]
// dd = ans * Math.PI / 180.0

vars.dd = rad(be.d[1] + 2*be.d[2]*t)

// ans = elements[19+index] * t + elements[18+index]
// M = ans * t + elements[17+index]

var workM = be.M[0] + t*(be.M[1] + t*be.M[2])

if workM >= 360 {
    workM = workM - 360
}

vars.M = rad(workM)

vars.dmu = rad(be.M[1] + 2 * t * be.M[2])

// calculate L1 and dl1
let type = vars.type

if type == CircumType.C1 || type == CircumType.Mid || type ==
 CircumType.C4{

    vars.l1 = be.l1[0] + t*(be.l1[1] + t*(be.l1[2]))

    //  dl1 = 2.0 * elements[22+index] * t + elements[21+index]

    vars.dl1 = be.l1[1] + 2.0*be.l1[2]*t
}
if type ==  CircumType.C2 || type == CircumType.Mid || type ==
 CircumType.C3 {

    vars.l2 = be.l2[0] + t*(be.l2[1] + t*(be.l2[2]))

    //   dl2 = 2.0 * elements[25+index] * t + elements[24+index]
```

```
        vars.dl2 = be.l2[1] + 2.0*be.l2[2]*t
    }

}

func timelocDependent(vars: inout Circumstances){

    timeDependent(vars: &vars)

    // calculate h
    //   h = circumstances[7] - obsvconst[1] - (elements[index+5] /
     13713.44)

    vars.h = vars.M - obsC.longitude - (be.ΔT/13713.44)
    vars.sinh = sin(vars.h)
    vars.cosh = cos(vars.h)

    // calculate xi
    vars.xi = obsC.ρCosϕP * vars.sinh

    // calculate eta
    // eta = obsvconst[4] * circumstances[6] - obsvconst[5] *
     circumstances[18] * circumstances[5]

    vars.eta = obsC.ρSinϕP * vars.cosD -  obsC.ρCosϕP * vars.cosh *
     vars.sinD

    // Calculate zeta
    // zeta = obsvconst[4] * circumstances[5] + obsvconst[5] *
     circumstances[18] * circumstances[6]

    vars.zeta = obsC.ρSinϕP * vars.sinD + obsC.ρCosϕP * vars.cosh *
     vars.cosD

    // Calculate dxi
    // dxi = circumstances[13] * obsvconst[5] * circumstances[18]
    vars.dxi = vars.dmu * obsC.ρCosϕP * vars.cosh

    // Calculate deta
    // deta = circumstances[13] * circumstances[19] * circumstances[5] -
     circumstances[21] * circumstances[12]

    vars.deta = vars.dmu*vars.xi*vars.sinD - vars.zeta * vars.dd

    // Calculate u
    // u = circumstances[2] - circumstances[19]
    vars.u = vars.X - vars.xi

    // Calculate v
    // v = circumstances[3] - circumstances[20]
```

```
    vars.v = vars.Y - vars.eta

    // Calculate a
    // a = circumstances[10] - circumstances[22]
    vars.a = vars.dx - vars.dxi

    // Calculate b
    // b = circumstances[11] - circumstances[23]
    vars.b = vars.dy - vars.deta

    let type = vars.type

    // Calculate l1'
    if type == CircumType.C1 || type == CircumType.Mid || type ==
     CircumType.C4{

        //     l1p = circumstances[8] - circumstances[21] *
         elements[26+index]

        vars.l1p = vars.l1 - vars.zeta * be.tanf1

    }

    // Calculate l2'
    if type ==  CircumType.C2 || type == CircumType.Mid || type ==
     CircumType.C3 {

        //   l2p = circumstances[9] - circumstances[21] *
         elements[27+index]

        vars.l2p = vars.l2 - vars.zeta * be.tanf2
    }
    // Calculate n^2
    //circumstances[30] = circumstances[26] * circumstances[26] +
     circumstances[27] * circumstances[27]

    vars.nsqrd = vars.a*vars.a + vars.b*vars.b


}

func c1c4iterate(vars: inout Circumstances){

    timelocDependent(vars: &vars)

    let type = vars.type
    var sign = -1.0

    if (type == CircumType.C1) {
        sign = -1.0
    } else {
```

```
        sign = 1.0
    }

    var tmp = 1.0
    var iter = 0

    while abs(tmp) > 0.000001 && iter < 50 {
        let n = sqrt(vars.nsqrd)

        //tmp = circumstances[26] * circumstances[25] - circumstances[24]
         * circumstances[27]
       // tmp = tmp / n / circumstances[28]
       // tmp = sign * Math.sqrt(1.0 - tmp * tmp) * circumstances[28] / n
       // tmp = (circumstances[24] * circumstances[26] + circumstances[25]
        * circumstances[27]) / circumstances[30] - tmp

        tmp = vars.a*vars.v - vars.u*vars.b
        let S = tmp / n / vars.l1p // from Meeus p 26
        tmp = sign * sqrt(1.0 - S * S ) * vars.l1p / n
        tmp = (vars.u*vars.a + vars.v*vars.b)/vars.nsqrd - tmp

        vars.t = vars.t - tmp

        timelocDependent(vars: &vars)

        iter += 1


    }
}

//
// Get C1 and C4 data
//   Entry conditions -
//   1. The mid array must be populated
//   2. The magnitude at mid eclipse must be > 0.0

func getc1c4(){
    let n = sqrt(mid.nsqrd)

    // tmp = mid[26] * mid[25] - mid[24] * mid[27]

    var tmp = mid.a*mid.v - mid.u*mid.b
    tmp = tmp / n / mid.l1p
    tmp = sqrt(1.0 - tmp*tmp) * mid.l1p / n

    c1.t = mid.t - tmp
    c4.t = mid.t + tmp
    c1c4iterate(vars: &c1)
    c1c4iterate(vars: &c4)
}
```

```
func c2c3iterate(vars: inout Circumstances){
    timelocDependent(vars: &vars)

    var sign : Double
    let type = vars.type

    if (type == CircumType.C2) {
        sign = -1.0
    } else {
        sign = 1.0
    }
    //if (mid[29] < 0.0) {
    if (mid.l2p < 0.0) {
        sign = -sign
      }
    var tmp = 1.0
    var iter = 0

    while abs(tmp) > 0.000001 && iter < 50 {
        let n = sqrt(vars.nsqrd)

        //tmp = circumstances[26] * circumstances[25] - circumstances[24]
         * circumstances[27]
        //tmp = tmp / n / circumstances[29]
        //tmp = sign * Math.sqrt(1.0 - tmp * tmp) * circumstances[29] / n
       // tmp = (circumstances[24] * circumstances[26] + circumstances[25]
        * circumstances[27]) / circumstances[30] - tmp

        tmp = vars.a*vars.v - vars.u*vars.b
        let S = tmp / n / vars.l2p     // from Meeus p 26
        tmp = sign * sqrt(1.0 - S * S ) * vars.l2p / n
        tmp = (vars.u*vars.a + vars.v*vars.b)/vars.nsqrd - tmp

        vars.t = vars.t - tmp

        timelocDependent(vars: &vars)

        iter += 1
    }
}

//
// Get C2 and C3 data
//   Entry conditions -
//   1. The mid array must be populated
//   2. There must be either a total or annular eclipse at the location!
func getc2c3(){

    let n = sqrt(mid.nsqrd)
```

```
    // tmp = mid[26] * mid[25] - mid[24] * mid[27]
    var tmp = mid.a*mid.v - mid.u*mid.b

    tmp = tmp / n / mid.l2p
    tmp = sqrt(1.0 - tmp*tmp) * mid.l2p / n

    ////////////////////////////////////////////////////////////////////
    // Ah Hah !!!
    //
    // The secret not included in the Meeus book that
    // causes C2 and C3 to be reversed

    if mid.l2p < 0.0 {
        c2.t = mid.t + tmp
        c3.t = mid.t - tmp
    } else {
        c2.t = mid.t - tmp
        c3.t = mid.t + tmp
    }
    c2c3iterate(vars: &c2)
    c2c3iterate(vars: &c3)

}
// Get the observational circumstances

func observational(vars: inout Circumstances, isC2C3: Bool){
    var contactType :Double

    // We are looking at an "external" contact UNLESS this is a total
     eclipse AND we are looking at
    // c2 or c3, in which case it is an INTERNAL contact! Note that if we
     are looking at mid eclipse,
    // then we may not have determined the type of eclipse (mid[39]) just
     yet!

    if vars.type == CircumType.Mid{
        contactType = 1.0
    } else {
        if mid.eType == EventType.total && (vars.type == CircumType.C2 ||
         vars.type == CircumType.C3){

            contactType = -1.0
        } else {
            contactType = 1.0
        }
    }

    // Calculate p
    //  ρ = Math.atan2(contacttype*circumstances[24],
     contacttype*circumstances[25])
    vars.ρ = atan2(contactType * vars.u, contactType*vars.v)
```

```
// Calculate alt
let sinlat = sin(obsC.latitude)
let coslat = cos(obsC.latitude)
// alt = Math.asin(circumstances[5] * sinlat + circumstances[6] *
 coslat * circumstances[18])

vars.alt = asin(vars.sinD*sinlat + vars.cosD*coslat*vars.cosh)
//  circumstances[33] = Math.asin(coslat * circumstances[17] /
 Math.cos(circumstances[32]))
//if (circumstances[20] < 0.0) {
//  circumstances[33] = Math.PI - circumstances[33]
//}

let sinq = sin(coslat * vars.sinh) / cos(vars.alt)
vars.q = asin(sinq)
if vars.eta < 0.0 {
    vars.q = Double.pi - vars.q
}

// Calculate azi
// azi = Math.atan2(-1.0*circumstances[17]*circumstances[6],
 circumstances[5]*coslat - circumstances[18]*sinlat*circumstances[6])

vars.azi = atan2(-1.0*vars.sinh*vars.cosD, vars.sinD*coslat -
 vars.cosh*sinlat*vars.cosD)

if vars.azi < 0 {
    vars.azi = vars.azi + 2 * 3.141592654
}

// calculate angle of contact
vars.tanP =  vars.u/vars.v
vars.P = deg(atan2 (vars.u, vars.v))
// vars.P = deg(atan(vars.tanP))

//NSLog(prt5(deg(test)))



if vars.P < 0 {
    vars.P += 360.0
}

if vars.P > 360.0 {
    vars.P -= 360.0
}


let tempq = deg(vars.q)
vars.Z = vars.P - tempq
```

```
    if vars.Z < 0 {
        vars.Z += 360.0
    }

    if vars.Z > 360.0 {
        vars.Z -= 360.0
    }

    // The calculation that Meeus does is accurate, but not completely
     useful.  His angle is east from Zenith
    // an easier reference is observer position.
    vars.Zclock = 360.0 - vars.Z

    // for total eclipse you want the opposite side for C2 C3 - however
    // at this point we do not know the type of eclipse
    if isC2C3 {
        vars.Zclock += 180.0

        if vars.Zclock > 360.0 {
            vars.Zclock -= 360.0
        }
    }



    // Calculate visibility
    if vars.alt > -0.00524 {
        vars.eVis = EventVis.aboveHorizon
    } else {
        vars.eVis = EventVis.belowHorizon

    }

    // calculate the UTC time
    vars.utcTime = vars.t + be.T0 - be.ΔT/3600
    vars.utcDate = be.T0Date + vars.utcTime*3600

    //var eventName:String

    //switch vars.type{
   // case .C1:
    //    eventName = "C1"
   // case .C2:
   //     eventName = "C2"
    //case .Mid:
   //     eventName = "mid"
  //  case .C3:
  //      eventName = "C3"
 //   case .C4:
  //      eventName = "C4"
```

```
      //}
      
        //NSLog(eventName + " time=" + whenFormatter.string(from:
         vars.utcDate) + " alt=" + String(format:"%1.1f", deg(vars.alt)) + "
         az=" + String(format:"%1.1f", deg(vars.azi)))

    }


    // Get the observational circumstances for mid eclipse
    func midObservational(){

        // calculate the remainding variables
        observational(vars: &mid, isC2C3: false)

        mid.correct = sqrt(mid.u*mid.u + mid.v*mid.v)
        mid.magnitude = (mid.l1p - mid.correct)/(mid.l1p + mid.l2p)
        mid.moonSun = (mid.l1p - mid.l2p)/(mid.l1p + mid.l2p)


   //     getsunrise(vars: mid)
  //      getsunset(vars: mid)
     }



    //
    // Calculate mid eclipse
    func getMid() {
        mid.t = 0.0
        var iter = 0
        var tmp = 1.0

        timelocDependent(vars: &mid)

        while abs(tmp) > 0.000001 && iter < 50 {
            //   tmp = (mid[24] * mid[26] + mid[25] * mid[27]) / mid[30]

            tmp = (mid.u*mid.a + mid.v*mid.b)/mid.nsqrd
            mid.t = mid.t - tmp
            iter += 1
            timelocDependent(vars: &mid)

        }

    }



}
```