```
//
//  ExecutionTests.swift
//  EclipseTimesTests
//
//  Created by Rob Hawley on 10/28/20.
//

import XCTest
import EclipseTimes

class UI2010_2021: XCTestCase {

#if dontcompile
///////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////   prototype   ////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////

func prototypeUI() throws {
    let decimalSec = DateFormatter() // used to convert numbers displayed
     decimalSec.timeZone = TimeZone(abbreviation: "UTC")
    decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    let noaaFmt = DateFormatter() // used to convert answer numbers from maps
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"


    var diff : Double

    ////////////////  Success criteria ////////////////////////

        //            Summary View

        let correctNSText = "Location is within 6 nm (11 km) of centerline"
        let correctDescription = "Prediction for Total Eclipse of Jul 02, 2019"
        let correctMap = "???"
        let correctLink = "???"
        let credit = app.staticTexts["EW source"]


    // By default we are testing EW with lat N  long W at height 0.  Code
     below will
    // have to be changed if this is different

    let latString = "xxxx"
    let longString = "yyyy"
    let wheelString = "zzzz"


    let correctMag : Double = -1.0
    let correctC1angle: Double  =  -1.0
```

```
    let semC2angle: Double =  -1.0
    let semC3angle: Double  =  -1.0
    let correctC4angle: Double  =  -1.0

    let noaaSunriseDate = noaaFmt.date(from: "sunrise")!
    let noaaSunsetDate = noaaFmt.date(from: "sunset")!

    //          circs

    let correctC1Date = decimalSec.date(from: "C1 time")!
      let correctC1Alt: Double  = -1.0
      let correctC1Az: Double  = -1.0

    let correctC2Date = decimalSec.date(from: "C2 time")!
      let correctC2Alt: Double  = -1.0
      let correctC2Az: Double  = -1.0

    let correctMidDate = decimalSec.date(from: "mid time")!
      let correctMidAlt: Double  = -1.0
      let correctMidAz: Double  = -1.0

    let correctC3Date = decimalSec.date(from: "C3 time")!
      let correctC3Alt: Double  = -1.0
      let correctC3Az: Double  = -1.0

    let correctC4Date = decimalSec.date(from: "C4 time")!
      let correctC4Alt: Double  = -1.0
      let correctC4Az: Double  = -1.0


     // Use XCTAssert and related functions to verify your tests produce the
      correct results.
    let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

    getState()

//    XCTAssert(!nasaMode)
    picker!.adjust(toPickerWheelValue: wheelString)
    getState()
    XCTAssert(pickerShown == wheelString)

    XCTAssert(degShowing)

     XCTAssert(nSwitchValue == true)
     XCTAssert(sSwitchValue == false)

    XCTAssert(eSwitchValue == false)
    XCTAssert(wSwitchValue == true)

    latTextField!.tap()
    latTextField!.clearAndEnterText(text: latString)
```

```
longTextField!.tap()
longTextField!.clearAndEnterText(text: longString)
heightTextField!.tap()
heightTextField!.clearAndEnterText(text: "0")

app.buttons["KeyDone"].tap()
getState()
XCTAssert(stripZero(latValue) ==  latString)
XCTAssert(stripZero(longValue) == longString)


app.buttons["PredictBut"].tap()

// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
 XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)




// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gNumbers)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gNumbers)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
```

```
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()
```

```
//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <=  gEWMapNum)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <= gEWMapNum)
```

```
///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <= gEWMapNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff <= gEWNum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff <= gEWMapNum)

///////////////// c4 ////////////////////////////////////
```

```
    let c4timeQ = app.staticTexts["eventTime5"]
    XCTAssert(c4timeQ.exists)
    let c4timeStr = c4timeQ.label
    let c4time = decimalSec.date(from: c4timeStr)
    XCTAssert(c4time != nil)
    diff = abs(c4time!.timeIntervalSince(correctC4Date))
    XCTAssert( diff <= gEWCalc)

    let c4alt = app.staticTexts["eventAlt5"]
    XCTAssert(c4alt.exists)
    let c4altStr = c4alt.label
    let c4AltAsNumber = Double(c4altStr)
    XCTAssert(c4AltAsNumber != nil)
    diff = abs( c4AltAsNumber! - correctC4Alt)
    XCTAssert( diff <= gEWNum)

    let c4az = app.staticTexts["eventAz5"]
    XCTAssert(c4az.exists)
    let c4azStr = c4az.label
    let c4AzAsNumber = Double(c4azStr)
    XCTAssert(c4AzAsNumber != nil)
    diff = abs( c4AzAsNumber! - correctC4Az)
    XCTAssert( diff <= gEWMapNum)


    XCTAssert(credit.exists)


    app.buttons["Map"].tap()
    //NSLog(app.debugDescription)
    XCTAssert(app.images[correctMap].exists)
    XCTAssert(app.buttons[correctLink].exists)
}

#endif


    var app  = XCUIApplication()


    //newer tests

    let  gEWCalc:Double = 1.0 // EW Bessels compared to the EW calculator must
     exactly agree allow numbers to be +- 1 to compensate for differences in
     rounding
    let  gEWNum: Double = 1 // allow numbers to be +- 1 to compensate for
     differences in rounding
    let gEWMapNum:Double = 1 // Since this number is not published in the
     calculation get it from a map. Allow numbers to be +- 1 to compensate for
     differences in rounding
```

```
    let gNASAMapTimes : Double = 1 // agreement required in the times of tests
     where NASA Bessels compare with the NASA map
    let gNASANum: Double = 1 // allow numbers to be +- 1 to compensate for
     differences in rounding

    // common to old and new
    let gPassSunsetRise : Double = 3.0 * 60.0  // compare to NOAA algorithm
   let gVisualMeasurement: Double = 12.5 // when I am visually measuring an
    angle


    // older tests
    var  gpassMap:Double  = 0.75  // .75 seconds  for comparision with google
     map  comparison is to seconds
    var  gpassNASA:Double  = 1.5 // for comparison against the map predictions
     using NASA elements
    let gNumbers:Double = 1 // allow for ± 1 for rounding

    let  gPassJavascript : Double = 0.1 / 3600.0  // pass criteria when
     comparing against javascript  comparison is to TDTHours

    // state variables on main screen
    var picker: XCUIElement?
    var pickerShown = "NotValid"

    // some of the fields below may not be value
    var latTextField: XCUIElement?
    var latValue  = "NotValid"
    var longTextField: XCUIElement?
    var longValue = "NotValid"

    var latDegTextField: XCUIElement?
    var latDegValue  = "NotValid"
    var latMinTextField: XCUIElement?
    var latMinValue  = "NotValid"
    var latSecTextField: XCUIElement?
    var latSecValue  = "NotValid"

    var longDegTextField: XCUIElement?
    var longDegValue  = "NotValid"
    var longMinTextField: XCUIElement?
    var longMinValue  = "NotValid"
    var longSecTextField: XCUIElement?
    var longSecValue  = "NotValid"

    var heightTextField: XCUIElement?
    var heightValue  = "NotValid"

//    var nasaSwitch:  XCUIElement?
 //   var ewSwitch:  XCUIElement?
//    var nasaMode = true
```

```
var degSwitch : XCUIElement?
var degShowing = true
var dmsSwitch : XCUIElement?


var nSwitch : XCUIElement?
var nSwitchValue = false
var sSwitch : XCUIElement?
var sSwitchValue = false
var eSwitch : XCUIElement?
var eSwitchValue = false
var wSwitch : XCUIElement?
var wSwitchValue = false

var predictButton : XCUIElement?
var predictEnabled = false

var hereButton: XCUIElement?



// convert DMS into degress
func dms (_ deg: Double,_ min: Double, _ sec: Double) -> Double{
    var sign = 1.0
    var workdeg = deg

    if deg < 0{
        sign = -1.0
        workdeg = abs(workdeg)
    }

    return sign * (workdeg + min/60 + sec/3600)
}

func getState(){
    picker = app.pickerWheels.firstMatch
    pickerShown = picker?.value as! String

    degSwitch = app.segmentedControls.buttons["Deg"]
    degShowing = degSwitch!.isSelected
    dmsSwitch = app.segmentedControls.buttons["DMS"]

    XCTAssert(degSwitch!.isSelected != dmsSwitch!.isSelected )

    if degShowing {
        latTextField = app.textFields["latasDeg"]
        latValue = latTextField!.value as! String
        longTextField = app.textFields["longasDeg"]
        longValue = longTextField!.value as! String
```

```
        } else {

            latDegTextField = app.textFields["latDMSDeg"]
            latDegValue = latDegTextField!.value as! String
            latMinTextField = app.textFields["latDMSMin"]
            latMinValue = latMinTextField!.value as! String
            latSecTextField = app.textFields["latDMSSec"]
            latSecValue = latSecTextField!.value as! String

            longDegTextField = app.textFields["longDMSDeg"]
            longDegValue = longDegTextField!.value as! String
            longMinTextField = app.textFields["longDMSMin"]
            longMinValue = longMinTextField!.value as! String
            longSecTextField = app.textFields["longDMSSec"]
            longSecValue = longSecTextField!.value as! String

        }

        heightTextField = app.textFields["Height"]
        heightValue  = heightTextField!.value as! String


        nSwitch = app.segmentedControls.buttons["N"]
        nSwitchValue = nSwitch!.isSelected

        sSwitch = app.segmentedControls.buttons["S"]
        sSwitchValue = sSwitch!.isSelected

        eSwitch = app.segmentedControls.buttons["E"]
        eSwitchValue = eSwitch!.isSelected
        wSwitch = app.segmentedControls.buttons["W"]
        wSwitchValue = wSwitch!.isSelected

        predictButton = app.buttons["PredictBut"]
        predictEnabled = predictButton!.isEnabled

//        nasaSwitch = app.segmentedControls.buttons["NASA"]
//        ewSwitch = app.segmentedControls.buttons["EclipseWise"]
//        nasaMode = nasaSwitch!.isSelected
//        XCTAssert(nasaSwitch!.isSelected != ewSwitch!.isSelected )

        hereButton = app.buttons["Here"]
    }
    override func setUpWithError() throws {

         // Put setup code here. This method is called before the invocation
          of each test method in the class.

        // In UI tests it is usually best to stop immediately when a failure
         occurs.
```

```
    continueAfterFailure = false

    // UI tests must launch the application that they test. Doing this in
     setup will make sure it happens for each test method.
    XCUIApplication().launch()

    // In UI tests it's important to set the initial state - such as
     interface orientation - required for your tests before they run. The
     setUp method is a good place to do this.
}

override func tearDownWithError() throws {
    // Put teardown code here. This method is called after the invocation
     of each test method in the class.
}



///////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////  1994  ////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////

func Not_used_test_1994_GE() throws {
    var diff: Double
    let decimalSec = DateFormatter() // used to convert numbers displayed
     let integerSec = DateFormatter() // used to convert answer numbers
      from maps
     integerSec.timeZone = TimeZone(abbreviation: "UTC")
     integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
     decimalSec.timeZone = TimeZone(abbreviation: "UTC")
     decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"



    ///////////////// Success criteria ////////////////////////

        //            Summary View

        let correctNSText = "Location is within 6 nm (11 km) of centerline"
        let correctDescription = "Prediction for Annular Eclipse of May
         10, 1994"

    // By default we are testing EW with lat N  long W at height 0.  Code
     below will
    // have to be changed if this is different

    let latString = "41.5357"
    let longString = "84.1207"
    let wheelString = "1994 May 10 A"
```

```
        let correctC1angle = 70.0
        let semC2angle = 270.0 - 180.0
        let semC3angle = 100.0 + 180.0
        let correctC4angle = 320.0



        //NSLog(whenFormatter.string(from:engine.sunriseDate))
        //NSLog(whenFormatter.string(from:engine.sunsetDate))
        let correctsunriseDate = decimalSec.date(from: "05/10/94 10:26:50.0")!

         let correctsunsetDate = decimalSec.date(from: "05/11/94 00:39:26.8")!


        // test magnitude
        let correctMag = 0.972


        // Test C1
        let correctC1Date = decimalSec.date(from: "05/10/94 15:29:26.1")!
        let correctC1Alt = 54.4
        let correctC1Az = 122.8

        let correctC2Date = decimalSec.date(from: "05/10/94 17:08:19.5")!
        let correctC2Alt = 65.6
        let correctC2Az = 165.8

        let correctMidDate = decimalSec.date(from: "05/10/94 17:11:26.2")!
        let correctMidAlt = 65.7
        let correctMidAz = 167.5

        let correctC3Date = decimalSec.date(from: "05/10/94 17:14:32.9")!
        let correctC3Alt = 65.8
        let correctC3Az = 169.3

        let correctC4Date = decimalSec.date(from: "05/10/94 18:58:09.7")!
        let correctC4Alt = 60.0
        let correctC4Az = 223.8


         // Use XCTAssert and related functions to verify your tests produce
          the correct results.
        let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

        getState()

//        XCTAssert(!nasaMode)
        picker!.adjust(toPickerWheelValue: wheelString)
        getState()
        XCTAssert(pickerShown == wheelString)
```

```
XCTAssert(degShowing)

 XCTAssert(nSwitchValue == true)
 XCTAssert(sSwitchValue == false)

XCTAssert(eSwitchValue == false)
XCTAssert(wSwitchValue == true)

// This is a GE test
//latTextField!.tap()
//latTextField!.clearAndEnterText(text: latString)
//longTextField!.tap()
//longTextField!.clearAndEnterText(text: longString)
heightTextField!.tap()
heightTextField!.clearAndEnterText(text: "0")

app.buttons["KeyDone"].tap()
getState()
XCTAssert(stripZero(latValue) ==  latString)
XCTAssert(stripZero(longValue) == longString)


app.buttons["PredictBut"].tap()

// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
 XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)




// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff < gNumbers)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
```

```
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff < gNumbers)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)


// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = integerSec.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(correctsunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)
```

```
let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = integerSec.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(correctsunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff < (gpassNASA ))

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff < gVisualMeasurement)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff < gVisualMeasurement)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff < (gpassNASA ))

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
```

```
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff < gVisualMeasurement)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff < gVisualMeasurement)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff < (gpassNASA ))

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff < gVisualMeasurement)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff < gVisualMeasurement)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff < (gpassNASA ))

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
```

```
    let c3AltAsNumber = Double(c3altStr)
    XCTAssert(c3AltAsNumber != nil)
    diff = abs( c3AltAsNumber! - correctC3Alt)
    XCTAssert( diff < gVisualMeasurement)

    let c3az = app.staticTexts["eventAz4"]
    XCTAssert(c3az.exists)
    let c3azStr = c3az.label
    let c3AzAsNumber = Double(c3azStr)
    XCTAssert(c3AzAsNumber != nil)
    diff = abs( c3AzAsNumber! - correctC3Az)
    XCTAssert( diff < gVisualMeasurement)

    ///////////////// c4 ///////////////////////////////////
    let c4timeQ = app.staticTexts["eventTime5"]
    XCTAssert(c4timeQ.exists)
    let c4timeStr = c4timeQ.label
    let c4time = decimalSec.date(from: c4timeStr)
    XCTAssert(c4time != nil)
    diff = abs(c4time!.timeIntervalSince(correctC4Date))
    XCTAssert( diff < (gpassNASA ))

    let c4alt = app.staticTexts["eventAlt5"]
    XCTAssert(c4alt.exists)
    let c4altStr = c4alt.label
    let c4AltAsNumber = Double(c4altStr)
    XCTAssert(c4AltAsNumber != nil)
    diff = abs( c4AltAsNumber! - correctC4Alt)
    XCTAssert( diff < gVisualMeasurement)

    let c4az = app.staticTexts["eventAz5"]
    XCTAssert(c4az.exists)
    let c4azStr = c4az.label
    let c4AzAsNumber = Double(c4azStr)
    XCTAssert(c4AzAsNumber != nil)
    diff = abs( c4AzAsNumber! - correctC4Az)
    XCTAssert( diff < gVisualMeasurement)

    let credit = app.staticTexts["EW source"]
    XCTAssert(credit.exists)


    app.buttons["Map"].tap()
    //NSLog(app.debugDescription)
    XCTAssert(app.images["1994m05d10"].exists)
    XCTAssert(app.buttons["https://www.willbell.com/math/mc6.htm"].exists)
}

///////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////  2010  ////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////
```

```
func testUINASA2010_Chile() throws {
    let decimalSec = DateFormatter() // used to convert numbers displayed
    let integerSec = DateFormatter() // used to convert answer numbers
     from maps
    integerSec.timeZone = TimeZone(abbreviation: "UTC")
    integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
    decimalSec.timeZone = TimeZone(abbreviation: "UTC")
    decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    let noaaFmt = DateFormatter() // used to convert answer numbers from
     maps
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"

    var diff : Double

    ///////////////// Success criteria //////////////////////////

        //            Summary View

        let correctNSText = "Location is south of the centerline"
        let correctDescription = "Prediction for Total Eclipse of Jul 11,
         2010"
        let correctMap = "2010m07d11"
        let correctLink =
         "https://eclipse.gsfc.nasa
         .gov/SEbeselm/SEbeselm2001/SE2010Jul11Tbeselm.html"
    let  credit = app.staticTexts["NASA Source"]


    // By default we are testing EW with lat N  long W at height 0.  Code
     below will
    // have to be changed if this is different

    let latString = "50.1045"
    let longString = "73.8657"
    let wheelString = "2010 Jul 11 T"


    let correctC1angle: Double  = 220
    let semC2angle: Double  = 355
    let semC3angle : Double = 260
   // let correctC4angle: Double  = -1.0


    //NSLog(whenFormatter.string(from:engine.sunriseDate))
    //NSLog(whenFormatter.string(from:engine.sunsetDate))
    let noaaSunriseDate = noaaFmt.date(from: "07/11/10 12:51")!
    let noaaSunsetDate  = noaaFmt.date(from: "07/11/10 21:12")!
```

```
    // test magnitude
      let correctMag: Double  = 1.008


    // Test C1
      // Test C1
       let correctC1Date = decimalSec.date(from: "07/11/10 19:43:02.6")!

      let correctC1Alt :Double = 9.4

      let correctC1Az :Double = 322.4

      ///////////////////////////////// C2 ////////////////////////////////////////////////////////////////////////////////////////////

       let correctC2Date = decimalSec.date(from: "07/11/10 20:48:11.7")!

      let correctC2Alt :Double = 2.2

      let correctC2Az :Double = 309.1

      ///////////////////////////////// Mid ///////////////////////////////////////////////////////////////////////////////////////////

       let correctMidDate = decimalSec.date(from: "07/11/10 20:49:18.1")!

      let correctMidAlt :Double = 2

      let correctMidAz :Double = 308.9

      ///////////////////////////////// C3 ////////////////////////////////////////////////////////////////////////////////////////////

       let correctC3Date = decimalSec.date(from: "07/11/10 20:50:24.2")!

      let correctC3Alt :Double = 1.9

      let correctC3Az :Double = 308.7



     // Use XCTAssert and related functions to verify your tests produce
      the correct results.
    let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

    getState()

//    XCTAssert(!nasaMode)
    picker!.adjust(toPickerWheelValue: wheelString)
    getState()
    XCTAssert(pickerShown == wheelString)

    XCTAssert(degShowing)
```

```
latTextField!.tap()
latTextField!.clearAndEnterText(text: latString)
longTextField!.tap()
longTextField!.clearAndEnterText(text: longString)
heightTextField!.tap()
heightTextField!.clearAndEnterText(text: "0")

app.buttons["KeyDone"].tap()

sSwitch!.tap()
wSwitch!.tap()
getState()
XCTAssert(stripZero(latValue) ==  latString)
XCTAssert(stripZero(longValue) == longString)


app.buttons["PredictBut"].tap()

// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
 XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)




// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gNumbers)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gNumbers)
```

```
// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
XCTAssert(c4angleStr == "")


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)
```

```
// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <=  gEWMapNum)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
```

```
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <= gEWMapNum)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <= gEWMapNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff <= gEWNum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
```

```
    diff = abs( c3AzAsNumber! - correctC3Az)
    XCTAssert( diff <= gEWMapNum)

    ///////////////// c4 ////////////////////////////////////
    let c4timeQ = app.staticTexts["eventTime5"]
    XCTAssert(c4timeQ.exists)
    let c4timeStr = c4timeQ.label
  XCTAssert(c4timeStr == "Not Visible")

    let c4alt = app.staticTexts["eventAlt5"]
    XCTAssert(c4alt.exists)
    let c4altStr = c4alt.label
    XCTAssert(c4altStr == "")

    let c4az = app.staticTexts["eventAz5"]
    XCTAssert(c4az.exists)
    let c4azStr = c4az.label
    XCTAssert(c4azStr == "")


     XCTAssert(credit.exists)


    app.buttons["Map"].tap()
    //NSLog(app.debugDescription)
    XCTAssert(app.images[correctMap].exists)
    XCTAssert(app.buttons[correctLink].exists)
}
///////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////  2012  ///////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////


func testUI_2012_05_GE() throws {
    let decimalSec = DateFormatter() // used to convert numbers displayed
    let integerSec = DateFormatter() // used to convert answer numbers
     from maps
    integerSec.timeZone = TimeZone(abbreviation: "UTC")
    integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
    decimalSec.timeZone = TimeZone(abbreviation: "UTC")
    decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    let noaaFmt = DateFormatter() // used to convert answer numbers
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"

    var diff : Double

    ///////////////// Success criteria ////////////////////////////

        //            Summary View
```

```
    let correctNSText = "Location is within 6 nm (11 km) of centerline"
    let correctDescription = "Prediction for Annular Eclipse of May
     20, 2012"
    let correctMap = "2012m05d20"
    let correctLink =
     "https://eclipse.gsfc.nasa
     .gov/SEbeselm/SEbeselm2001/SE2012May20Abeselm.html"
    let  credit = app.staticTexts["NASA Source"]

// By default we are testing EW with lat N  long W at height 0.  Code
 below will
// have to be changed if this is different

let latString = "49.0942"
let longString = "176.273"
let wheelString = "2012 May 20 A"


let correctMag : Double = 0.972

let noaaSunriseDate = noaaFmt.date(from: "05/19/12 16:26")!
let noaaSunsetDate  = noaaFmt.date(from: "05/20/12 07:57")!

//          circs

let correctC1Date = decimalSec.date(from: "05/20/12 22:15:40.8")!
let correctC1Alt: Double  = 53.0
let correctC1Az: Double  = 131.0

let correctC2Date = decimalSec.date(from: "05/20/12 23:49:53.8")!
let correctC2Alt: Double  = 60.8
let correctC2Az: Double  = 169.6

let correctMidDate = decimalSec.date(from: "05/20/12 23:52:46.9")!
let correctMidAlt: Double  = 60.0
let correctMidAz: Double  = 171.0

let correctC3Date = decimalSec.date(from: "05/20/12 23:55:40.1")!
let correctC3Alt: Double  = 61.0
let correctC3Az: Double  = 172.3

let correctC4Date = decimalSec.date(from: "05/21/12 01:32:42.4")!
let correctC4Alt: Double  = 56.9
let correctC4Az: Double  = 216.6


 // Use XCTAssert and related functions to verify your tests produce
  the correct results.
let correctDur = correctC3Date.timeIntervalSince(correctC2Date)
```

```
    getState()

//    XCTAssert(!nasaMode)
    picker!.adjust(toPickerWheelValue: wheelString)
    getState()
    XCTAssert(pickerShown == wheelString)

    XCTAssert(degShowing)

     XCTAssert(nSwitchValue == true)
     XCTAssert(sSwitchValue == false)

    XCTAssert(eSwitchValue == true)
    XCTAssert(wSwitchValue == false)

    heightTextField!.tap()
    heightTextField!.clearAndEnterText(text: "0")

    app.buttons["KeyDone"].tap()
    getState()
    XCTAssert(stripZero(latValue) ==  latString)
    XCTAssert(stripZero(longValue) == longString)


    app.buttons["PredictBut"].tap()

    // OK we have a prediction
    app.buttons["Summary"].tap()

    //NSLog(app.debugDescription)
    let nstext = app.staticTexts["ns"]
    XCTAssert(nstext.exists)
     XCTAssert(nstext.label == correctNSText)

    let descript = app.staticTexts["description"]
    XCTAssert(descript.exists)
     XCTAssert(descript.label == correctDescription)




    // we have to search by value instead by field
    let magtext = app.staticTexts["magnitudeValue"]
    XCTAssert(magtext.exists)
    let magStr = magtext.label
    let magAsNumber = Double(magStr)
    XCTAssert(magAsNumber != nil)
    diff = abs( magAsNumber! - correctMag)
    XCTAssert( diff <= gNumbers)

    let durtext = app.staticTexts["durationValue"]
```

```
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gNumbers)



let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)
```

```
let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <=  gEWMapNum)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <= gEWMapNum)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
```

```
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <= gEWMapNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff <= gEWNum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff <= gEWMapNum)

///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
XCTAssert(c4timeQ.exists)
let c4timeStr = c4timeQ.label
let c4time = decimalSec.date(from: c4timeStr)
XCTAssert(c4time != nil)
diff = abs(c4time!.timeIntervalSince(correctC4Date))
XCTAssert( diff <= gEWCalc)

let c4alt = app.staticTexts["eventAlt5"]
XCTAssert(c4alt.exists)
let c4altStr = c4alt.label
let c4AltAsNumber = Double(c4altStr)
XCTAssert(c4AltAsNumber != nil)
diff = abs( c4AltAsNumber! - correctC4Alt)
XCTAssert( diff <= gEWNum)

let c4az = app.staticTexts["eventAz5"]
XCTAssert(c4az.exists)
let c4azStr = c4az.label
```

```
    let c4AzAsNumber = Double(c4azStr)
    XCTAssert(c4AzAsNumber != nil)
    diff = abs( c4AzAsNumber! - correctC4Az)
    XCTAssert( diff <= gEWMapNum)




    XCTAssert(credit.exists)


    app.buttons["Map"].tap()
    //NSLog(app.debugDescription)
    XCTAssert(app.images[correctMap].exists)
    XCTAssert(app.buttons[correctLink].exists)
}

func testUI_2012_11_GE() throws {
    let decimalSec = DateFormatter() // used to convert numbers displayed
    let integerSec = DateFormatter() // used to convert answer numbers
     from maps
    integerSec.timeZone = TimeZone(abbreviation: "UTC")
    integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
    decimalSec.timeZone = TimeZone(abbreviation: "UTC")
    decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"
    let noaaFmt = DateFormatter() // used to convert answer numbers
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"

    var diff : Double

    ///////////////// Success criteria //////////////////////////

        //            Summary View

        let correctNSText = "Location is within 6 nm (11 km) of centerline"
        let correctDescription = "Prediction for Total Eclipse of Nov 13,
         2012"
        let correctMap = "2012m11d13"
        let correctLink =
         "http://eclipsewise.com/solar/SEprime/2001-2100/SE2012Nov13Tprime
         .html"
        let credit = app.staticTexts["EW source"]


    // By default we are testing EW with lat N  long W at height 0.  Code
     below will
    // have to be changed if this is different

    let latString = "39.9561"
```

```
    let longString = "161.3368"
    let wheelString = "2012 Nov 13 T"


    let correctMag : Double = 1.025

    let correctC1angle: Double  =  295
    let semC2angle: Double =  85
    let semC3angle: Double  =  265
    let correctC4angle: Double  =  50

    let noaaSunriseDate = noaaFmt.date(from: "11/13/12 15:21")!
    let noaaSunsetDate = noaaFmt.date(from: "11/14/12 05:39")!

    //          circs

    let correctC1Date = decimalSec.date(from: "11/13/12 20:51:04.4")!
    let correctC1Alt: Double  = 59.6
    let correctC1Az: Double  = 51.6

    let correctC2Date = decimalSec.date(from: "11/13/12 22:09:47.4")!
    let correctC2Alt: Double  = 67.9
    let correctC2Az: Double  = 12.7

    let correctMidDate = decimalSec.date(from: "11/13/12 22:11:48.4")!
    let correctMidAlt: Double  = 68.0
    let correctMidAz: Double  = 11.4

    let correctC3Date = decimalSec.date(from: "11/13/12 22:13:49.5")!
    let correctC3Alt: Double  = 68.0
    let correctC3Az: Double  = 10.2

    let correctC4Date = decimalSec.date(from: "11/13/12 23:34:45.4")!
    let correctC4Alt: Double  = 64.2
    let correctC4Az: Double  = 322.4


     // Use XCTAssert and related functions to verify your tests produce
      the correct results.
    let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

    getState()

//    XCTAssert(!nasaMode)
    picker!.adjust(toPickerWheelValue: wheelString)
    getState()
    XCTAssert(pickerShown == wheelString)

    XCTAssert(degShowing)
    XCTAssert(stripZero(latValue) ==  latString)
    XCTAssert(stripZero(longValue) == longString)
```

```
 XCTAssert(nSwitchValue == false)
 XCTAssert(sSwitchValue == true)

XCTAssert(eSwitchValue == false)
XCTAssert(wSwitchValue == true)

heightTextField!.tap()
heightTextField!.clearAndEnterText(text: "0")

app.buttons["KeyDone"].tap()
getState()


app.buttons["PredictBut"].tap()

// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
 XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)




// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gNumbers)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gNumbers)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
```

```
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)
```

```
// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <=  gEWMapNum)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
```

```
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <= gEWMapNum)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <= gEWMapNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff <= gEWNum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
```

```
    XCTAssert( diff <= gEWMapNum)

    ///////////////// c4 ////////////////////////////////////
    let c4timeQ = app.staticTexts["eventTime5"]
    XCTAssert(c4timeQ.exists)
    let c4timeStr = c4timeQ.label
    let c4time = decimalSec.date(from: c4timeStr)
    XCTAssert(c4time != nil)
    diff = abs(c4time!.timeIntervalSince(correctC4Date))
    XCTAssert( diff <= gEWCalc)

    let c4alt = app.staticTexts["eventAlt5"]
    XCTAssert(c4alt.exists)
    let c4altStr = c4alt.label
    let c4AltAsNumber = Double(c4altStr)
    XCTAssert(c4AltAsNumber != nil)
    diff = abs( c4AltAsNumber! - correctC4Alt)
    XCTAssert( diff <= gEWNum)

    let c4az = app.staticTexts["eventAz5"]
    XCTAssert(c4az.exists)
    let c4azStr = c4az.label
    let c4AzAsNumber = Double(c4azStr)
    XCTAssert(c4AzAsNumber != nil)
    diff = abs( c4AzAsNumber! - correctC4Az)
    XCTAssert( diff <= gEWMapNum)


    XCTAssert(credit.exists)


    app.buttons["Map"].tap()
    //NSLog(app.debugDescription)
    XCTAssert(app.images[correctMap].exists)
    XCTAssert(app.buttons[correctLink].exists)
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////  2013  //////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////
func testUI_2013_GE() throws {
    let decimalSec = DateFormatter() // used to convert numbers displayed
    let integerSec = DateFormatter() // used to convert answer numbers
     from maps
    integerSec.timeZone = TimeZone(abbreviation: "UTC")
    integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
    decimalSec.timeZone = TimeZone(abbreviation: "UTC")
    decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    let noaaFmt = DateFormatter() // used to convert answer numbers
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
```

```
noaaFmt.dateFormat = "MM/dd/yy HH:mm"

var diff : Double

///////////////  Success criteria //////////////////////////

    //           Summary View

    let correctNSText = "Location is within 6 nm (11 km) of centerline"
    let correctDescription = "Prediction for Total Eclipse of Nov 3,
     2013"
    let correctMap = "2013m11d03"
    let correctLink =
     "https://eclipse.gsfc.nasa
     .gov/SEbeselm/SEbeselm2001/SE2013Nov03Hbeselm.html"
    let credit = app.staticTexts["NASA Source"]


// By default we are testing EW with lat N  long W at height 0.  Code
 below will
// have to be changed if this is different

let latString = "3.4906"
let longString = "11.6988"
let wheelString = "2013 Nov 03 H"


let correctMag : Double = 1.008

let correctC1angle: Double  =  20
let semC2angle: Double =  255
let semC3angle: Double  =  80
let correctC4angle: Double  =  315

let correctsunriseDate = integerSec.date(from: "11/03/13 06:32:50")!
let correctsunsetDate = integerSec.date(from: "11/03/13 18:27:50")!

//          circs

let correctC1Date = decimalSec.date(from: "11/03/13  11:04:23.4")!
let correctC1Alt: Double  = 61.7
let correctC1Az: Double  = 131.8

let correctC2Date = decimalSec.date(from: "11/03/13  12:45:39.1")!
let correctC2Alt: Double  = 70.9
let correctC2Az: Double  = 191.4

let correctMidDate = decimalSec.date(from: "11/03/13  12:46:28.9")!
let correctMidAlt: Double  = 70.9
let correctMidAz: Double  = 192.0
```

```
    let correctC3Date = decimalSec.date(from: "11/03/13 12:47:18.7")!
    let correctC3Alt: Double  = 70.8
    let correctC3Az: Double  = 192.5

    let correctC4Date = decimalSec.date(from: "11/03/13  14:29:37.2")!
    let correctC4Alt: Double  = 55.0
    let correctC4Az: Double  = 236.9


     // Use XCTAssert and related functions to verify your tests produce
      the correct results.
    let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

    getState()

//    XCTAssert(!nasaMode)
    picker!.adjust(toPickerWheelValue: wheelString)
    getState()
    XCTAssert(pickerShown == wheelString)

    XCTAssert(degShowing)

     XCTAssert(nSwitchValue == true)
     XCTAssert(sSwitchValue == false)

    XCTAssert(eSwitchValue == false)
    XCTAssert(wSwitchValue == true)

     XCTAssert(stripZero(latValue) ==  latString)
    XCTAssert(stripZero(longValue) == longString)


    app.buttons["PredictBut"].tap()

    // OK we have a prediction
    app.buttons["Summary"].tap()

    //NSLog(app.debugDescription)
    let nstext = app.staticTexts["ns"]
    XCTAssert(nstext.exists)
     XCTAssert(nstext.label == correctNSText)

    let descript = app.staticTexts["description"]
    XCTAssert(descript.exists)
     XCTAssert(descript.label == correctDescription)




    // we have to search by value instead by field
    let magtext = app.staticTexts["magnitudeValue"]
```

```
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gNumbers)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gNumbers)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
```

```
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(correctsunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(correctsunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <=  gEWMapNum)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
```

```
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <= gEWMapNum)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <= gEWMapNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
```

```
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff <= gEWNum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff <= gEWMapNum)

///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
XCTAssert(c4timeQ.exists)
let c4timeStr = c4timeQ.label
let c4time = decimalSec.date(from: c4timeStr)
XCTAssert(c4time != nil)
diff = abs(c4time!.timeIntervalSince(correctC4Date))
XCTAssert( diff <= gEWCalc)

let c4alt = app.staticTexts["eventAlt5"]
XCTAssert(c4alt.exists)
let c4altStr = c4alt.label
let c4AltAsNumber = Double(c4altStr)
XCTAssert(c4AltAsNumber != nil)
diff = abs( c4AltAsNumber! - correctC4Alt)
XCTAssert( diff <= gEWNum)

let c4az = app.staticTexts["eventAz5"]
XCTAssert(c4az.exists)
let c4azStr = c4az.label
let c4AzAsNumber = Double(c4azStr)
XCTAssert(c4AzAsNumber != nil)
diff = abs( c4AzAsNumber! - correctC4Az)
XCTAssert( diff <= gEWMapNum)


XCTAssert(credit.exists)


app.buttons["Map"].tap()
//NSLog(app.debugDescription)
```

```
    XCTAssert(app.images[correctMap].exists)
    XCTAssert(app.buttons[correctLink].exists)
}
///////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////  2015 /////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////
func testUI_2015_GE() throws {
    let decimalSec = DateFormatter() // used to convert numbers displayed
    let integerSec = DateFormatter() // used to convert answer numbers
     from maps
    integerSec.timeZone = TimeZone(abbreviation: "UTC")
    integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
    decimalSec.timeZone = TimeZone(abbreviation: "UTC")
    decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    let noaaFmt = DateFormatter() // used to convert answer numbers
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"

    var diff : Double

    ///////////////// Success criteria ////////////////////////

        //            Summary View

        let correctNSText = "Location is within 6 nm (11 km) of centerline"
        let correctDescription = "Prediction for Total Eclipse of Mar 20,
         2015"
        let correctMap = "2015m03d20"
        let correctLink =
         "https://eclipse.gsfc.nasa
         .gov/SEbeselm/SEbeselm2001/SE2015Mar20Tbeselm.html"
        let credit = app.staticTexts["NASA Source"]


    // By default we are testing EW with lat N  long W at height 0.  Code
     below will
    // have to be changed if this is different

    let latString = "64.4319"
    let longString = "6.6472"
    let wheelString = "2015 Mar 20 T"


    let correctMag : Double = 1.022

    let correctC1angle: Double  =  90
    let semC2angle: Double =  270
    let semC3angle: Double  =  90
    let correctC4angle: Double  =  280
```

```
    let noaaSunriseDate = noaaFmt.date(from: "03/20/15 06:29")!
    let noaaSunsetDate = noaaFmt.date(from: "03/20/15 18:41")!

    //          circs

    let correctC1Date = decimalSec.date(from: "03/20/15  08:43:06.4")!
    let correctC1Alt: Double  = 13.1
    let correctC1Az: Double  = 119.7

    let correctC2Date = decimalSec.date(from: "03/20/15  09:44:15.9")!
    let correctC2Alt: Double  = 18.4
    let correctC2Az: Double  = 134.6

    let correctMidDate = decimalSec.date(from: "03/20/15  09:45:39.2")!
    let correctMidAlt: Double  = 18.5
    let correctMidAz: Double  = 135.0

    let correctC3Date = decimalSec.date(from: "03/20/15  09:47:02.8")!
    let correctC3Alt: Double  = 18.6
    let correctC3Az: Double  = 135.3

    let correctC4Date = decimalSec.date(from: "03/20/15  10:50:35.7")!
    let correctC4Alt: Double  = 22.7
    let correctC4Az: Double  = 151.8


     // Use XCTAssert and related functions to verify your tests produce
      the correct results.
    let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

    getState()

//    XCTAssert(!nasaMode)
    picker!.adjust(toPickerWheelValue: wheelString)
    getState()
    XCTAssert(pickerShown == wheelString)

    XCTAssert(degShowing)

     XCTAssert(nSwitchValue == true)
     XCTAssert(sSwitchValue == false)

    XCTAssert(eSwitchValue == false)
    XCTAssert(wSwitchValue == true)


    getState()
    XCTAssert(stripZero(latValue) ==  latString)
    XCTAssert(stripZero(longValue) == longString)
```

```
app.buttons["PredictBut"].tap()

// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
 XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)




// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gNumbers)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gNumbers)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)
```

```
let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)
```

```
let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <=  gEWMapNum)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <= gEWMapNum)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
```

```
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <= gEWMapNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff <= gEWNum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff <= gEWMapNum)

///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
XCTAssert(c4timeQ.exists)
let c4timeStr = c4timeQ.label
let c4time = decimalSec.date(from: c4timeStr)
XCTAssert(c4time != nil)
diff = abs(c4time!.timeIntervalSince(correctC4Date))
XCTAssert( diff <= gEWCalc)

let c4alt = app.staticTexts["eventAlt5"]
XCTAssert(c4alt.exists)
let c4altStr = c4alt.label
```

```
    let c4AltAsNumber = Double(c4altStr)
    XCTAssert(c4AltAsNumber != nil)
    diff = abs( c4AltAsNumber! - correctC4Alt)
    XCTAssert( diff <= gEWNum)

    let c4az = app.staticTexts["eventAz5"]
    XCTAssert(c4az.exists)
    let c4azStr = c4az.label
    let c4AzAsNumber = Double(c4azStr)
    XCTAssert(c4AzAsNumber != nil)
    diff = abs( c4AzAsNumber! - correctC4Az)
    XCTAssert( diff <= gEWMapNum)


    XCTAssert(credit.exists)


    app.buttons["Map"].tap()
    //NSLog(app.debugDescription)
    XCTAssert(app.images[correctMap].exists)
    XCTAssert(app.buttons[correctLink].exists)
}

///////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////  2016  ///////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////

func testUI_EW2016_GE() throws {
    let decimalSec = DateFormatter() // used to convert numbers displayed
    let integerSec = DateFormatter() // used to convert answer numbers
     from maps
    integerSec.timeZone = TimeZone(abbreviation: "UTC")
    integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
    decimalSec.timeZone = TimeZone(abbreviation: "UTC")
    decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"
    let noaaFmt = DateFormatter() // used to convert answer numbers
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"

    ///////////////  Success criteria ////////////////////////

        //           Summary View

        let correctNSText = "Location is within 6 nm (11 km) of centerline"
        let correctDescription = "Prediction for Total Eclipse of Mar 9,
         2016"

    // By default we are testing EW with lat N  long W at height 0.  Code
     below will
    // have to be changed if this is different
```

```
        let latString = "10.0669"
        let longString = "148.7015"
        let wheelString = "2016 Mar 09 T"

        let correctC1angle:Double = 60.0
        let semC2angle:Double = 270.0
        let semC3angle:Double = 115.0
        let correctC4angle:Double = 355.0

        var dmsloc = dms(10,4,0)
        XCTAssert( (Double(latString)! - dmsloc) < 0.0003)
        dmsloc = dms(148,42,5)
        XCTAssert( (Double(longString)! - dmsloc) < 0.0003)


        let noaaSunriseDate = noaaFmt.date(from: "03/08/16 20:16")!
        let noaaSunsetDate  = noaaFmt.date(from: "03/09/16 08:16")!


        let correctMag = 1.023


        // Test C1
        let correctC1Date = integerSec.date(from: "03/09/16 00:24:24")!
        let correctC1Alt:Double = 59
        let correctC1Az:Double = 116.3

        let correctC2Date = integerSec.date(from: "03/09/16 01:54:47")!
        let correctC2Alt:Double = 75
        let correctC2Az:Double = 159.9

        let correctMidDate = integerSec.date(from: "03/09/16 01:56:52")!
        let correctMidAlt:Double = 75
        let correctMidAz:Double = 161.8

        let correctC3Date = integerSec.date(from: "03/09/16 01:58:57")!
        let correctC3Alt:Double = 75.0
        let correctC3Az:Double = 163.7

        let correctC4Date = integerSec.date(from: "03/09/16 03:30:08")!
        let correctC4Alt:Double = 67
        let correctC4Az:Double = 233.0

        var diff : Double
        // Use XCTAssert and related functions to verify your tests produce
         the correct results.
        let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

        getState()

//        XCTAssert(!nasaMode)
```

```
picker!.adjust(toPickerWheelValue: wheelString)
getState()
XCTAssert(pickerShown == wheelString)

XCTAssert(degShowing)

 XCTAssert(nSwitchValue == true)
 XCTAssert(sSwitchValue == false)

XCTAssert(eSwitchValue == true)
XCTAssert(wSwitchValue == false)

// GE test
//latTextField!.tap()
//latTextField!.clearAndEnterText(text: latString)
//longTextField!.tap()
//longTextField!.clearAndEnterText(text: longString)
heightTextField!.tap()
heightTextField!.clearAndEnterText(text: "0")

app.buttons["KeyDone"].tap()
getState()
XCTAssert(stripZero(latValue) ==  latString)
XCTAssert(stripZero(longValue) == longString)


app.buttons["PredictBut"].tap()

// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
 XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)




// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gEWNum)
```

```
let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gEWNum)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)


// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
```

```
 diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)!
 diff = abs(c1time.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <= gEWNum)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)!
 diff = abs(c2time.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)
```

```
let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <= gEWNum)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)!
diff = abs(midtime.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <= gEWNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
```

```
        let c3altStr = c3alt.label
        let c3AltAsNumber = Double(c3altStr)
        XCTAssert(c3AltAsNumber != nil)
        diff = abs( c3AltAsNumber! - correctC3Alt)
        XCTAssert( diff <= gEWNum)

        let c3az = app.staticTexts["eventAz4"]
        XCTAssert(c3az.exists)
        let c3azStr = c3az.label
        let c3AzAsNumber = Double(c3azStr)
        XCTAssert(c3AzAsNumber != nil)
        diff = abs( c3AzAsNumber! - correctC3Az)
        XCTAssert( diff <= gEWNum)

        ///////////////// c4 ////////////////////////////////////
        let c4timeQ = app.staticTexts["eventTime5"]
        XCTAssert(c4timeQ.exists)
        let c4timeStr = c4timeQ.label
        let c4time = decimalSec.date(from: c4timeStr)
        XCTAssert(c4time != nil)
        diff = abs(c4time!.timeIntervalSince(correctC4Date))
        XCTAssert( diff <= gEWCalc)

        let c4alt = app.staticTexts["eventAlt5"]
        XCTAssert(c4alt.exists)
        let c4altStr = c4alt.label
        let c4AltAsNumber = Double(c4altStr)
        XCTAssert(c4AltAsNumber != nil)
        diff = abs( c4AltAsNumber! - correctC4Alt)
        XCTAssert( diff <= gEWNum)

        let c4az = app.staticTexts["eventAz5"]
        XCTAssert(c4az.exists)
        let c4azStr = c4az.label
        let c4AzAsNumber = Double(c4azStr)
        XCTAssert(c4AzAsNumber != nil)
        diff = abs( c4AzAsNumber! - correctC4Az)
        XCTAssert( diff <= gEWNum)

        let credit = app.staticTexts["EW source"]
        XCTAssert(credit.exists)


        app.buttons["Map"].tap()
        //NSLog(app.debugDescription)
        XCTAssert(app.images["2016m03d09"].exists)
        XCTAssert(app.buttons
         ["http://eclipsewise.com/solar/SEprime/2001-2100/SE2016Mar09Tprime
         .html"].exists)

}
```

```
///////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////  2017  ////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////

func testUI_EW2017_Madras() throws {
    var diff: Double
    let decimalSec = DateFormatter() // used to convert numbers displayed
     let integerSec = DateFormatter() // used to convert answer numbers
      from maps
     integerSec.timeZone = TimeZone(abbreviation: "UTC")
     integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
     decimalSec.timeZone = TimeZone(abbreviation: "UTC")
     decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"
    let noaaFmt = DateFormatter() // used to convert answer numbers
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"


    let latString = "44.7087"
    let longString = "121.0939"
    let wheelString = "2017 Aug 21 T"

    ///////////////// Success criteria //////////////////////////

        //           Summary View

        let correctNSText = "Location is within 6 nm (11 km) of centerline"
        let correctDescription = "Prediction for Total Eclipse of Aug 21,
         2017"

    let correctMag = 1.013
    let correctC1angle = 33.0
    let semC2angle = 217.0
    let semC3angle = 35.0
    let correctC4angle = 227.0
    let noaaSunriseDate = noaaFmt.date(from: "08/21/17 13:14")!
    let noaaSunsetDate = noaaFmt.date(from: "08/22/17 03:00")!

    //          circs
    let correctC1Date = decimalSec.date(from: "08/21/17 16:06:48.7")!
    let correctC1Alt = 29.4
    let correctC1Az = 103.0

    let correctC2Date = decimalSec.date(from: "08/21/17 17:19:40.3")!
    let correctC2Alt = 41.5
    let correctC2Az = 119.2

    let correctMidDate = decimalSec.date(from: "08/21/17 17:20:42.0")!
    let correctMidAlt = 41.6
```

```
        let correctMidAz = 119.5

        let correctC3Date = decimalSec.date(from: "08/21/17 17:21:44.0")!
        let correctC3Alt = 41.8
        let correctC3Az = 119.7

        let correctDur = correctC3Date.timeIntervalSince(correctC2Date)


        let correctC4Date = decimalSec.date(from: "08/21/17 18:41:09.3")!
        let correctC4Alt = 52.3
        let correctC4Az = 144.0


         // Use XCTAssert and related functions to verify your tests produce
          the correct results.

        getState()

//        XCTAssert(!nasaMode)
        picker!.adjust(toPickerWheelValue: wheelString)
        getState()
        XCTAssert(pickerShown == wheelString)

        XCTAssert(degShowing)

         XCTAssert(nSwitchValue == true)
         XCTAssert(sSwitchValue == false)

        XCTAssert(eSwitchValue == false)
        XCTAssert(wSwitchValue == true)

        latTextField!.tap()
        latTextField!.clearAndEnterText(text: latString)
        longTextField!.tap()
        longTextField!.clearAndEnterText(text: longString)
        heightTextField!.tap()
        heightTextField!.clearAndEnterText(text: "740")

        app.buttons["KeyDone"].tap()
        getState()
        XCTAssert(stripZero(latValue) ==  latString)
        XCTAssert(stripZero(longValue) == longString)


        app.buttons["PredictBut"].tap()

        // OK we have a prediction
        app.buttons["Summary"].tap()

        //NSLog(app.debugDescription)
```

```
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
XCTAssert(descript.label == correctDescription)



// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gEWNum)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gEWNum)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)
```

```
// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)
diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
```

```
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <= gNASANum)
///////////////// c2 ///////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <= gNASANum)

///////////////// mid ///////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)!
diff = abs(midtime.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)
```

```
let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <= gNASANum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff <= gEWNum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff <= gNASANum)

///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
XCTAssert(c4timeQ.exists)
let c4timeStr = c4timeQ.label
let c4time = decimalSec.date(from: c4timeStr)
XCTAssert(c4time != nil)
diff = abs(c4time!.timeIntervalSince(correctC4Date))
XCTAssert( diff <= gEWCalc)

let c4alt = app.staticTexts["eventAlt5"]
XCTAssert(c4alt.exists)
let c4altStr = c4alt.label
let c4AltAsNumber = Double(c4altStr)
XCTAssert(c4AltAsNumber != nil)
diff = abs( c4AltAsNumber! - correctC4Alt)
XCTAssert( diff <= gEWNum)

let c4az = app.staticTexts["eventAz5"]
```

```
    XCTAssert(c4az.exists)
    let c4azStr = c4az.label
    let c4AzAsNumber = Double(c4azStr)
    XCTAssert(c4AzAsNumber != nil)
    diff = abs( c4AzAsNumber! - correctC4Az)
    XCTAssert( diff <= gNASANum)


}

func testUI_EW2017_GE() throws {
    var diff: Double
    let decimalSec = DateFormatter() // used to convert numbers displayed
    let integerSec = DateFormatter() // used to convert answer numbers
     from maps
    integerSec.timeZone = TimeZone(abbreviation: "UTC")
    integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
    decimalSec.timeZone = TimeZone(abbreviation: "UTC")
    decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"
    let noaaFmt = DateFormatter() // used to convert answer numbers
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"

    ///////////////// Success criteria //////////////////////////

    //            Summary View

    let correctNSText = "Location is within 6 nm (11 km) of centerline"
    let correctDescription = "Prediction for Total Eclipse of Aug 21, 2017"

    // By default we are testing EW with lat N  long W at height 0.  Code
     below will
    // have to be changed if this is different

    let latString = "36.96668"
    let longString = "87.67167"
    let wheelString = "2017 Aug 21 T"

    var dmsloc = dms(36,58,0)
    XCTAssert( (Double(latString)! - dmsloc) < 0.0003)
    dmsloc = dms(87,40,18)
    XCTAssert( (Double(longString)! - dmsloc) < 0.0003)


    let correctMag:Double = 1.015
    let correctC1angle:Double = 40.0
    let semC2angle:Double = 260.0
    let semC3angle:Double = 70.0
    let correctC4angle:Double = 290.0

    let noaaSunriseDate = noaaFmt.date(from: "08/21/17 11:13")!
```

```
let noaaSunsetDate = noaaFmt.date(from: "08/22/17 00:34")!

//          circs

let correctC1Date = integerSec.date(from: "08/21/17 16:56:05")!
let correctC1Alt:Double = 62
let correctC1Az:Double = 149.0

let correctC2Date = integerSec.date(from: "08/21/17 18:24:12")!
let correctC2Alt:Double = 64
let correctC2Az:Double = 197.2

let correctMidDate = integerSec.date(from: "08/21/17 18:25:32")!
let correctMidAlt:Double = 64
let correctMidAz:Double = 197.9

let correctC3Date = integerSec.date(from: "08/21/17 18:26:52")!
let correctC3Alt:Double = 64
let correctC3Az:Double = 198.6

let correctC4Date = integerSec.date(from: "08/21/17 19:51:16")!
let correctC4Alt:Double = 54
let correctC4Az:Double = 234.0


// Use XCTAssert and related functions to verify your tests produce
 the correct results.
let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

getState()

//        XCTAssert(!nasaMode)
picker!.adjust(toPickerWheelValue: wheelString)
getState()
XCTAssert(pickerShown == wheelString)

XCTAssert(degShowing)

XCTAssert(nSwitchValue == true)
XCTAssert(sSwitchValue == false)

XCTAssert(eSwitchValue == false)
XCTAssert(wSwitchValue == true)

// This is a GE test
//latTextField!.tap()
//latTextField!.clearAndEnterText(text: latString)
//longTextField!.tap()
//longTextField!.clearAndEnterText(text: longString)
heightTextField!.tap()
heightTextField!.clearAndEnterText(text: "0")
```

```
app.buttons["KeyDone"].tap()
getState()
XCTAssert(stripZero(latValue) ==  latString)
XCTAssert(stripZero(longValue) == longString)


app.buttons["PredictBut"].tap()

// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
XCTAssert(descript.label == correctDescription)



// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gEWNum)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gEWNum)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
```

```
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)


// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
```

```
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <=  gEWMapNum)

///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <=  gEWMapNum)

///////////////// mid ////////////////////////////////////
```

```
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <=  gEWMapNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff <= gEWNum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff <=  gEWMapNum)

///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
```

```
    XCTAssert(c4timeQ.exists)
    let c4timeStr = c4timeQ.label
    let c4time = decimalSec.date(from: c4timeStr)
    XCTAssert(c4time != nil)
    diff = abs(c4time!.timeIntervalSince(correctC4Date))
    XCTAssert( diff <= gEWCalc)

    let c4alt = app.staticTexts["eventAlt5"]
    XCTAssert(c4alt.exists)
    let c4altStr = c4alt.label
    let c4AltAsNumber = Double(c4altStr)
    XCTAssert(c4AltAsNumber != nil)
    diff = abs( c4AltAsNumber! - correctC4Alt)
    XCTAssert( diff <= gEWNum)

    let c4az = app.staticTexts["eventAz5"]
    XCTAssert(c4az.exists)
    let c4azStr = c4az.label
    let c4AzAsNumber = Double(c4azStr)
    XCTAssert(c4AzAsNumber != nil)
    diff = abs( c4AzAsNumber! - correctC4Az)
    XCTAssert( diff <=  gEWMapNum)

    let credit = app.staticTexts["EW source"]
    XCTAssert(credit.exists)


    app.buttons["Map"].tap()
    //NSLog(app.debugDescription)
    XCTAssert(app.images["2017m08d21"].exists)
    XCTAssert(app.buttons
     ["http://eclipsewise.com/oh/ec2017.html#SE2017Aug21T"].exists)
}

#if NoNASA
func _testUI_NASA2017_GE() throws {
    let decimalSec = DateFormatter() // used to convert numbers displayed
     let integerSec = DateFormatter() // used to convert answer numbers
      from maps
     integerSec.timeZone = TimeZone(abbreviation: "UTC")
     integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
     decimalSec.timeZone = TimeZone(abbreviation: "UTC")
     decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    ///////////////// Success criteria //////////////////////////

        //           Summary View

         let correctNSText = "Location is within 6 nm (11 km) of
          centerline"
```

```
    let correctDescription = "Prediction for Total Eclipse of Aug 21,
     2017"


let latString = "36.96668"
let longString = "87.67167"
let wheelString = "N2017 Aug 21 T"


let correctMag = 1.013
let correctC1angle = 40.0
let semC2angle = 260.0
let semC3angle = 70.0
let correctC4angle = 290.0
let correctsunriseDate = decimalSec.date(from: "08/21/17 11:15:59.1")!
let correctsunsetDate = decimalSec.date(from: "08/22/17 00:30:52.1")!

//          circs
let correctC1Date = decimalSec.date(from: "08/21/17 16:56:05.5")!
let correctC1Alt = 61.8
let correctC1Az = 149.0

let correctC2Date = decimalSec.date(from: "08/21/17 18:24:11.9")!
let correctC2Alt = 64.0
let correctC2Az = 197.2

let correctMidDate = decimalSec.date(from: "08/21/17 18:25:32.0")!
let correctMidAlt = 63.9
let correctMidAz = 197.9

let correctC3Date = decimalSec.date(from: "08/21/17 18:26:51.9")!
let correctC3Alt = 63.8
let correctC3Az = 198.7

let correctC4Date = decimalSec.date(from: "08/21/17 19:51:16.0")!
let correctC4Alt = 53.6
let correctC4Az = 234.0

let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

var diff : Double
// Use XCTAssert and related functions to verify your tests produce
 the correct results.

getState()

XCTAssert(!nasaMode)
nasaSwitch!.tap()
getState()
XCTAssert(nasaMode)
```

```
 picker!.adjust(toPickerWheelValue: wheelString)
 getState()
 XCTAssert(pickerShown == wheelString)

 XCTAssert(degShowing)

  XCTAssert(nSwitchValue == true)
  XCTAssert(sSwitchValue == false)

 XCTAssert(eSwitchValue == false)
 XCTAssert(wSwitchValue == true)

 // This is a GE test
 //latTextField!.tap()
// latTextField!.clearAndEnterText(text: "36.9664")
// longTextField!.tap()
// longTextField!.clearAndEnterText(text: "87.6709")
 heightTextField!.tap()
 heightTextField!.clearAndEnterText(text: "0")

 app.buttons["KeyDone"].tap()
 getState()
 XCTAssert(stripZero(latValue) ==  latString)
 XCTAssert(stripZero(longValue) == longString)


 app.buttons["PredictBut"].tap()

 // OK we have a prediction
 app.buttons["Summary"].tap()

 //NSLog(app.debugDescription)
 let nstext = app.staticTexts["ns"]
 XCTAssert(nstext.exists)
  XCTAssert(nstext.label == correctNSText)

 let descript = app.staticTexts["description"]
 XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)



 // we have to search by value instead by field
 let magtext = app.staticTexts["magnitudeValue"]
 XCTAssert(magtext.exists)
 let magStr = magtext.label
 let magAsNumber = Double(magStr)
 XCTAssert(magAsNumber != nil)
 diff = abs( magAsNumber! - correctMag)
 XCTAssert( diff <= gEWNum)
```

```
let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gEWNum)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)


// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = integerSec.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(correctsunriseDate.timeIntervalSince(sunrisetime!))
```

```
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = integerSec.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(correctsunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff  <= gNASAMapTimes)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gNASANum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <= gNASANum)

/////////////////  c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
```

```
XCTAssert( diff <= gNASAMapTimes)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gNASANum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <= gNASANum)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gNASAMapTimes)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gNASANum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <= gNASANum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gNASAMapTimes)
```

```
let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff <= gNASANum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff <= gNASANum)

///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
XCTAssert(c4timeQ.exists)
let c4timeStr = c4timeQ.label
let c4time = decimalSec.date(from: c4timeStr)
XCTAssert(c4time != nil)
diff = abs(c4time!.timeIntervalSince(correctC4Date))
XCTAssert( diff <= gNASAMapTimes)

let c4alt = app.staticTexts["eventAlt5"]
XCTAssert(c4alt.exists)
let c4altStr = c4alt.label
let c4AltAsNumber = Double(c4altStr)
XCTAssert(c4AltAsNumber != nil)
diff = abs( c4AltAsNumber! - correctC4Alt)
XCTAssert( diff <= gNASANum)

let c4az = app.staticTexts["eventAz5"]
XCTAssert(c4az.exists)
let c4azStr = c4az.label
let c4AzAsNumber = Double(c4azStr)
XCTAssert(c4AzAsNumber != nil)
diff = abs( c4AzAsNumber! - correctC4Az)
XCTAssert( diff <= gNASANum)

let credit = app.staticTexts["EW source"]
XCTAssert(credit.exists)


app.buttons["Map"].tap()
//NSLog(app.debugDescription)
XCTAssert(app.images["2017m08d21"].exists)
```

```
    XCTAssert(app.buttons
     ["https://eclipse.gsfc.nasa
     .gov/SEbeselm/SEbeselm2001/SE2017Aug21Tbeselm.html"].exists)
}
#endif


///////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////  2019  ////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////

func testUI_EW2019_VicuñaChile() throws {
    var diff: Double
    let decimalSec = DateFormatter() // used to convert numbers displayed
     let integerSec = DateFormatter() // used to convert answer numbers
      from maps
     integerSec.timeZone = TimeZone(abbreviation: "UTC")
     integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
     decimalSec.timeZone = TimeZone(abbreviation: "UTC")
     decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"
    let noaaFmt = DateFormatter() // used to convert answer numbers from
     maps
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"

   // let latString = see below
   // let longString = see below
    let wheelString = "2019 Jul 02 T"

        //           Summary View

        let correctNSText = "Location is south of the centerline"
        let correctDescription = "Prediction for Total Eclipse of Jul 2,
         2019"

    let correctMag:Double = 1.012
    let correctDur:Double = 120.0 + 24.0
    let correctC1angle:Double = 227.0
    let semC2angle:Double = 17.0
    let semC3angle:Double = 240.0
    let correctC4angle:Double = 34.0
    let noaaSunriseDate = noaaFmt.date(from: "07/02/19 11:40")!
    let noaaSunsetDate = noaaFmt.date(from: "07/02/19 21:55")!

    //          circs
    let correctC1Date = integerSec.date(from: "07/02/19 19:23:24")!
    let correctC1Alt:Double = 25
    let correctC1Az:Double = 320.1

    let correctC2Date = integerSec.date(from: "07/02/19 20:38:35")!
    let correctC2Alt:Double = 13
```

```
        let correctC2Az:Double = 306.8

        let correctMidDate = integerSec.date(from: "07/02/19 20:39:47")!
        let correctMidAlt:Double = 13
        let correctMidAz:Double = 306.6

        let correctC3Date = integerSec.date(from: "07/02/19 20:40:59")!
        let correctC3Alt:Double = 13
        let correctC3Az = 306.4

        let correctC4Date = integerSec.date(from: "07/02/19 21:46:43")!
        let correctC4Alt:Double = 1
        let correctC4Az:Double = 297.3


        // Use XCTAssert and related functions to verify your tests produce
         the correct results.

        getState()

//        XCTAssert(!nasaMode)
        picker!.adjust(toPickerWheelValue: wheelString)
        getState()
        XCTAssert(pickerShown == wheelString)

        XCTAssert(degShowing)

         XCTAssert(nSwitchValue == false)
         XCTAssert(sSwitchValue == true)

        XCTAssert(eSwitchValue == false)
        XCTAssert(wSwitchValue == true)

        latTextField!.tap()
        latTextField!.clearAndEnterText(text: dmsStr(30,2,42))
        longTextField!.tap()
        longTextField!.clearAndEnterText(text: dmsStr(70,43,1))
        heightTextField!.tap()
        heightTextField!.clearAndEnterText(text: "589")
        app.buttons["KeyDone"].tap()
        getState()
        XCTAssert(stripZero(latValue) ==  stripZero(dmsStr(30,2,42)))
        XCTAssert(stripZero(longValue) == stripZero(dmsStr(70,43, 1)))


        app.buttons["PredictBut"].tap()

        // OK we have a prediction
        app.buttons["Summary"].tap()

        //NSLog(app.debugDescription)
```

```
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
 XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)



// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gEWNum)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gEWNum)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)
```

```
// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
```

```
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <= gEWMapNum)

///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <= gEWMapNum)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
```

```
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <=  gEWMapNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff <= gEWNum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff <=  gEWMapNum)

///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
XCTAssert(c4timeQ.exists)
let c4timeStr = c4timeQ.label
let c4time = decimalSec.date(from: c4timeStr)
XCTAssert(c4time != nil)
diff = abs(c4time!.timeIntervalSince(correctC4Date))
XCTAssert( diff <= gEWCalc)

let c4alt = app.staticTexts["eventAlt5"]
XCTAssert(c4alt.exists)
let c4altStr = c4alt.label
let c4AltAsNumber = Double(c4altStr)
XCTAssert(c4AltAsNumber != nil)
diff = abs( c4AltAsNumber! - correctC4Alt)
```

```
    XCTAssert( diff <= gEWNum)

    let c4az = app.staticTexts["eventAz5"]
    XCTAssert(c4az.exists)
    let c4azStr = c4az.label
    let c4AzAsNumber = Double(c4azStr)
    XCTAssert(c4AzAsNumber != nil)
    diff = abs( c4AzAsNumber! - correctC4Az)
    XCTAssert( diff <=  gEWMapNum)


}
// C4 below horizon on south border of totality
// Chascomús is near the point where totality does not complete before
 sunset
func testUI_EW2019_ChascomúsAR_map() throws {
    var diff: Double
    let decimalSec = DateFormatter() // used to convert numbers displayed
     let integerSec = DateFormatter() // used to convert answer numbers
      from maps
     integerSec.timeZone = TimeZone(abbreviation: "UTC")
     integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
     decimalSec.timeZone = TimeZone(abbreviation: "UTC")
     decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    let noaaFmt = DateFormatter() // used to convert answer numbers
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"

    ///////////////// Success criteria //////////////////////////

        //           Summary View

         let correctNSText = "Location is within 6 nm (11 km) of
          centerline"
        let correctDescription = "Prediction for Total Eclipse of Jul 2,
         2019"

    let correctMag = 1.015
    let correctDur = 123.4
    let correctC1angle = 223.0
    let semC2angle = 43.0
    let semC3angle = 212.0
    //let correctC4angle = 41.0
    let correctsunriseDate = decimalSec.date(from: "07/02/19 11:02:19.9")!
    let correctsunsetDate = decimalSec.date(from: "07/02/19 20:49:55.4")!

    //          circs
    let correctC1Date = decimalSec.date(from: "07/02/2019 19:35:57.5")!
    let correctC1Alt = 11.7
    let correctC1Az = 309.7
```

```
        let correctC2Date = decimalSec.date(from: "07/02/2019 20:42:38.0")!
        let correctC2Alt = 0.5
        let correctC2Az = 299.1

        let correctMidDate = decimalSec.date(from: "07/02/2019 20:43:39.8")!
        let correctMidAlt = 0.3
        let correctMidAz = 299.0

        let correctC3Date = decimalSec.date(from: "07/02/2019 20:44:41.4")!
        let correctC3Alt = 0.1
        let correctC3Az = 298.8

        //let correctC4Date = whenFormatter.date(from: "12/14/2020
         17:40:02.1")!
        //let correctC4Alt = 67.0
        //let correctC4Az = 312.4


         // Use XCTAssert and related functions to verify your tests produce
          the correct results.

        getState()

//        XCTAssert(!nasaMode)
        picker!.adjust(toPickerWheelValue: "2019 Jul 02 T")
        getState()
        XCTAssert(pickerShown == "2019 Jul 02 T")

        XCTAssert(degShowing)

         XCTAssert(nSwitchValue == false)
         XCTAssert(sSwitchValue == true)

        XCTAssert(eSwitchValue == false)
        XCTAssert(wSwitchValue == true)

        latTextField!.tap()
        latTextField!.clearAndEnterText(text: "35.5986")
        longTextField!.tap()
        longTextField!.clearAndEnterText(text: "58.0005")
        app.buttons["KeyDone"].tap()
        getState()
        XCTAssert(stripZero(latValue) == "35.5986")
        XCTAssert(stripZero(longValue) == "58.0005")


        XCTAssert(predictEnabled)

        app.buttons["PredictBut"].tap()
```

```
// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)



// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gEWNum)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gEWNum)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
```

```
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( diff < gVisualMeasurement)


// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
XCTAssert(c4angleStr == "")


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(correctsunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(correctsunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff < gpassMap)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
```

```
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <=  gEWMapNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff < gVisualMeasurement)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff < gpassMap)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <=  gEWMapNum)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff < gpassMap)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
```

```
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff < gVisualMeasurement)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff < gVisualMeasurement)

///////////////// c3 ///////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff < gpassMap)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff < gVisualMeasurement)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff < gVisualMeasurement)

///////////////// c4 ///////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
XCTAssert(c4timeQ.exists)
let c4timeStr = c4timeQ.label
XCTAssert( c4timeStr == "Not Visible")

let c4alt = app.staticTexts["eventAlt5"]
XCTAssert(c4alt.exists)
let c4altStr = c4alt.label
XCTAssert( c4altStr == "")

let c4az = app.staticTexts["eventAz5"]
XCTAssert(c4az.exists)
let c4azStr = c4az.label
XCTAssert( c4azStr == "")
```

```
}
///////////////////////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////  2020  ////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////

// Test GE for 2020.  EWPredict verified by unit test
func testUI_EW2020_GE() throws {
    var diff: Double
    let decimalSec = DateFormatter() // used to convert numbers displayed
     let integerSec = DateFormatter() // used to convert answer numbers
      from maps
     integerSec.timeZone = TimeZone(abbreviation: "UTC")
     integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
     decimalSec.timeZone = TimeZone(abbreviation: "UTC")
     decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    let noaaFmt = DateFormatter() // used to convert answer numbers
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"

    let latString = "40.334"
    let longString = "67.9632"
    let wheelString = "2020 Dec 14 T"

    var dmsloc = dms(40,20,2)
    XCTAssert( (Double(latString)! - dmsloc) < 0.0003)
    dmsloc = dms(67,57,47)
    XCTAssert( (Double(longString)! - dmsloc) < 0.0003)

        //           Summary View

        let correctNSText = "Location is within 6 nm (11 km) of centerline"
        let correctDescription = "Prediction for Total Eclipse of Dec 14,
         2020"

    let correctMag:Double = 1.013
    let correctDur:Double = 129.7
    let correctC1angle:Double = 297.7
    let semC2angle:Double = 90.0
    let semC3angle:Double = 270.0
    let correctC4angle:Double = 41.0
    let noaaSunriseDate = noaaFmt.date(from: "12/14/20 08:56")!
    let noaaSunsetDate  = noaaFmt.date(from: "12/14/20 23:58")!

    //          circs
    let correctC1Date = integerSec.date(from: "12/14/2020 14:49:23")!
    let correctC1Alt:Double = 63
    let correctC1Az:Double = 57.9
```

```
        let correctC2Date = integerSec.date(from: "12/14/2020  16:12:24")!
        let correctC2Alt:Double = 73
        let correctC2Az:Double = 11.2

        let correctMidDate = integerSec.date(from: "12/14/2020 16:13:29")!
        let correctMidAlt:Double = 73
        let correctMidAz:Double = 10.3

        let correctC3Date = integerSec.date(from: "12/14/2020 16:14:34")!
        let correctC3alt:Double = 73
        let correctC3Az:Double = 9.5

        let correctC4Date = integerSec.date(from: "12/14/2020 17:40:02")!
        let correctC4Alt:Double = 67
        let correctC4Az:Double = 312.4


         // Use XCTAssert and related functions to verify your tests produce
          the correct results.

        getState()

//        XCTAssert(!nasaMode)
        XCTAssert(pickerShown == wheelString)

        XCTAssert(degShowing)
        XCTAssert(stripZero(latValue) == latString)
        XCTAssert(stripZero(longValue) == longString)


         XCTAssert(nSwitchValue == false)
         XCTAssert(sSwitchValue == true)


        XCTAssert(eSwitchValue == false)
        XCTAssert(wSwitchValue == true)

        XCTAssert(predictEnabled)

        app.buttons["PredictBut"].tap()

        // OK we have a prediction
        app.buttons["Summary"].tap()

        //NSLog(app.debugDescription)
        let nstext = app.staticTexts["ns"]
        XCTAssert(nstext.exists)
         XCTAssert(nstext.label == correctNSText)

        let descript = app.staticTexts["description"]
```

```
XCTAssert(descript.exists)
XCTAssert(descript.label == correctDescription)



// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gEWNum)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gEWNum)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( diff < gVisualMeasurement)


// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
```

```
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
```

```
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <=  gEWMapNum)

///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <=  gEWMapNum)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
```

```
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <=  gEWMapNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3alt)
XCTAssert( diff <= gEWNum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff <=  gEWMapNum)

///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
XCTAssert(c4timeQ.exists)
let c4timeStr = c4timeQ.label
let c4time = decimalSec.date(from: c4timeStr)
XCTAssert(c4time != nil)
diff = abs(c4time!.timeIntervalSince(correctC4Date))
XCTAssert( diff <= gEWCalc)

let c4alt = app.staticTexts["eventAlt5"]
XCTAssert(c4alt.exists)
let c4altStr = c4alt.label
let c4AltAsNumber = Double(c4altStr)
XCTAssert(c4AltAsNumber != nil)
diff = abs( c4AltAsNumber! - correctC4Alt)
XCTAssert( diff <= gEWNum)

let c4az = app.staticTexts["eventAz5"]
XCTAssert(c4az.exists)
let c4azStr = c4az.label
```

```
        let c4AzAsNumber = Double(c4azStr)
        XCTAssert(c4AzAsNumber != nil)
        diff = abs( c4AzAsNumber! - correctC4Az)
        XCTAssert( diff <=  gEWMapNum)

        let credit = app.staticTexts["EW source"]
        XCTAssert(credit.exists)


        app.buttons["Map"].tap()
        //NSLog(app.debugDescription)
        XCTAssert(app.images["2020m12d14"].exists)
        XCTAssert(app.buttons
         ["http://eclipsewise.com/oh/ec2020.html#SE2020JDec14T"].exists)
    }
     func testUI_EWnorth2020_GE() throws {

        // Use XCTAssert and related functions to verify your tests produce
         the correct results.

        getState()

//        XCTAssert(!nasaMode)
        XCTAssert(pickerShown == "2020 Dec 14 T")

        XCTAssert(degShowing)

        latTextField!.tap()
        latTextField!.clearAndEnterText(text: "40.234") // .1 degree north of
         GE
        app.buttons["KeyDone"].tap()

        getState()

        XCTAssert(stripZero(latValue) == "40.234")
        XCTAssert(stripZero(longValue) == "67.9632")


         XCTAssert(nSwitchValue == false)
         XCTAssert(sSwitchValue == true)


        XCTAssert(eSwitchValue == false)
        XCTAssert(wSwitchValue == true)

        XCTAssert(predictEnabled)

        app.buttons["PredictBut"].tap()

        // OK we have a prediction
        app.buttons["Summary"].tap()
```

```
        //NSLog(app.debugDescription)
        let nstext = app.staticTexts["ns"]
        XCTAssert(nstext.exists)
        XCTAssert(nstext.label == "Location is north of the centerline")
    }
    func testUI_EWsouth2020_GE() throws {

        // Use XCTAssert and related functions to verify your tests produce the
         correct results.

        getState()

//      XCTAssert(!nasaMode)
        XCTAssert(pickerShown == "2020 Dec 14 T")

        XCTAssert(degShowing)

        latTextField!.tap()
        latTextField!.clearAndEnterText(text: "40.434") // .1 degree north of GE
        app.buttons["KeyDone"].tap()

        getState()

        XCTAssert(stripZero(latValue) == "40.434")
        XCTAssert(stripZero(longValue) == "67.9632")


         XCTAssert(nSwitchValue == false)
         XCTAssert(sSwitchValue == true)


        XCTAssert(eSwitchValue == false)
        XCTAssert(wSwitchValue == true)

        XCTAssert(predictEnabled)

        app.buttons["PredictBut"].tap()

        // OK we have a prediction
        app.buttons["Summary"].tap()

        //NSLog(app.debugDescription)
        let nstext = app.staticTexts["ns"]
        XCTAssert(nstext.exists)
        XCTAssert(nstext.label == "Location is south of the centerline")
    }
    // opposite side of the world from GE.  EWPredict verified by unit test
    func testUI_EW2020_noeclipse() throws {
        let decimalSec = DateFormatter() // used to convert numbers displayed
```

```
        let integerSec = DateFormatter() // used to convert answer numbers
         from maps
        integerSec.timeZone = TimeZone(abbreviation: "UTC")
        integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
        decimalSec.timeZone = TimeZone(abbreviation: "UTC")
        decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

       let noaaFmt = DateFormatter() // used to convert answer numbers
       noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
       noaaFmt.dateFormat = "MM/dd/yy HH:mm"

       ///////////////// Success criteria //////////////////////////

           //           Summary View

           let correctNSText = ""
           let correctDescription = "No Eclipse visible at this location"


       let correctsunriseDate = decimalSec.date(from: "12/14/20 02:44:06.7")!
       let correctsunsetDate = decimalSec.date(from: "12/14/20 12:01:44.5")!

       //          circs



       var diff : Double
       // Use XCTAssert and related functions to verify your tests produce
        the correct results.

       getState()

//        XCTAssert(!nasaMode)
       XCTAssert(pickerShown == "2020 Dec 14 T")

       XCTAssert(degShowing)
       XCTAssert(stripZero(latValue) == "40.334")
       XCTAssert(stripZero(longValue) == "67.9632")

        nSwitch!.tap()
        eSwitch!.tap()
        getState()

        XCTAssert(nSwitchValue == true)
        XCTAssert(sSwitchValue == false)


       XCTAssert(eSwitchValue == true)
       XCTAssert(wSwitchValue == false)

       XCTAssert(predictEnabled)
```

```
app.buttons["PredictBut"].tap()

// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
 XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)




// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
XCTAssert(magtext.label == "N/A")

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
XCTAssert(durStr == "N/A")

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label

XCTAssert( c1angleStr == "")

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label

XCTAssert( c2angleStr == "")

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label

XCTAssert( c3angleStr == "")


// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
```

```
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
XCTAssert( c4angleStr == "")


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(correctsunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(correctsunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
XCTAssert(c1timeStr == "Not Visible")

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
XCTAssert( c1altStr == "")

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
XCTAssert( c1azStr == "")

///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
XCTAssert(c2timeStr == "Not Visible")
```

```
let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
XCTAssert( c2altStr == "")

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
XCTAssert( c2azStr == "")


///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
XCTAssert(midtimeStr == "Not Visible")

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
XCTAssert( midaltStr == "")


let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
XCTAssert( midazStr == "")


///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
XCTAssert(c3timeStr == "Not Visible")

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
XCTAssert( c3altStr == "")


let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
XCTAssert( c3azStr == "")


///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
```

```
    XCTAssert(c4timeQ.exists)
    let c4timeStr = c4timeQ.label
     XCTAssert(c4timeStr == "Not Visible")

    let c4alt = app.staticTexts["eventAlt5"]
    XCTAssert(c4alt.exists)
    let c4altStr = c4alt.label
    XCTAssert( c4altStr == "")


    let c4az = app.staticTexts["eventAz5"]
    XCTAssert(c4az.exists)
    let c4azStr = c4az.label
    XCTAssert( c4azStr == "")



}

// This is a partial eclipse in Puerto Montt Chile
func testUI_EW2020partial() throws {
    var diff: Double
    let decimalSec = DateFormatter() // used to convert numbers displayed
     let integerSec = DateFormatter() // used to convert answer numbers
      from maps
     integerSec.timeZone = TimeZone(abbreviation: "UTC")
     integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
     decimalSec.timeZone = TimeZone(abbreviation: "UTC")
     decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    let noaaFmt = DateFormatter() // used to convert answer numbers
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"


    let latString = "41.4647"  //south
    let longString = "72.9299" // west
    let wheelString = "2020 Dec 14 T"

    var dmsloc = dms(41,27,54)
    XCTAssert( (Double(latString)! - dmsloc) < 0.0003)
    dmsloc = dms(72,55,47)
    XCTAssert( (Double(longString)! - dmsloc) < 0.0003)

        //           Summary View

         let correctNSText = "Location is south of the centerline"
        let correctDescription = "Partial Eclipse Visible at this location"

    let correctMag:Double = 0.943
   // let correctDur = 129.7
```

```
        let correctC1angle:Double = 300.0
        //let semC2angle = 90.0
        //let semC3angle = 270.0
        let correctC4angle:Double = 64.0
        let noaaSunriseDate = noaaFmt.date(from: "12/14/20 09:12")!
        let noaaSunsetDate  = noaaFmt.date(from: "12/15/20 00:21")!

        //          circs
        let correctC1Date = integerSec.date(from: "12/14/2020 14:43:26")!
        let correctC1Alt:Double = 59
        let correctC1Az:Double = 64.3

        //let correctC2Date = whenFormatter.date(from: "12/14/2020
         16:12:23.9")!
        //let correctC1Alt = 72.7
       // let correctC2Az = 11.2

        let correctMidDate = integerSec.date(from: "12/14/2020 16:04:40")!
        let correctMidAlt:Double = 70
        let correctMidAz:Double = 29.0

        //let correctC3Date = whenFormatter.date(from: "12/14/2020
         16:14:33.7")!
       // let correctC3alt = 72.7
       // let correctC3Az = 9.5

        let correctC4Date = integerSec.date(from: "12/14/2020 17:29:56")!
        let correctC4Alt:Double = 70
        let correctC4Az:Double = 330.3



        // Use XCTAssert and related functions to verify your tests produce
         the correct results.

        getState()

//        XCTAssert(!nasaMode)
        XCTAssert(pickerShown == wheelString)

        XCTAssert(degShowing)

        latTextField!.tap()
        latTextField!.clearAndEnterText(text: latString)
        longTextField!.tap()
        longTextField!.clearAndEnterText(text: longString)
        app.buttons["KeyDone"].tap()
        getState()
        XCTAssert(stripZero(latValue) == latString)
        XCTAssert(stripZero(longValue) == longString)
```

```
 XCTAssert(nSwitchValue == false)
 XCTAssert(sSwitchValue == true)


XCTAssert(eSwitchValue == false)
XCTAssert(wSwitchValue == true)

XCTAssert(predictEnabled)

app.buttons["PredictBut"].tap()

// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
XCTAssert(descript.label == correctDescription)



// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gEWNum)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
 XCTAssert(durStr == "N/A")


// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
```

```
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label

XCTAssert( c2angleStr == "")

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
XCTAssert( c3angleStr == "")


// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
```

```
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <=  gEWMapNum)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
 XCTAssert(c2timeStr == "Not Visible")

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
XCTAssert( c2altStr == "")

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
XCTAssert( c2azStr == "")

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
```

```
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <=  gEWMapNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
 XCTAssert(c3timeStr == "Not Visible")

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
XCTAssert( c3altStr == "")


let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
XCTAssert( c3azStr == "")


///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
XCTAssert(c4timeQ.exists)
let c4timeStr = c4timeQ.label
let c4time = decimalSec.date(from: c4timeStr)!
diff = abs(c4time.timeIntervalSince(correctC4Date))
XCTAssert( diff <= gEWCalc)

let c4alt = app.staticTexts["eventAlt5"]
XCTAssert(c4alt.exists)
let c4altStr = c4alt.label
let c4AltAsNumber = Double(c4altStr)
XCTAssert(c4AltAsNumber != nil)
diff = abs( c4AltAsNumber! - correctC4Alt)
XCTAssert( diff <= gEWNum)

let c4az = app.staticTexts["eventAz5"]
XCTAssert(c4az.exists)
let c4azStr = c4az.label
let c4AzAsNumber = Double(c4azStr)
XCTAssert(c4AzAsNumber != nil)
diff = abs( c4AzAsNumber! - correctC4Az)
XCTAssert( diff < gVisualMeasurement)
```

```
}
///////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////// 2021 Jun ///////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////////////////
func test2021_06_GE() throws {
    let decimalSec = DateFormatter() // used to convert numbers displayed
     decimalSec.timeZone = TimeZone(abbreviation: "UTC")
    decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    let noaaFmt = DateFormatter() // used to convert answer numbers from
     maps
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"


    var diff : Double

    ////////////////  Success criteria ////////////////////////

        //            Summary View

        let correctNSText = "Location is within 6 nm (11 km) of centerline"
        let correctDescription = "Prediction for Annular Eclipse of Jun
         10, 2021"
        let correctMap = "2021m06d10"
        let correctLink =
         "http://eclipsewise.com/solar/SEprime/2001-2100/SE2021Jun10Aprime
         .html"
        let credit = app.staticTexts["EW source"]


    // By default we are testing EW with lat N  long W at height 0.  Code
     below will
    // have to be changed if this is different

    let latString = "80.8235"  // north
    let longString = "66.7741" // west
    let wheelString = "2021 Jun 10 A"


    let correctC1angle: Double  = 90
    let semC2angle: Double  = 90
    let semC3angle : Double = 270
    let correctC4angle: Double  = 275


    //NSLog(integerSec.string(from:engine.sunriseDate))
    //NSLog(integerSec.string(from:engine.sunsetDate))
```

```
    //https://www.esrl.noaa.gov/gmd/grad/solcalc/sunrise.html

    // sun does not set
    //let noaaSunriseDate = noaaFmt.date(from: "xxxxxxx")!
    //let noaaSunsetDate  = noaaFmt.date(from: "xxxxxxx")!


    // test magnitude
    let correctMag: Double  = 0.972


    // Test C1
    // Test C1
    let correctC1Date = decimalSec.date(from: "06/10/21 09:36:24.6")!
    let correctC1Alt :Double = 20.8
    let correctC1Az :Double = 73.9

    ///////////////////////////////// C2 ////////////////////////////////////////////////////////////////////////////////////////////

    let correctC2Date = decimalSec.date(from: "06/10/21 10:40:01.9")!
    let correctC2Alt :Double = 23.3
    let correctC2Az :Double = 89.4
    ///////////////////////////////// Mid ////////////////////////////////////////////////////////////////////////////////////////////

    let correctMidDate = decimalSec.date(from: "06/10/21 10:41:57.4")!
    let correctMidAlt :Double = 23.3
    let correctMidAz :Double = 89.9

    ///////////////////////////////// C3 ////////////////////////////////////////////////////////////////////////////////////////////

    let correctC3Date = decimalSec.date(from: "06/10/21 10:43:52.9")!
    let correctC3Alt :Double = 23.4
    let correctC3Az :Double = 90.4

    ///////////////////////////////// C4 ////////////////////////////////////////////////////////////////////////////////////////////

    let correctC4Date = decimalSec.date(from: "06/10/21 11:48:58.8")!
    let correctC4Alt :Double = 26.0
    let correctC4Az :Double = 106.4

     // Use XCTAssert and related functions to verify your tests produce
      the correct results.
    let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

    getState()

//    XCTAssert(!nasaMode)
    picker!.adjust(toPickerWheelValue: wheelString)
    getState()
    XCTAssert(pickerShown == wheelString)
```

```
XCTAssert(degShowing)

 XCTAssert(nSwitchValue == true)
 XCTAssert(sSwitchValue == false)

XCTAssert(eSwitchValue == false)
XCTAssert(wSwitchValue == true)

heightTextField!.tap()
heightTextField!.clearAndEnterText(text: "0")

app.buttons["KeyDone"].tap()
getState()
XCTAssert(stripZero(latValue) ==  latString)
XCTAssert(stripZero(longValue) == longString)


app.buttons["PredictBut"].tap()

// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
 XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)




// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff <= gNumbers)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff <= gNumbers)
```

```
// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
XCTAssert(sunriseStr == "Sun above Horizon")


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
XCTAssert(sunsetStr == "Sun above Horizon")



// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
```

```
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff <= gEWCalc)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff <= gEWNum)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff <=  gEWMapNum)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff <= gEWCalc)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff <= gEWNum)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff <= gEWMapNum)

///////////////// mid ////////////////////////////////////
```

```
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff <= gEWCalc)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff <= gEWNum)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff <= gEWMapNum)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff <= gEWCalc)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff <= gEWNum)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff <= gEWMapNum)

///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
```

```
    XCTAssert(c4timeQ.exists)
    let c4timeStr = c4timeQ.label
    let c4time = decimalSec.date(from: c4timeStr)
    XCTAssert(c4time != nil)
    diff = abs(c4time!.timeIntervalSince(correctC4Date))
    XCTAssert( diff <= gEWCalc)

    let c4alt = app.staticTexts["eventAlt5"]
    XCTAssert(c4alt.exists)
    let c4altStr = c4alt.label
    let c4AltAsNumber = Double(c4altStr)
    XCTAssert(c4AltAsNumber != nil)
    diff = abs( c4AltAsNumber! - correctC4Alt)
    XCTAssert( diff <= gEWNum)

    let c4az = app.staticTexts["eventAz5"]
    XCTAssert(c4az.exists)
    let c4azStr = c4az.label
    let c4AzAsNumber = Double(c4azStr)
    XCTAssert(c4AzAsNumber != nil)
    diff = abs( c4AzAsNumber! - correctC4Az)
    XCTAssert( diff <= gEWMapNum)


    XCTAssert(credit.exists)


    app.buttons["Map"].tap()
    //NSLog(app.debugDescription)
    XCTAssert(app.images[correctMap].exists)
    XCTAssert(app.buttons[correctLink].exists)
}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////// 2021 Dec ///////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////////////////////////////////

func testUI_EW2021_GE() throws {
    var diff: Double
    let decimalSec = DateFormatter() // used to convert numbers displayed
     let integerSec = DateFormatter() // used to convert answer numbers
      from maps
     integerSec.timeZone = TimeZone(abbreviation: "UTC")
     integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
     decimalSec.timeZone = TimeZone(abbreviation: "UTC")
     decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    /////////////// Success criteria ////////////////////////

        //            Summary View
```

```
    let correctNSText = "Location is within 6 nm (11 km) of centerline"
    let correctDescription = "Prediction for Total Eclipse of Dec 4,
     2021"

// By default we are testing EW with lat N  long W at height 0.  Code
 below will
// have to be changed if this is different

let latString = "76.7762"
let longString = "46.2308"
let wheelString = "2021 Dec 04 T"



  let correctC1angle :Double = 270.0
  //let semC2angle :Double = 125.0
  //let semC3angle :Double = 240.0
  let correctC4angle :Double = 90.0


//NSLog(whenFormatter.string(from:engine.sunriseDate))
//NSLog(whenFormatter.string(from:engine.sunsetDate))

// below antartic circle
let correctsunriseDateStr = "Sun above Horizon"
let correctsunsetDateStr  = "Sun above Horizon"


// test magnitude
  let correctMag :Double  = 1.018


// Test C1
let correctC1Date = decimalSec.date(from: "12/04/21 06:41:03.5")!
  let correctC1Alt :Double  = 14.6
  let correctC1Az :Double  = 127.1

let correctC2Date = decimalSec.date(from: "12/04/21 07:32:30.1")!
  let correctC2Alt :Double  = 17.1
  let correctC2Az :Double  = 115.0

let correctMidDate = decimalSec.date(from: "12/04/21 07:33:27.3")!
  let correctMidAlt :Double  = 17.2
  let correctMidAz :Double  = 114.8

let correctC3Date = decimalSec.date(from: "12/04/21 07:34:24.6")!
  let correctC3Alt :Double  = 17.2
  let correctC3Az :Double  = 114.6

let correctC4Date = decimalSec.date(from: "12/04/21 08:26:51.4")!
  let correctC4Alt :Double  = 20.1
```

```
        let correctC4Az :Double  = 102.1



        // Use XCTAssert and related functions to verify your tests produce
         the correct results.
       let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

       getState()

//       XCTAssert(!nasaMode)
       picker!.adjust(toPickerWheelValue: wheelString)
       getState()
       XCTAssert(pickerShown == wheelString)

       XCTAssert(degShowing)

        XCTAssert(nSwitchValue == false)
        XCTAssert(sSwitchValue == true)

       XCTAssert(eSwitchValue == false)
       XCTAssert(wSwitchValue == true)

       latTextField!.tap()
       latTextField!.clearAndEnterText(text: latString)
       longTextField!.tap()
       longTextField!.clearAndEnterText(text: longString)
       heightTextField!.tap()
       heightTextField!.clearAndEnterText(text: "0")

       app.buttons["KeyDone"].tap()
       getState()
       XCTAssert(stripZero(latValue) ==  latString)
       XCTAssert(stripZero(longValue) == longString)


       app.buttons["PredictBut"].tap()

       // OK we have a prediction
       app.buttons["Summary"].tap()

       //NSLog(app.debugDescription)
       let nstext = app.staticTexts["ns"]
       XCTAssert(nstext.exists)
        XCTAssert(nstext.label == correctNSText)

       let descript = app.staticTexts["description"]
       XCTAssert(descript.exists)
        XCTAssert(descript.label == correctDescription)
```

```
// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff < gNumbers)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff < gNumbers)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)

// Sky X predicts too far off of Eclipsewise to be believed
#if DONTINCLUDE
let c2angle = app.staticTexts["C2 angle"]
XCTAssert(c2angle.exists)
let c2angleStr = c2angle.label
let c2angleAsNumber = Double(c2angleStr)
XCTAssert(c2angleAsNumber != nil)
let c2diff = abs( c2angleAsNumber! - semC2angle)
XCTAssert( c2diff < gVisualMeasurement)

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)
#endif

// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
```

```
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)

// sun never sets
let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
XCTAssert(sunriseStr == correctsunriseDateStr)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
XCTAssert(sunsetStr == correctsunsetDateStr)




// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff < gpassMap)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff < gVisualMeasurement)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff < gVisualMeasurement)
///////////////// c2 ////////////////////////////////////
```

```
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff < gpassMap)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff < gVisualMeasurement)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff < gVisualMeasurement)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff < gpassMap)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff < gVisualMeasurement)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff < gVisualMeasurement)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
```

```
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff < gpassMap)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff < gVisualMeasurement)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
XCTAssert( diff < gVisualMeasurement)

///////////////// c4 ////////////////////////////////////
let c4timeQ = app.staticTexts["eventTime5"]
XCTAssert(c4timeQ.exists)
let c4timeStr = c4timeQ.label
let c4time = decimalSec.date(from: c4timeStr)
XCTAssert(c4time != nil)
diff = abs(c4time!.timeIntervalSince(correctC4Date))
XCTAssert( diff < gpassMap)

let c4alt = app.staticTexts["eventAlt5"]
XCTAssert(c4alt.exists)
let c4altStr = c4alt.label
let c4AltAsNumber = Double(c4altStr)
XCTAssert(c4AltAsNumber != nil)
diff = abs( c4AltAsNumber! - correctC4Alt)
XCTAssert( diff < gVisualMeasurement)

let c4az = app.staticTexts["eventAz5"]
XCTAssert(c4az.exists)
let c4azStr = c4az.label
let c4AzAsNumber = Double(c4azStr)
XCTAssert(c4AzAsNumber != nil)
diff = abs( c4AzAsNumber! - correctC4Az)
XCTAssert( diff < gVisualMeasurement)

let credit = app.staticTexts["EW source"]
XCTAssert(credit.exists)
```

```
    app.buttons["Map"].tap()
    //NSLog(app.debugDescription)
    XCTAssert(app.images["2021m12d04"].exists)
    XCTAssert(app.buttons
     ["http://eclipsewise.com/solar/SEprime/2001-2100/SE2021Dec04Tprime
     .html"].exists)
}
func testUI_EW2021_WilderenessT() throws {
    let decimalSec = DateFormatter() // used to convert numbers displayed
     let integerSec = DateFormatter() // used to convert answer numbers
      from maps
     integerSec.timeZone = TimeZone(abbreviation: "UTC")
     integerSec.dateFormat = "MM/dd/yy HH:mm:ss"
     decimalSec.timeZone = TimeZone(abbreviation: "UTC")
     decimalSec.dateFormat = "MM/dd/yy HH:mm:ss.S"

    let noaaFmt = DateFormatter() // used to convert answer numbers
    noaaFmt.timeZone = TimeZone(abbreviation: "UTC")
    noaaFmt.dateFormat = "MM/dd/yy HH:mm"

    ///////////////// Success criteria //////////////////////////

        //           Summary View

        let correctNSText = "Location is within 6 nm (11 km) of centerline"
        let correctDescription = "Prediction for Total Eclipse of Dec 4,
         2021"

    // By default we are testing EW with lat N  long W at height 0.  Code
     below will
    // have to be changed if this is different

    let latString = "60.2388"  //south
    let longString = "41.3309" //west
    let wheelString = "2021 Dec 04 T"




    let correctC1angle: Double  = 280.0
    //let semC2angle: Double  = 120.0
    let semC3angle : Double = 279.0 // using generated instead of measured
    let correctC4angle: Double  = 105.0


  //NSLog(whenFormatter.string(from:engine.sunriseDate))
  //NSLog(whenFormatter.string(from:engine.sunsetDate))
    //https://www.esrl.noaa.gov/gmd/grad/solcalc/sunrise.html
    let noaaSunriseDate = noaaFmt.date(from: "12/04/21 05:22")!
    let noaaSunsetDate  = noaaFmt.date(from: "12/04/21 23:51")!
```

```
        // test magnitude
          let correctMag: Double  = 1.017


        // Test C1
        let correctC1Date = decimalSec.date(from: "12/04/21 06:18:05.9    ")!
          let correctC1Alt : Double = 4.0
          let correctC1Az : Double = 130.0

        let correctC2Date = decimalSec.date(from: "12/04/21 07:06:36.2")!
          let correctC2Alt: Double  = 8.9
          let correctC2Az : Double = 119.9

        let correctMidDate = decimalSec.date(from: "12/04/21 07:07:26.7")!
          let correctMidAlt : Double = 9.0
          let correctMidAz : Double = 119.7

        let correctC3Date = decimalSec.date(from: "12/04/21 07:08:17.2")!
          let correctC3Alt : Double = 9.1
          let correctC3Az : Double = 119.5

        let correctC4Date = decimalSec.date(from: "12/04/21 07:58:36.4")!
          let correctC4Alt : Double = 14.8
          let correctC4Az : Double = 109.1
          var diff : Double
          // Use XCTAssert and related functions to verify your tests produce
           the correct results.
          let correctDur = correctC3Date.timeIntervalSince(correctC2Date)

          getState()

//          XCTAssert(!nasaMode)
          picker!.adjust(toPickerWheelValue: wheelString)
          getState()
          XCTAssert(pickerShown == wheelString)

          XCTAssert(degShowing)

           XCTAssert(nSwitchValue == false)
           XCTAssert(sSwitchValue == true)

          XCTAssert(eSwitchValue == false)
          XCTAssert(wSwitchValue == true)

          latTextField!.tap()
          latTextField!.clearAndEnterText(text: latString)
          longTextField!.tap()
          longTextField!.clearAndEnterText(text: longString)
          heightTextField!.tap()
```

```
heightTextField!.clearAndEnterText(text: "0")

app.buttons["KeyDone"].tap()
getState()
XCTAssert(stripZero(latValue) ==  latString)
XCTAssert(stripZero(longValue) == longString)


app.buttons["PredictBut"].tap()

// OK we have a prediction
app.buttons["Summary"].tap()

//NSLog(app.debugDescription)
let nstext = app.staticTexts["ns"]
XCTAssert(nstext.exists)
 XCTAssert(nstext.label == correctNSText)

let descript = app.staticTexts["description"]
XCTAssert(descript.exists)
 XCTAssert(descript.label == correctDescription)




// we have to search by value instead by field
let magtext = app.staticTexts["magnitudeValue"]
XCTAssert(magtext.exists)
let magStr = magtext.label
let magAsNumber = Double(magStr)
XCTAssert(magAsNumber != nil)
diff = abs( magAsNumber! - correctMag)
XCTAssert( diff < gNumbers)

let durtext = app.staticTexts["durationValue"]
XCTAssert(durtext.exists)
let durStr = durtext.label
let durAsNumber = Double(durStr)
XCTAssert(durAsNumber != nil)
diff = abs( durAsNumber! - correctDur)
XCTAssert( diff < gNumbers)

// use displayed value - correct value tested in unit test
let c1angle = app.staticTexts["C1 angle"]
XCTAssert(c1angle.exists)
let c1angleStr = c1angle.label
let c1angleAsNumber = Double(c1angleStr)
XCTAssert(c1angleAsNumber != nil)
diff = abs( c1angleAsNumber! - correctC1angle)
XCTAssert( diff < gVisualMeasurement)
```

```
// C2 not tested since SkyX prediction not reliable
#if SKYNOTRELIABLE
    let c2angle = app.staticTexts["C2 angle"]
    XCTAssert(c2angle.exists)
    let c2angleStr = c2angle.label
    let c2angleAsNumber = Double(c2angleStr)
    XCTAssert(c2angleAsNumber != nil)
    let c2diff = abs( c2angleAsNumber! - semC2angle)
    XCTAssert( c2diff < gVisualMeasurement)
#endif

let c3angle = app.staticTexts["C3 angle"]
XCTAssert(c3angle.exists)
let c3angleStr = c3angle.label
let c3angleAsNumber = Double(c3angleStr)
XCTAssert(c3angleAsNumber != nil)
let c3diff = abs( c3angleAsNumber! - semC3angle)
XCTAssert( c3diff < gVisualMeasurement)


// use displayed value - correct value tested in unit test

let c4angle = app.staticTexts["C4 angle"]
XCTAssert(c4angle.exists)
let c4angleStr = c4angle.label
let c4angleAsNumber = Double(c4angleStr)
XCTAssert(c4angleAsNumber != nil)
let c4diff = abs( c4angleAsNumber! - correctC4angle)
XCTAssert( c4diff < gVisualMeasurement)


let sunrise = app.staticTexts["sunriseValue"]
XCTAssert(sunrise.exists)
let sunriseStr = sunrise.label
let sunrisetime = noaaFmt.date(from: sunriseStr)
XCTAssert(sunrisetime != nil)
 diff = abs(noaaSunriseDate.timeIntervalSince(sunrisetime!))
XCTAssert( diff < gPassSunsetRise)


let sunset = app.staticTexts["sunsetValue"]
XCTAssert(sunset.exists)
let sunsetStr = sunset.label
let sunsettime = noaaFmt.date(from: sunsetStr)
XCTAssert(sunsettime != nil)

diff = abs(noaaSunsetDate.timeIntervalSince(sunsettime!))
XCTAssert( diff < gPassSunsetRise)
```

```
// OK we have a prediction
app.buttons["Times"].tap()

//let foo = app.debugDescription
//NSLog(foo)
// we are now on the Circumstances screen
let c1timeQ = app.staticTexts["eventTime1"]
XCTAssert(c1timeQ.exists)
let c1timeStr = c1timeQ.label
let c1time = decimalSec.date(from: c1timeStr)
XCTAssert(c1time != nil)
 diff = abs(c1time!.timeIntervalSince(correctC1Date))
XCTAssert( diff < gpassMap)

let c1alt = app.staticTexts["eventAlt1"]
XCTAssert(c1alt.exists)
let c1altStr = c1alt.label
let c1AltAsNumber = Double(c1altStr)
XCTAssert(c1AltAsNumber != nil)
diff = abs( c1AltAsNumber! - correctC1Alt)
XCTAssert( diff < gVisualMeasurement)

let c1az = app.staticTexts["eventAz1"]
XCTAssert(c1az.exists)
let c1azStr = c1az.label
let c1AzAsNumber = Double(c1azStr)
XCTAssert(c1AzAsNumber != nil)
diff = abs( c1AzAsNumber! - correctC1Az)
XCTAssert( diff < gVisualMeasurement)
///////////////// c2 ////////////////////////////////////
let c2timeQ = app.staticTexts["eventTime2"]
XCTAssert(c2timeQ.exists)
let c2timeStr = c2timeQ.label
let c2time = decimalSec.date(from: c2timeStr)
XCTAssert(c2time != nil)
diff = abs(c2time!.timeIntervalSince(correctC2Date))
XCTAssert( diff < gpassMap)

let c2alt = app.staticTexts["eventAlt2"]
XCTAssert(c2alt.exists)
let c2altStr = c2alt.label
let c2AltAsNumber = Double(c2altStr)
XCTAssert(c2AltAsNumber != nil)
diff = abs( c2AltAsNumber! - correctC2Alt)
XCTAssert( diff < gVisualMeasurement)

let c2az = app.staticTexts["eventAz2"]
XCTAssert(c2az.exists)
let c2azStr = c2az.label
let c2AzAsNumber = Double(c2azStr)
XCTAssert(c2AzAsNumber != nil)
```

```
diff = abs( c2AzAsNumber! - correctC2Az)
XCTAssert( diff < gVisualMeasurement)

///////////////// mid ////////////////////////////////////
let midtimeQ = app.staticTexts["eventTime3"]
XCTAssert(midtimeQ.exists)
let midtimeStr = midtimeQ.label
let midtime = decimalSec.date(from: midtimeStr)
XCTAssert(midtime != nil)
diff = abs(midtime!.timeIntervalSince(correctMidDate))
XCTAssert( diff < gpassMap)

let midalt = app.staticTexts["eventAlt3"]
XCTAssert(midalt.exists)
let midaltStr = midalt.label
let midAltAsNumber = Double(midaltStr)
XCTAssert(midAltAsNumber != nil)
diff = abs( midAltAsNumber! - correctMidAlt)
XCTAssert( diff < gVisualMeasurement)

let midaz = app.staticTexts["eventAz3"]
XCTAssert(midaz.exists)
let midazStr = midaz.label
let midAzAsNumber = Double(midazStr)
XCTAssert(midAzAsNumber != nil)
diff = abs( midAzAsNumber! - correctMidAz)
XCTAssert( diff < gVisualMeasurement)

///////////////// c3 ////////////////////////////////////
let c3timeQ = app.staticTexts["eventTime4"]
XCTAssert(c3timeQ.exists)
let c3timeStr = c3timeQ.label
let c3time = decimalSec.date(from: c3timeStr)
XCTAssert(c3time != nil)
diff = abs(c3time!.timeIntervalSince(correctC3Date))
XCTAssert( diff < gpassMap)

let c3alt = app.staticTexts["eventAlt4"]
XCTAssert(c3alt.exists)
let c3altStr = c3alt.label
let c3AltAsNumber = Double(c3altStr)
XCTAssert(c3AltAsNumber != nil)
diff = abs( c3AltAsNumber! - correctC3Alt)
XCTAssert( diff < gVisualMeasurement)

let c3az = app.staticTexts["eventAz4"]
XCTAssert(c3az.exists)
let c3azStr = c3az.label
let c3AzAsNumber = Double(c3azStr)
XCTAssert(c3AzAsNumber != nil)
diff = abs( c3AzAsNumber! - correctC3Az)
```

```
    XCTAssert( diff < gVisualMeasurement)

    ///////////////// c4 ////////////////////////////////////
    let c4timeQ = app.staticTexts["eventTime5"]
    XCTAssert(c4timeQ.exists)
    let c4timeStr = c4timeQ.label
    let c4time = decimalSec.date(from: c4timeStr)
    XCTAssert(c4time != nil)
    diff = abs(c4time!.timeIntervalSince(correctC4Date))
    XCTAssert( diff < gpassMap)

    let c4alt = app.staticTexts["eventAlt5"]
    XCTAssert(c4alt.exists)
    let c4altStr = c4alt.label
    let c4AltAsNumber = Double(c4altStr)
    XCTAssert(c4AltAsNumber != nil)
    diff = abs( c4AltAsNumber! - correctC4Alt)
    XCTAssert( diff < gVisualMeasurement)

    let c4az = app.staticTexts["eventAz5"]
    XCTAssert(c4az.exists)
    let c4azStr = c4az.label
    let c4AzAsNumber = Double(c4azStr)
    XCTAssert(c4AzAsNumber != nil)
    diff = abs( c4AzAsNumber! - correctC4Az)
    XCTAssert( diff < gVisualMeasurement)

    let credit = app.staticTexts["EW source"]
    XCTAssert(credit.exists)


    app.buttons["Map"].tap()
    //NSLog(app.debugDescription)
    XCTAssert(app.images["2021m12d04"].exists)
    XCTAssert(app.buttons
     ["http://eclipsewise.com/solar/SEprime/2001-2100/SE2021Dec04Tprime
     .html"].exists)
}


func stripZero(_ input: String) ->String{
    var work = input
    while true{
        let lastChar = work.removeLast()
        if lastChar == "0" {
            continue
        }
        if work.count == 0  && lastChar == "0" {
            // only contained 0
            return "0"
        }
```

```
            // stripped one to many put it back and exit
            work.append(lastChar)
            return work
        }
    }
    // convert DMS into degress
    func dmsStr (_ deg: Double,_ min: Double, _ sec: Double) -> String{
        var sign = 1.0
        var workdeg = deg

        if deg < 0{
            sign = -1.0
            workdeg = abs(workdeg)
        }

        let value = sign * (workdeg + min/60 + sec/3600)

        return String(format:"%1.5f", value)
    }

}
```