```
//////////////////////////////////////////////////////////////////////////////
// Swift source code of UserInterface.swift Version 9.1.0 at greenchess.com //
//////////////////////////////////////////////////////////////////////////////

// import needed system files //
import SwiftUI
import AppKit

// the screen view information //
class ScreenInfo
{
   // the x position of the screen view //
   var XP:Double = 0

   // the y position of the screen view //
   var YP:Double = 0

   // the x origin of the view //
   var OX:Double = 0

   // the y origin of the view //
   var OY:Double = 0

   // the width of the screen view //
   var Width:Double = 0

   // the height of the screen view //
   var Height:Double = 0

   // the list of items //
   var Items:[String] = []

   // the dictionary key //
   var Key:String = ""

   // the control label to display //
   var Label:String = ""

   // the placeholder text //
   var Placeholder:String = ""

   // the subwindow //
   var SubWindow:String = ""

   // the window to display the view on //
   var Window:NSWindow!

   // whether the item is selectable or not //
   var Selectable:Bool = false

   // the toolbar buttons //
   var BarButtons:[(String, Color, String, String)] = []

   // the slider function for a view //
   var SliderFunc:(String, Double) -> String = {(Key:String, Value:Double) in
                                               return ""}

   // the toolbar function for a view //
   var ToolbarFunc:(Int) -> () = {(Selection:Int) in ()}

   // the toggle function for a view //
   var ToggleFunc:(String) -> () = {(Key:String) in ()}

   // the date picker function for a view //
   var DateFunc:(String, Date) -> () = {(Key:String, GameDate:Date) in ()}

   // the picker function for a view //
   var PickerFunc:(String, Int) -> () = {(Key:String, Selection:Int) in ()}

   // the refresh list or load game function for a view //
   var RefreshOrLoadFunc:(String, String, Bool) -> () = {(Key:String, Search:
                                                        String, Load:Bool) in
                                                        ()}

   // the entered text field function //
   var EnteredFunc:(String, String) -> () = {(Key:String, Entered:String) in ()}

   // a screen view initializer for a toggle button control view //
   init(_ Label:String, Key:String, SubWindow:String, XP:Double, YP:Double,
       Width:Double, Height:Double, Window:NSWindow, ToggleFunc:
       @escaping(String) -> ())
   {
      // initialize the screen view values for a toggle button control view //
      (self.Label, self.Key, self.SubWindow, self.XP, self.YP, self.Width, self
      .Height, self.Window, self.ToggleFunc) = (Label, Key, SubWindow, XP, YP,
      Width, Height, Window, ToggleFunc)
   }

   // a screen view initializer for a slider control view //
   init(_ Key:String, SubWindow:String, XP:Double, YP:Double, Width:Double,
       Height:Double, Window:NSWindow, SliderFunc:@escaping(String, Double) ->
       String)
   {
      // initialize the screen view values for a slider control view //
      (self.Key, self.SubWindow, self.XP, self.YP, self.Width, self.Height,
      self.Window, self.SliderFunc) = (Key, SubWindow, XP, YP, Width, Height,
      Window, SliderFunc)
   }

   // a screen view initializer for an image control view //
   init(_ SubWindow:String, XP:Double, YP:Double, Width:Double, Height:Double,
       Window:NSWindow)
   {
      // initialize the screen view values for an image control view //
      (self.SubWindow, self.XP, self.YP, self.Width, self.Height, self.Window)
      = (SubWindow, XP, YP, Width, Height, Window)
   }

   // a screen view initializer for a text control view //
   init(_ Label:String, SubWindow:String, XP:Double, YP:Double, Width:Double,
       Height:Double, Window:NSWindow)
   {
      // initialize the screen view values for a text control view //
      (self.Label, self.SubWindow, self.XP, self.YP, self.Width, self.Height,
      self.Window) = (Label, SubWindow, XP, YP, Width, Height, Window)
   }

   // a screen view initializer for a color control view //
   init(_ SubWindow:String, OX:Double, OY:Double, Window:NSWindow)
   {
      // initialize the screen view values for a color control view //
      (self.SubWindow, self.OX, self.OY, self.Window) = (SubWindow, OX, OY,
      Window)
   }
```

```
   // a screen view initializer for an entered text box control view //
   init(_ Label:String, Placeholder:String, Key:String, XP:Double, YP:Double,
       Width:Double, Height:Double, Window:NSWindow, EnteredFunc:
       @escaping(String, String) -> ())
   {
      // initialize screen view values for an entered text box control view //
      (self.Label, self.Placeholder, self.Key, self.XP, self.YP, self.Width,
      self.Height, self.Window, self.EnteredFunc) = (Label, Placeholder, Key,
      XP, YP, Width, Height, Window, EnteredFunc)
   }

   // screen view initializer for load selected row in list database view //
   init(_ Key:String, Window:NSWindow, RefreshOrLoadFunc:@escaping(String,
       String, Bool) -> ())
   {
      // initialize screen view values for load selected row in list view //
      (self.Key, self.Window, self.RefreshOrLoadFunc) = (Key, Window,
      RefreshOrLoadFunc)
   }

   // a screen view initializer for date picker control view //
   init(_ Label:String, Key:String, XP:Double, YP:Double, Width:Double, Height:
       Double, Window:NSWindow, DateFunc:@escaping(String, Date) -> ())
   {
      // initialize the screen view values for a date picker control view //
      (self.Label, self.Key, self.XP, self.YP, self.Width, self.Height, self
      .Window, self.DateFunc) = (Label, Key, XP, YP, Width, Height, Window,
      DateFunc)
   }

   // a screen view initializer for a picker control view //
   init(_ Key:String, SubWindow:String, XP:Double, YP:Double, Width:Double,
       Height:Double, Window:NSWindow, Items:[String], PickerFunc:
       @escaping(String, Int) -> ())
   {
      // initialize the screen view values for a picker control view //
      (self.Key, self.SubWindow, self.XP, self.YP, self.Width, self.Height,
      self.Window, self.Items, self.PickerFunc) = (Key, SubWindow, XP, YP,
      Width, Height, Window, Items, PickerFunc)
   }

   // a screen view initializer for a toolbar control view //
   init(_ Window:NSWindow, Selectable:Bool, BarButtons:[(String, Color, String,
       String)], ToolbarFunc:@escaping(Int) -> ())
   {
      // initialize the screen view values for a toolbar control view //
      (self.Window, self.Selectable, self.BarButtons, self.ToolbarFunc) =
      (Window, Selectable, BarButtons, ToolbarFunc)
   }

   // the default initializer //
   init() {}
}

// the toolbar button style //
struct CustomButton:ButtonStyle
{
   // create the button style //
   func makeBody(configuration:Configuration) -> some View {configuration.label}
}

// the toolbar view structure //
struct ToolbarStruct:View
{
   // the observed screen view object //
   @ObservedObject var ScreenObj:Obj = Obj()

   // the observed screen view initializer //
   init(_ ScreenObj:Obj) {self.ScreenObj = ScreenObj}

   // the main body //
   var body:some View {VStack(alignment:.leading, spacing:0)
   {
      // the hstack //
      HStack()
      {
         // loop over each toolbar item //
         ForEach(0..<ScreenObj.Inf.BarButtons.count, id:\.self)
         {
            // the looping variable //
            Index in

            // the hstack for each toolbar button //
            HStack()
            {
               // the toolbar action button //
               Button(action:{if (Index > 0) && ScreenObj.Inf
                      .Selectable {ScreenObj.Selection = Index}; ScreenObj.Inf
                      .ToolbarFunc(Index)})
               {
                  // the toolbar button label //
                  Label("", systemImage:ScreenObj.Inf.BarButtons[Index].0)

                  // set the font size //
                  .font(.system(size:15))

                  // set the image scale to large //
                  .imageScale(.large)

                  // set the view frame //
                  .frame(width:(Index == 0) ? 20 : 40, height:(Index == 0) ? 20
                        : 35)

                  // set the content shape //
                  .contentShape(Rectangle())
               }

               // add help tooltip //
               .help(ScreenObj.Inf.BarButtons[Index].2)

               // if hoving over the toolbar button, save the hovered index //
               .onHover {Hovering in ScreenObj.Hovered = Hovering ? Index : -1}

               // set foreground color for the close button or other button //
               .foregroundColor(((ScreenObj.Hovered == 0) && (Index == 0)) ?
                               Color(red:0.4, green:0.5, blue:0.6) : ScreenObj
                               .Inf.BarButtons[Index].1)

               // set the background color for the toolbar button //
               .background(.gray.opacity(([ScreenObj.Selection, ScreenObj
                          .Hovered].contains(Index) && (Index > 0)) ? 0.2 : 0))

               // set the custom button style //
               .buttonStyle(CustomButton())
            }

            // add the leading padding in the hstack //
            .padding([.leading], (Index == 0) ? 10 : 0)

            // add the trailing padding in the stack //
            .padding([.trailing], (Index == 0) ? 30 : 5)
         }
      }
```

```
      // add the padding to the top and bottom of the hstack //
      .padding([.bottom, .top], (ScreenObj.Inf.BarButtons.count == 1) ? 7.5 : 0)

      // refresh the hovered toolbar button index when need to refresh //
      .onChange(of:ScreenObj.Refresh) {_ in ScreenObj.Hovered = -1}

      // the horizontal divider //
      Divider().overlay(.black.opacity(0.25))
   }}
}

// the game list view structure //
struct GameListStruct:View
{
   // the observed screen view object //
   @ObservedObject var ScreenObj:Obj = Obj()

   // the key input //
   @State var KeyInput:Any?

   // whether currently in focus //
   @FocusState private var InFocus:Bool

   // the observed screen view initializer //
   init(_ ScreenObj:Obj) {self.ScreenObj = ScreenObj}

   // the main body //
   var body:some View
   {
      // the vstack //
      LazyVStack(alignment:.leading, spacing:0)
      {
         // the hstack //
         HStack()
         {
            // display the current game count in the database //
            Text("Games:  \(ScreenObj.GameList.count) ")

            // set the system font //
            .font(.system(size:15))

            // set the text frame size and alignment //
            .frame(width:660, height:30, alignment:.center)
         }

         // add padding to the bottom of the game count text //
         .padding([.bottom], 6)

         // the hstack //
         HStack()
         {
            // the magnifying glass button to start a search in search box //
            Button(action:{InFocus = true})
            {
               // the magnifying glass icon in the search bar //
               Image(systemName:"magnifyingglass")

               // set the image color to gray //
               .foregroundColor(.gray)
            }

            // set the custom button style //
            .buttonStyle(CustomButton())

            // set the button frame size //
            .frame(width:25, height:25, alignment:.center)

            // add padding after the search icon image //
            .padding([.leading], 5)

            // the text field search box //
            TextField("", text:Binding(get:{ScreenObj.Entered}, set:
                     {Entered in ScreenObj.Entered = Entered}))

            // set the text field style to plain //
            .textFieldStyle(.plain)

            // set the search box frame size //
            .frame(width:nil, height:30, alignment:.center)

            // update to the current focus state //
            .focused($InFocus)

            // if the focus changed //
            .onChange(of:InFocus)
            {
               // for variable in //
               _ in

               // if in focus, cause the text field to update by changing it //
               if InFocus {_ = ScreenObj.Entered.popLast()}

               // if not in focus, cause text field to update by changing it //
               if !InFocus {ScreenObj.Entered += " "}
            }

            // if the entered search string changes //
            .onChange(of:ScreenObj.Entered)
            {
               // for variable in //
               _ in

               // refresh the list view based on the new search text //
               ScreenObj.Inf.RefreshOrLoadFunc(ScreenObj.Inf.Key, ScreenObj
                                              .Entered, false)
            }

            // if the entered search string is not empty //
            if !GetString().FromTextField(ScreenObj.Entered).isEmpty
            {
               // add the button to clear the search string //
               Button(action:{ScreenObj.Entered = ""; InFocus = true; ScreenObj
                     .Inf.RefreshOrLoadFunc(ScreenObj.Inf.Key, "", false)})
               {
                  // the clear search string image //
                  Image(systemName:"multiply.circle.fill")

                  // set the image color //
                  .foregroundColor(.gray)
               }

               // set the custom button style //
               .buttonStyle(CustomButton())

               // set the image size //
               .frame(width:25, height:25, alignment:.center)

               // add the trailing padding after the clear icon image //
               .padding([.trailing], 5)
            }
         }
```

```
// add the rounded rectangle border color based if in focus or not //
.background(RoundedRectangle(cornerRadius:10, style:.continuous)
        .strokeBorder(InFocus ? Color.blue.opacity(0.65) : Color
        .gray.opacity(0.65), lineWidth:2))

// set the search box frame size //
.frame(width:630, height:30, alignment:.center)

// set the system font //
.font(.system(size:14))

// add the leading padding before the search text field //
.padding([.leading], 14)

// add the padding after the search text field //
.padding([.bottom], 15)

// the scroll view reader //
ScrollViewReader {SView in LazyVStack(alignment:.leading, spacing:0)
{
   // the list of games //
   List(ScreenObj.GameList.indices, id:\.self)
   {
      // loop through the list rows //
      Index in LazyVStack(alignment:.leading, spacing:0)
      {
         // the hstack //
         HStack(spacing:0)
         {
            // the game list description text //
            Text(ScreenObj.GameList[Index])

            // set the font type //
            .font(ScreenObj.FontType)

            // set the font color //
            .foregroundColor(ScreenObj.TextColor)

            // add padding around the game description text //
            .padding(25)

            // set the line spacing //
            .lineSpacing(15)

            // set the game list description text frame size //
            .frame(width:660, height:nil, alignment:.leading)
         }

         // the content shape //
         .contentShape(Rectangle())

         // set the action for when double click on the row //
         .onTapGesture(count:2)
         {
            // set the current selection index //
            ScreenObj.Selection = Index

            // load the selected game from the list //
            ScreenObj.Inf.RefreshOrLoadFunc(ScreenObj.Inf.Key,
                                           ScreenObj.Entered, true)
         }

         // the single tap gesture to select the list row //
         .simultaneousGesture(TapGesture().onEnded {ScreenObj
                             .Selection = Index})

         // set the list row background color //
         .background(ScreenObj.Selection == Index ? Color(red:0.75,
                    green:0.85, blue:1) : Color(red:0.87, green:0.92,
                    blue:1))

         // add a list row divider //
         Divider()

         // set the row divider frame //
         .frame(width:660, height:4)

         // set the row divider color //
         .overlay(.gray.opacity(0.5))
      }

      // add the top and bottom row padding //
      .padding([.top, .bottom], -6)
   }

   // set the edge insets //
   .padding(EdgeInsets(top:0, leading:-8, bottom:0, trailing:0))

   // set the scroll list frame size //
   .frame(width:660, height:550, alignment:.center)

   // set the list style to plain //
   .listStyle(.plain)

   // do on change of the given command //
   .onChange(of:ScreenObj.Command)
   {
      // the command variable //
      Command in

      // if need to start the key code monitoring //
      if Command == "Start"
      {
         // set the focus to be true //
         InFocus = true

         // start the key input monitoring for key and mouse moved //
         KeyInput = NSEvent.addLocalMonitorForEvents(matching:
                            [.keyDown, .mouseMoved])
         {
            // the event to process //
            Event in

            // if is for the mouse moving //
            if Event.type == NSEvent.EventType.mouseMoved
            {
               // if over text search box area and in focus //
               if (50...(!GetString().FromTextField(ScreenObj.Entered)
                  .isEmpty ? 612 : 640)).contains(NSEvent.mouseLocation
                  .x - ScreenObj.Inf.Window.frame.origin.x) &&
                  (580...605).contains(NSEvent.mouseLocation.y -
                  ScreenObj.Inf.Window.frame.origin.y) && InFocus
               {
                  // set the cursor to be the ibeam //
                  NSCursor.iBeam.set()
               }
               // if outside of the text search box area //
               else
               {
                  // set the cursor to be an arrow //
                  NSCursor.arrow.set()
               }
```

```
                        // return the event //
                        return Event
                     }
                     // if is the key code for up //
                     else if Event.keyCode == 126
                     {
                        // go back one row in the list bounded by 0 //
                        ScreenObj.Selection = max(0, ScreenObj.Selection - 1)

                        // return nil so the event isn't further processed //
                        return nil
                     }
                     // if is the key code for down //
                     else if Event.keyCode == 125
                     {
                        // go forward one row in the list bounded by max rows //
                        ScreenObj.Selection = min(ScreenObj.GameList.count - 1,
                                                 ScreenObj.Selection + 1)

                        // return nil so the event isn't further processed //
                        return nil
                     }

                     // return the event //
                     return Event
                  }
               }
               // if is the command to stop the key code monitoring //
               else if Command == "End"
               {
                  // if the key input monitor is valid //
                  if KeyInput != nil
                  {
                     // remove the key code event monitoring //
                     NSEvent.removeMonitor(KeyInput!)

                     // set the key code input to nil //
                     KeyInput = nil
                  }

                  // set that search text is no longer in focus //
                  InFocus = false
               }
            }

            // if the list row selection changed //
            .onChange(of:ScreenObj.Selection)
            {
               // scroll to the current row selection //
               _ in SView.scrollTo(ScreenObj.Selection)
            }

            // if the list needs to be refreshed //
            .onChange(of:ScreenObj.Refresh)
            {
               // scroll to the current row selection //
               _ in SView.scrollTo(ScreenObj.Selection)
            }
         }}
      }

      // set the scroll view reader frame size //
      .frame(width:660, height:600, alignment:.center)

      // set the scrollview reader position //
      .position(x:330, y:-330)
   }
}

// the date picker structure //
struct DateStruct:View
{
   // the observed screen view object //
   @ObservedObject var ScreenObj:Obj = Obj()

   // the observed screen view initializer //
   init(_ ScreenObj:Obj) {self.ScreenObj = ScreenObj}

   // the main body //
   var body:some View
   {
      // the date picker //
      DatePicker("", selection:Binding(get:{ScreenObj.GameDate}, set:{GameDate
                in ScreenObj.GameDate = GameDate; ScreenObj.Inf
                .DateFunc(ScreenObj.Inf.Key, ScreenObj.GameDate)}),
                displayedComponents:[.date])

      // set the font size //
      .font(.system(size:15))

      // set the border color and width //
      .border(.gray, width:1)

      // set the labels to be hidden //
      .labelsHidden()

      // set the frame size //
      .frame(width:ScreenObj.Inf.Width, height:ScreenObj.Inf.Height)

      // set the frame position //
      .position(x:ScreenObj.Inf.XP, y:ScreenObj.Inf.YP)
   }
}

// the entered text field structure //
struct EnteredStruct:View
{
   // the observed screen view object //
   @ObservedObject var ScreenObj:Obj = Obj()

   // the observed screen view initializer //
   init(_ ScreenObj:Obj) {self.ScreenObj = ScreenObj}

   // the main body //
   var body:some View
   {
      // the hstack //
      HStack()
      {
         // the text field label //
         Text(ScreenObj.Inf.Label)

         // set the font size //
         .font(.system(size:15))

         // set the frame size //
         .frame(width:200, height:ScreenObj.Inf.Height, alignment:.trailing)

         // the text field enter box //
         TextField(ScreenObj.Inf.Placeholder, text:Binding(get:{ScreenObj
                  .Entered}, set:{Entered in ScreenObj.Entered = Entered;
                  ScreenObj.Inf.EnteredFunc(ScreenObj.Inf.Key, ScreenObj
                  .Entered)}))
```

```
         // set the text field style to rounded border //
         .textFieldStyle(.roundedBorder)

         // set the overlay rounded rectangle //
         .overlay(RoundedRectangle(cornerRadius:7, style:.continuous)
                 .strokeBorder(Color.gray, lineWidth:1))

         // set the font size //
         .font(.system(size:14))

         // set the frame size //
         .frame(width:ScreenObj.Inf.Width, height:ScreenObj.Inf.Height)
      }

      // set the view position //
      .position(x:ScreenObj.Inf.XP, y:ScreenObj.Inf.YP)
   }
}

// the picker view structure //
struct PickerStruct:View
{
   // the observed screen view object //
   @ObservedObject var ScreenObj:Obj = Obj()

   // the observed screen view initializer //
   init(_ ScreenObj:Obj) {self.ScreenObj = ScreenObj}

   // the main body //
   var body:some View
   {
      // the picker for the view //
      Picker("", selection:Binding(get:{ScreenObj.Selection}, set:{Selection in
            ScreenObj.Selection = Selection; ScreenObj.Inf.PickerFunc(ScreenObj
            .Inf.Key, ScreenObj.Selection)}), content:{ForEach(0..<ScreenObj
            .Inf.Items.count, id:\.self) {Index in Text(" " + ScreenObj.Inf
            .Items[Index])}})

      // set the picker style to menu //
      .pickerStyle(.menu)

      // set the frame size //
      .frame(width:ScreenObj.Inf.Width, height:ScreenObj.Inf.Height)

      // set the view position //
      .position(x:ScreenObj.Inf.XP, y:ScreenObj.Inf.YP)

      // set the shadow color //
      .shadow(color:Color(red:0.25, green:0.25, blue:0.25), radius:0.5, x:0,
             y:0)
   }
}

// a text view structure //
struct TextStruct:View
{
   // the observed screen view object //
   @ObservedObject var ScreenObj:Obj = Obj()

   // the observed screen view initializer //
   init(_ ScreenObj:Obj) {self.ScreenObj = ScreenObj}

   // the main body //
   var body:some View
   {
      // the text for the view //
      Text(ScreenObj.Inf.Label)

      // set the font size //
      .font(.system(size:15))

      // set the frame size //
      .frame(width:ScreenObj.Inf.Width, height:ScreenObj.Inf.Height, alignment:
            .center)

      // set the view position //
      .position(x:ScreenObj.Inf.XP, y:ScreenObj.Inf.YP)
   }
}

// an image view structure //
struct ImageStruct:View
{
   // the observed screen view object //
   @ObservedObject var ScreenObj:Obj = Obj()

   // the observed screen view initializer //
   init(_ ScreenObj:Obj) {self.ScreenObj = ScreenObj}

   // the main body //
   var body:some View
   {
      // the vertical view stack //
      VStack(alignment:.leading)
      {
         // the image for the view //
         Image(ScreenObj.Preview, scale:2, orientation:Image.Orientation.up,
              label:Text(""))
      }

      // set the frame size //
      .frame(width:ScreenObj.Inf.Width, height:ScreenObj.Inf.Height)

      // set the view position //
      .position(x:ScreenObj.Inf.XP, y:ScreenObj.Inf.YP)
   }
}

// a toggle button view structure //
struct ToggleStruct:View
{
   // the observed screen view object //
   @ObservedObject var ScreenObj:Obj = Obj()

   // the observed screen view initializer //
   init(_ ScreenObj:Obj) {self.ScreenObj = ScreenObj}

   // the main body //
   var body:some View
   {
      // the toggle button for the view //
      Toggle(isOn:Binding(get:{ScreenObj.Status}, set:{Status in ScreenObj
            .Status = Status; ScreenObj.Inf.ToggleFunc(ScreenObj.Inf.Key)}))
      {
         // the text for the view //
         Text(ScreenObj.Inf.Label)

         // set the padding //
         .padding(2)

         // set the frame size //
         .frame(width:ScreenObj.Inf.Width, height:ScreenObj.Inf.Height,
               alignment:.leading)
      }
```

```
      // set the toggle style //
      .toggleStyle(SwitchToggleStyle())

      // set the control size //
      .controlSize(ControlSize.regular)

      // set the font size //
      .font(.system(size:14))

      // set the view position //
      .position(x:ScreenObj.Inf.XP, y:ScreenObj.Inf.YP)
   }
}

// a slider view structure //
struct SliderStruct:View
{
   // the observed screen view object //
   @ObservedObject var ScreenObj:Obj = Obj()

   // the observed screen view initializer //
   init(_ ScreenObj:Obj) {self.ScreenObj = ScreenObj}

   // the main body //
   var body:some View
   {
      // the vertical view stack with alignment and spacing //
      VStack(alignment:.leading, spacing:4)
      {
         // the slider for the view //
         Slider(value:Binding(get:{ScreenObj.Val}, set:{Val in ScreenObj.Val =
               Val}), in:ScreenObj.MinV...ScreenObj.MaxV)

         // set the shadow color //
         .shadow(color:Color(red:0.25, green:0.25, blue:0.25), radius:0.5, x:0,
                y:0)

         // set the control size //
         .controlSize(ControlSize.regular)

         // the text for the view //
         Text(ScreenObj.Inf.SliderFunc(ScreenObj.Inf.Key, ScreenObj.Val))

         // set the font size //
         .font(.system(size:15))
      }

      // set the frame size //
      .frame(width:ScreenObj.Inf.Width, height:ScreenObj.Inf.Height)

      // set the view position //
      .position(x:ScreenObj.Inf.XP, y:ScreenObj.Inf.YP)
   }
}

// the observable object //
class Obj:ObservableObject
{
   // the published image //
   @Published var Preview:CGImage = CoreGraphics().InitImage()

   // publish if will refresh the view //
   @Published var Refresh:Bool = false

   // the published command //
   @Published var Command:String = ""

   // the published value //
   @Published var Val:Double = 0

   // the published minimum value //
   @Published var MinV:Double = 0

   // the published maximum value //
   @Published var MaxV:Double = 1

   // the published button status //
   @Published var Status:Bool = false

   // the published selection value //
   @Published var Selection:Int = 0

   // the hovered toolbar index //
   @Published var Hovered:Int = -1

   // the published font type //
   @Published var FontType:Font = .system(size:16, weight:.regular)

   // the published text color //
   @Published var TextColor = Color.black

   // the published entered text string //
   @Published var Entered:String = ""

   // the published date picker //
   @Published var GameDate:Date = Date()

   // the published game list //
   @Published var GameList:[String] = []

   // the screen view information //
   var Inf:ScreenInfo = ScreenInfo()

   // a color view //
   var ColorView:ColorsView!

   // a clock view //
   var ClockView:TimeClockDisplay!

   // the object view //
   var ObjView:NSView!

   // the title bar //
   var TitleBar:NSTitlebarAccessoryViewController =
               NSTitlebarAccessoryViewController()

   // the observable object initializer for a slider view //
   init(_ InfVals:InfoValues, InfSlide:ScreenInfo)
   {
      // initialize the observable object slider view attributes //
      (Val, MinV, MaxV, Inf, ObjView) = (Double(InfVals.VI), Double(InfVals
      .VMinInt), Double(InfVals.VMaxInt), InfSlide, NSHostingView(rootView:
      SliderStruct(self)))
   }

   // the observable object initializer for a toolbar //
   init(_ Selection:Int, InfBar:ScreenInfo)
   {
      // initialize the observable object toolbar view attributes //
      (self.Selection, Inf, ObjView) = (Selection, InfBar,
      NSHostingView(rootView:ToolbarStruct(self)))

      // set the title bar //
      TitleBar.view = ObjView

      // set the title bar view frame height //
      TitleBar.view.frame.size.height = 35

      // add the title bar to the window toolbar area //
      Inf.Window?.addTitlebarAccessoryViewController(TitleBar)
   }

   // the observable object initializer for a picker view //
   init(_ Selection:Int, InfPick:ScreenInfo)
   {
      // initialize the oberservable object picker view attributes //
      (self.Selection, Inf, ObjView) = (Selection, InfPick,
      NSHostingView(rootView:PickerStruct(self)))

      // add the object view to the window if the subwindow is empty //
      if Inf.SubWindow.isEmpty {Inf.Window.contentView!.addSubview(ObjView)}
   }

   // the observable object initializer for the game list view //
   init(InfLst:ScreenInfo)
   {
      // initialize the observable game list view //
      (Inf, ObjView) = (InfLst, NSHostingView(rootView:GameListStruct(self)))

      // add the object view to the window //
      Inf.Window.contentView!.addSubview(ObjView)
   }

   // the observable object initializer for an enter text view //
   init(_ Entered:String, InfEnt:ScreenInfo)
   {
      // initialize the observable object enter text view attributes //
      (self.Entered, Inf, ObjView) = (Entered, InfEnt, NSHostingView(rootView:
      EnteredStruct(self)))

      // add the object view to the window //
      Inf.Window.contentView!.addSubview(ObjView)
   }

   // the observable object initializer for a date picker view //
   init(_ GameDate:Date, InfDte:ScreenInfo)
   {
      // initialize the observable object date picker view attributes //
      (self.GameDate, Inf, ObjView) = (GameDate, InfDte, NSHostingView(rootView:
      DateStruct(self)))

      // add the object view to the window //
      Inf.Window.contentView!.addSubview(ObjView)
   }

   // the observable object initializer for a toggle view //
   init(_ Status:Bool, InfTogg:ScreenInfo)
   {
      // initialize the observable object toggle view attributes //
      (self.Status, Inf, ObjView) = (Status, InfTogg, NSHostingView(rootView:
      ToggleStruct(self)))
   }

   // the observable object initializer for an image view //
   init(InfImage:ScreenInfo)
   {
      // initialize the observable object image view attributes //
      (Inf, ObjView) = (InfImage, NSHostingView(rootView:ImageStruct(self)))
   }

   // the observable object initializer for a text view //
   init(InfText:ScreenInfo)
   {
      // initialize the observable object text view attributes //
      (Inf, ObjView) = (InfText, NSHostingView(rootView:TextStruct(self)))

      // add the object view to the window if the subwindow is empty //
      if Inf.SubWindow.isEmpty {Inf.Window.contentView!.addSubview(ObjView)}
   }

   // the object initializer for a color view //
   init(_ ColorView:ColorsView, InfCol:ScreenInfo)
   {
      // initialize the color view attributes //
      (self.ColorView, ColorView.frame.origin, Inf, ObjView) = (ColorView,
      CGPoint(x:InfCol.OX, y:InfCol.OY), InfCol, ColorView)
   }

   // the object initializer for a clock view //
   init(_ ClockView:TimeClockDisplay, InfClk:ScreenInfo)
   {
      // initialize the clock view attributes //
      (self.ClockView, ClockView.frame.origin, Inf, ObjView) = (ClockView,
      CGPoint(x:InfClk.OX, y:InfClk.OY), InfClk, ClockView)
   }

   // the game list index //
   func GameListIndex() -> Int
   {
      // return the game list index //
      return Int(GameList[Selection].components(separatedBy: " ").first!)! - 1
   }

   // the default initializer //
   init() {}
}

// the public user interface class //
public class GCUserInterface
{
   // a dictionary of user interface objects //
   var Objs:[String:Obj] = [:]

   // the main board //
   var Board:GreenChess!

   // the board and piece compositor //
   var BPComp:BoardPieceCompositor = BoardPieceCompositor()

   // the game attributes //
   var Attributes:GameAttributes = GameAttributes()

   // the pgn moves data //
   var PGNMovesData:PGNMovesDataSource = PGNMovesDataSource()

   // the game sounds //
   var GameSounds:GameSoundEffects = GameSoundEffects()

   // the frames and movie creator //
   var FramesMovie:FramesMovieCreator = FramesMovieCreator()

   // a dictionary key //
   var Key:String = ""

   // the enter game info pgn tags toolbar information //
   var TagsBar:[(String, Color, String, String)] = [("xmark.circle.fill",
               Color.gray, "", "")]

   // the game list toolbar information //
   var ListBar:[(String, Color, String, String)] = [("xmark.circle.fill",
               Color.gray, "", ""), ("arrow.up.circle.fill", Color.green,
               "Load", "")]

   // the preferences toolbar information //
   var PrefsBar:[(String, Color, String, String)] = [("xmark.circle.fill",
                Color.gray, "", ""), ("rectangle.fill", Color.green,
                "Background", "PrefsBackground"), ("checkerboard.rectangle",
                Color.blue, "Board", "PrefsBoard"), ("clock", Color.purple,
                "Clock", "PrefsClock"), ("photo", Color.teal, "Diagrams",
                "PrefsDiagram"), ("textformat", Color.red, "Font", "PrefsFont"),
                ("square.grid.3x2.fill", Color.cyan, "Gallery", "PrefsGallery"),
                ("photo.stack.fill", Color.mint, "Movie", "PrefsMovie"),
                ("speedometer", Color.orange, "Speed", "PrefsSpeed"),
                ("gearshape.2.fill", Color.indigo, "Stockfish",
                "PrefsStockfish")]
```

```
// update the visibility of the screen controls //
func UpdateVisibility(_ SubWindow:String)
{
   // loop over all the values in the user interface objects dictionary //
   for (_, ScreenObj) in Objs
   {
      // if matches the subwindow to make visible //
      if ScreenObj.Inf.SubWindow == SubWindow
      {
         // add the control to the window //
         ScreenObj.Inf.Window.contentView!.addSubview(ScreenObj.ObjView)
      }
      // if the subwindow is not empty //
      else if !ScreenObj.Inf.SubWindow.isEmpty
      {
         // remove the control from the window //
         ScreenObj.ObjView.removeFromSuperview()
      }
   }
}

// get the current color view //
func GetColorView(_ Title:String) -> ColorsView
{
   // return the color view //
   return Objs[Title]!.ColorView
}

// get the current clock view //
func GetClockView(_ Title:String) -> TimeClockDisplay
{
   // return the clock view //
   return Objs[Title]!.ClockView
}

// refresh the views //
func RefreshViews(_ Views:[String])
{
   // loop over the views to refresh //
   for TheView in Views {Objs[TheView]!.Refresh.toggle()}
}

// add the nsviews to the user interface //
func AddNSViews(_ Board:GreenChess)
{
   // set the board //
   self.Board = Board

   // add the window color view to the screen dictionary //
   Objs.updateValue(Obj(ColorsView().Create(6), InfCol:
        ScreenInfo("PrefsBackground", OX:30, OY:295, Window:Board
        .PreferencesWindow)), forKey:"WindowColorView")

   // add the board labels color view to the screen dictionary //
   Objs.updateValue(Obj(ColorsView().Create(0), InfCol:
        ScreenInfo("PrefsFont", OX:30, OY:520, Window:Board
        .PreferencesWindow)), forKey:"BoardLabelsColorView")

   // add the game info color view to the screen dictionary //
   Objs.updateValue(Obj(ColorsView().Create(3), InfCol:
        ScreenInfo("PrefsFont", OX:30, OY:365, Window:Board
        .PreferencesWindow)),forKey:"GameInfoColorView")

   // add the move list color view to the screen dictionary //
   Objs.updateValue(Obj(ColorsView().Create(4), InfCol:
        ScreenInfo("PrefsFont", OX:30, OY:210, Window:Board
        .PreferencesWindow)), forKey:"MoveListColorView")

   // add the game list color view to the screen dictionary //
   Objs.updateValue(Obj(ColorsView().Create(5), InfCol:
        ScreenInfo("PrefsFont", OX:30, OY:55, Window:Board
        .PreferencesWindow)), forKey:"GameListColorView")

   // add the clock surface color view to the screen dictionary //
   Objs.updateValue(Obj(ColorsView().Create(2), InfCol:
        ScreenInfo("PrefsClock", OX:30, OY:260, Window:Board
        .PreferencesWindow)), forKey:"ClockSurfaceColorView")

   // add the clock color view to the screen dictionary //
   Objs.updateValue(Obj(ColorsView().Create(1), InfCol:
        ScreenInfo("PrefsClock", OX:30, OY:380, Window:Board
        .PreferencesWindow)), forKey:"ClockColorView")

   // add the preview clock view to the screen dictionary //
   Objs.updateValue(Obj(TimeClockDisplay(), InfClk:ScreenInfo("PrefsClock",
        OX:280, OY:505, Window:Board.PreferencesWindow)), forKey:"ClockView")
}

// add the swiftui views to the user interface //
func AddSwiftUIViews()
{
   // add the movie frames per second slider to the screen dictionary //
   Key = "MovieFPSFromSlider"; Objs.updateValue(Obj(FramesMovie.Inf
         .Vals(Key), InfSlide:ScreenInfo(Key, SubWindow:"PrefsMovie", XP:330,
         YP:-590, Width:600, Height:30, Window:Board.PreferencesWindow,
         SliderFunc:FramesMovie.UpdateMovieFPS)), forKey:Key)

   // add the movie size slider to the screen dictionary //
   Key = "MovieSize"; Objs.updateValue(Obj(FramesMovie.Inf.Vals(Key),
         InfSlide:ScreenInfo(Key, SubWindow:"PrefsMovie", XP:330, YP:-500,
         Width:600, Height:30, Window:Board.PreferencesWindow, SliderFunc:
         FramesMovie.UpdateMovieSize)), forKey:Key)

   // add the board animation rate slider to the screen dictionary //
   Key = "AnimationRate"; Objs.updateValue(Obj(Board.Inf.Vals(Key), InfSlide:
         ScreenInfo(Key, SubWindow:"PrefsSpeed", XP:330, YP:-590, Width:600,
         Height:30, Window:Board.PreferencesWindow, SliderFunc:Board
         .UpdateAnimationRate)), forKey:Key)

   // add the search time slider to the screen dictionary //
   Key = "SearchTime"; Objs.updateValue(Obj(Board.Inf.Vals(Key), InfSlide:
         ScreenInfo(Key, SubWindow:"PrefsSpeed", XP:330, YP:-500, Width:600,
         Height:30, Window:Board.PreferencesWindow, SliderFunc:
         Board.UpdateSearchTime)), forKey:Key)

   // add the uci engine difficulty slider to the screen dictionary //
   Key = "UCIEngineDifficulty"; Objs.updateValue(Obj(Board.Inf.Vals(Key),
         InfSlide:ScreenInfo(Key, SubWindow:"PrefsStockfish", XP:330, YP:
         -590, Width:600, Height:30, Window:Board.PreferencesWindow,
         SliderFunc:Board.UpdateUCIEngineDifficulty)), forKey:Key)

   // add the uci engine hash slider to the screen dictionary //
   Key = "UCIEngineHash"; Objs.updateValue(Obj(Board.Inf.Vals(Key), InfSlide:
         ScreenInfo(Key, SubWindow:"PrefsStockfish", XP:330, YP:-500, Width:
         600, Height:30, Window:Board.PreferencesWindow, SliderFunc:Board
         .UpdateUCIEngineHash)), forKey:Key)

   // add the multithread stockfish toggle to the screen dictionary //
   Key = "MultithreadUCIEngine"; Objs.updateValue(Obj(Board.Inf.Vals(Key).VB,
         InfTogg:ScreenInfo("Multithread Stockfish Engine", Key:Key,
         SubWindow:"PrefsStockfish", XP:155, YP:-420, Width:210, Height:30,
         Window:Board.PreferencesWindow, ToggleFunc:Board
         .UpdateMultithreadStockfish)), forKey:Key)
```

```
// add the show stockfish analysis toggle to the screen dictionary //
Key = "ShowStockfishAnalysis"; Objs.updateValue(Obj(Board.Inf.Vals(Key)
      .VB, InfTogg:ScreenInfo("Show Stockfish Engine Analysis", Key:Key,
      SubWindow:"PrefsStockfish", XP:155, YP:-345, Width:210, Height:30,
      Window:Board.PreferencesWindow, ToggleFunc:Board
      .UpdateStockfishAnalysis)), forKey:Key)

// add the number of grid rows picker to the screen dictionary //
Key = "NumberOfGridRows"; Objs.updateValue(Obj(Board.Inf.Vals(Key).VI - 2,
      InfPick:ScreenInfo(Key, SubWindow:"PrefsGallery", XP:330, YP:-595,
      Width:310, Height:30, Window:Board.PreferencesWindow, Items:Board
      .GridRows, PickerFunc:Board.SetNumberOfGridRows)), forKey:Key)

// add the gallery animation rate slider to the screen dictionary //
Key = "GalleryAnimationRate"; Objs.updateValue(Obj(Board.Inf.Vals(Key),
      InfSlide:ScreenInfo(Key, SubWindow:"PrefsGallery", XP:330, YP:-510,
      Width:600, Height:30, Window:Board.PreferencesWindow, SliderFunc:
      Board.UpdateGalleryAnimationRate)), forKey:Key)

// add the chess piece set type picker to the screen dictionary //
Key = "ChessPieceSetType"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key).VI,
      InfPick:ScreenInfo(Key, SubWindow:"PrefsBoard", XP:503, YP:-550,
      Width:165, Height:30, Window:Board.PreferencesWindow, Items:
      BPComp.PieceImages.Sets, PickerFunc:BPComp.UpdatePieceSetType)),
      forKey:Key)

// add the board type picker to the screen dictionary //
Key = "BoardType"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key).VI, InfPick:
      ScreenInfo(Key, SubWindow:"PrefsBoard", XP:503, YP:-595, Width:165,
      Height:30, Window:Board.PreferencesWindow, Items:BPComp.BoardImages
      .Boards, PickerFunc:BPComp.UpdateBoardTypeLayer)), forKey:Key)

// add the border texture slider to the screen dictionary //
Key = "BorderTextureColor"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key),
      InfSlide:ScreenInfo(Key, SubWindow:"PrefsBoard", XP:170, YP:-155,
      Width:275, Height:30, Window:Board.PreferencesWindow, SliderFunc:
      BPComp.UpdateBorderTexture)), forKey:Key)

// add the border visibility slider to the screen dictionary //
Key = "BorderVisibility"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key),
      InfSlide:ScreenInfo(Key, SubWindow:"PrefsBoard", XP:170, YP:-100,
      Width:275, Height:30, Window:Board.PreferencesWindow, SliderFunc:
      BPComp.UpdateBorderVisibility)), forKey:Key)

// add the border visibility slider to the screen dictionary //
Key = "SquareBorder"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key), InfSlide:
      ScreenInfo(Key, SubWindow:"PrefsBoard", XP:170, YP:-50, Width:270,
      Height:30, Window:Board.PreferencesWindow, SliderFunc:BPComp
      .UpdateSquareBorderOpacity)), forKey:Key)

// add the board roughness slider to the screen dictionary //
Key = "BoardRoughness"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key),
      InfSlide:ScreenInfo(Key, SubWindow:"PrefsBoard", XP:495, YP:-155,
      Width:275, Height:30, Window:Board.PreferencesWindow, SliderFunc:
      BPComp.UpdateBoardRoughness)), forKey:Key)

// add the board tint hue slider to the screen dictionary //
Key = "BoardTintHue"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key), InfSlide:
      ScreenInfo(Key, SubWindow:"PrefsBoard", XP:495, YP:-100, Width:275,
      Height:30, Window:Board.PreferencesWindow, SliderFunc:BPComp
      .UpdateBoardTintColor)), forKey:Key)

// add the board tint visibility slider to the screen dictionary //
Key = "BoardTintVisibility"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key),
      InfSlide:ScreenInfo(Key, SubWindow:"PrefsBoard", XP:495, YP:-45,
      Width:275, Height:30, Window:Board.PreferencesWindow, SliderFunc:
      BPComp.UpdateBoardTintVisibility)), forKey:Key)

// add the preferences board preview image to the screen dictionary //
Key = "BoardPreview"; Objs.updateValue(Obj(InfImage:
      ScreenInfo("PrefsBoard", XP:218, YP:-405, Width:400, Height:400,
      Window:Board.PreferencesWindow)), forKey:Key)

// add the autosave game toggle to the screen dictionary //
Key = "AutosaveGame"; Objs.updateValue(Obj(Board.Inf.Vals(Key).VB,
      InfTogg:ScreenInfo("Autosave Game", Key:Key, SubWindow:"PrefsBoard",
      XP:536, YP:-505, Width:174, Height:30, Window:Board
      .PreferencesWindow, ToggleFunc:Board.UpdateAutosaveGame)),
      forKey:Key)

// add the game sounds off toggle the screen dictionary //
Key = "SoundOff"; Objs.updateValue(Obj(!Board.Inf.Vals(Key).VB, InfTogg:
      ScreenInfo("Sound", Key:Key, SubWindow:"PrefsBoard", XP:536, YP:
      -470, Width:174, Height:30, Window:Board.PreferencesWindow,
      ToggleFunc:Board.UpdateGameSoundsOff)), forKey:Key)

// add the use eco book toggle to the screen dictionary //
Key = "UseEcoBook"; Objs.updateValue(Obj(Board.Inf.Vals(Key).VB, InfTogg:
      ScreenInfo("Opening Book", Key:Key, SubWindow:"PrefsBoard", XP:536,
      YP:-435, Width:174, Height:30, Window:Board.PreferencesWindow,
      ToggleFunc:Board.UpdateUseEcoBook)), forKey:Key)

// add the highlight last move made toggle to the screen dictionary //
Key = "HighlightLastMoveMade"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key)
      .VB, InfTogg:ScreenInfo("Highlight Last Move", Key:Key, SubWindow:
      "PrefsBoard", XP:536, YP:-400, Width:174, Height:30, Window:Board
      .PreferencesWindow, ToggleFunc:BPComp.UpdateToggleStatus)),
      forKey:Key)

// add the highlight captures toggle to the screen dictionary //
Key = "HighlightCapturedPieces"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key)
      .VB, InfTogg:ScreenInfo("Highlight Captures", Key:Key, SubWindow:
      "PrefsBoard", XP:536, YP:-365, Width:174, Height:30, Window:Board
      .PreferencesWindow, ToggleFunc:BPComp.UpdateToggleStatus)),
      forKey:Key)

// add the highlight threats toggle to the screen dictionary //
Key = "HighlightThreatenedPieces"; Objs.updateValue(Obj(BPComp.Inf
      .Vals(Key).VB, InfTogg:ScreenInfo("Highlight Threats", Key:Key,
      SubWindow:"PrefsBoard", XP:536, YP:-330, Width:174, Height:30,
      Window:Board.PreferencesWindow, ToggleFunc:BPComp
      .UpdateToggleStatus)), forKey:Key)

// add the move by selecting squares toggle to the screen dictionary //
Key = "MoveBySelectingSquares"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key)
      .VB, InfTogg:ScreenInfo("Move by Selecting Square", Key:Key,
      SubWindow:"PrefsBoard", XP:536, YP:-295, Width:174, Height:30,
      Window:Board.PreferencesWindow, ToggleFunc:BPComp
      .UpdateToggleStatus)), forKey:Key)

// add the display captured pieces toggle to the screen dictionary //
Key = "HideCapturedPiecesInBoard"; Objs.updateValue(Obj(!BPComp.Inf
      .Vals(Key).VB, InfTogg:ScreenInfo("Display Captured Pieces", Key:
      Key, SubWindow:"PrefsBoard", XP:536, YP:-260, Width:174, Height:30,
      Window:Board.PreferencesWindow, ToggleFunc:BPComp
      .UpdateToggleStatus)), forKey:Key)

// add the rotate board toggle to the screen dictionary //
Key = "RotateBoard"; Objs.updateValue(Obj(BPComp.Inf.Vals(Key).VB,
      InfTogg:ScreenInfo("Rotate Board", Key:Key, SubWindow:"PrefsBoard",
      XP:536, YP:-225, Width:174, Height:30, Window:Board
      .PreferencesWindow, ToggleFunc:BPComp.UpdateToggleStatus)), forKey:
      Key)
```

```
// add the window texture opacity slider to the screen dictionary //
Key = "WindowTextureOpacity"; Objs.updateValue(Obj(Board.Inf.Vals(Key),
      InfSlide:ScreenInfo(Key, SubWindow:"PrefsBackground", XP:330, YP:
      -215, Width:600, Height:30, Window:Board.PreferencesWindow,
      SliderFunc:Board.UpdateWindowTextureOpacity)), forKey:Key)

// add preferences window background preview image to screen dictionary //
Key = "WindowPreview"; Objs.updateValue(Obj(InfImage:
      ScreenInfo("PrefsBackground", XP:330, YP:-505, Width:200, Height:
      200, Window:Board.PreferencesWindow)), forKey:Key)

// add the window background title text to the screen dictionary //
Key = "WindowBackgroundTitle"; Objs.updateValue(Obj(InfText:
      ScreenInfo("Window Background", SubWindow:"PrefsBackground", XP:330,
      YP:-347, Width:200, Height:30, Window:Board.PreferencesWindow)),
      forKey:Key)

// add the preferences diagram preview image to the screen dictionary //
Key = "DiagramPreview"; Objs.updateValue(Obj(InfImage:
      ScreenInfo("PrefsDiagram", XP:500, YP:-480, Width:250, Height:250,
      Window:Board.PreferencesWindow)), forKey:Key)

// add the hide illustrations toggle to the screen dictionary //
Key = "HideIllustrations"; Objs.updateValue(Obj(!Board.Inf.Vals(Key).VB,
      InfTogg:ScreenInfo("Display Diagrams in Game Moves", Key:Key,
      SubWindow:"PrefsDiagram", XP:162, YP:-480, Width:225, Height:30,
      Window:Board.PreferencesWindow, ToggleFunc:Board
      .UpdateHideIllustrations)), forKey:Key)

// add the illustration size slider to the screen dictionary //
Key = "IllustrationSize"; Objs.updateValue(Obj(Board.Inf.Vals(Key),
      InfSlide:ScreenInfo(Key, SubWindow:"PrefsDiagram", XP:330, YP:-290,
      Width:600, Height:30, Window:Board.PreferencesWindow, SliderFunc:
      Board.UpdateIllustrationSize)), forKey:Key)

// add the display game clock toggle to the screen dictionary //
Key = "HideGameClockInPanel"; Objs.updateValue(Obj(!Board.Inf.Vals(Key)
       .VB, InfTogg:ScreenInfo("Display Game Clock", Key:Key, SubWindow:
       "PrefsClock", XP:118, YP:-555, Width:135, Height:30, Window:Board
       .PreferencesWindow, ToggleFunc:Board.UpdateDisplayGameClock)),
       forKey:Key)

// add the clock background visibility slider to the screen dictionary //
Key = "ClockBackgroundVisibility"; Objs.updateValue(Obj(Board.Inf
      .Vals(Key), InfSlide:ScreenInfo(Key, SubWindow:"PrefsClock", XP:330,
      YP:-170, Width:600, Height:30, Window:Board.PreferencesWindow,
      SliderFunc:Board.UpdateClockBackgroundVisibility)), forKey:Key)

// add the clock background title text to the screen dictionary //
Key = "GameClockBackgroundTitle"; Objs.updateValue(Obj(InfText:
      ScreenInfo("Background", SubWindow:"PrefsClock", XP:330, YP:-312,
      Width:200, Height:30, Window:Board.PreferencesWindow)), forKey:Key)

// add the clock border title text to the screen dictionary //
Key = "GameClockBorderTitle"; Objs.updateValue(Obj(InfText:
      ScreenInfo("Border", SubWindow:"PrefsClock", XP:330, YP:-432, Width:
      200, Height:30, Window:Board.PreferencesWindow)), forKey:Key)

// add the board labels font type picker to the screen dictionary //
Key = "BoardLabelsFontType"; Objs.updateValue(Obj(BPComp.BoardLabelsFont
      .Inf.Vals(Key).VI, InfPick:ScreenInfo(Key, SubWindow:"PrefsFont",
      XP:427, YP:-575, Width:195, Height:30, Window:Board
      .PreferencesWindow, Items:BPComp.BoardLabelsFont.Fonts,
      PickerFunc:Board.UpdateBoardLabelsFont)), forKey:Key)

// add the game info font type picker to the screen dictionary //
Key = "GameInfoFontType"; Objs.updateValue(Obj(Board.GameInfoFont.Inf
      .Vals(Key).VI, InfPick:ScreenInfo(Key, SubWindow:"PrefsFont", XP:
      427, YP:-420, Width:195, Height:30,Window:Board.PreferencesWindow,
      Items:Board.GameInfoFont.Fonts, PickerFunc:Board
      .UpdateGameInfoFont)), forKey:Key)

// add the move list font type picker to the screen dictionary //
Key = "MoveListFontType"; Objs.updateValue(Obj(Board.MoveListFont.Inf
      .Vals(Key).VI, InfPick:ScreenInfo(Key, SubWindow:"PrefsFont", XP:
      427, YP:-265, Width:195, Height:30, Window:Board.PreferencesWindow,
      Items:Board.MoveListFont.Fonts, PickerFunc:Board
      .UpdateMoveListFont)), forKey:Key)

// add the game list font type picker to the screen dictionary //
Key = "GameListFontType"; Objs.updateValue(Obj(Board.GameListFont.Inf
      .Vals(Key).VI, InfPick:ScreenInfo(Key, SubWindow:"PrefsFont", XP:
      427, YP:-110, Width:195, Height:30, Window:Board.PreferencesWindow,
      Items:Board.GameListFont.Fonts, PickerFunc:Board
      .UpdateGameListFont)), forKey:Key)

// add the board labels bold picker to the screen dictionary //
Key = "BoardLabelsBold"; Objs.updateValue(Obj(BoolTo().Num(BPComp
      .BoardLabelsFont.Inf.Vals(Key).VB), InfPick:ScreenInfo(Key,
      SubWindow:"PrefsFont", XP:280, YP:-575, Width:100, Height:30,
      Window:Board.PreferencesWindow, Items:BPComp.BoardLabelsFont
      .Weights, PickerFunc:Board.UpdateBoardLabelsFont)), forKey:Key)

// add the game info bold picker to the screen dictionary //
Key = "GameInfoBold"; Objs.updateValue(Obj(BoolTo().Num(Board.GameInfoFont
      .Inf.Vals(Key).VB), InfPick:ScreenInfo(Key, SubWindow:"PrefsFont",
      XP:280, YP:-420, Width:100, Height:30, Window:Board
      .PreferencesWindow, Items:Board.GameInfoFont.Weights, PickerFunc:
      Board.UpdateGameInfoFont)), forKey:Key)

// add the move list bold picker to the screen dictionary //
Key = "MoveListBold"; Objs.updateValue(Obj(BoolTo().Num(Board
      .MoveListFont.Inf.Vals(Key).VB), InfPick:ScreenInfo(Key, SubWindow:
      "PrefsFont", XP:280, YP:-265, Width:100, Height:30, Window:Board
      .PreferencesWindow, Items:Board.MoveListFont.Weights, PickerFunc:
      Board.UpdateMoveListFont)), forKey:Key)

// add the game list bold picker to the screen dictionary //
Key = "GameListBold"; Objs.updateValue(Obj(BoolTo().Num(Board.GameListFont
      .Inf.Vals(Key).VB), InfPick:ScreenInfo(Key, SubWindow:"PrefsFont",
      XP:280, YP:-110, Width:100, Height:30, Window:Board
      .PreferencesWindow, Items:Board.GameListFont.Weights, PickerFunc:
      Board.UpdateGameListFont)), forKey:Key)

// add the board labels sizes picker to the screen dictionary //
Key = "BoardLabelsSizes"; Objs.updateValue(Obj(BPComp.BoardLabelsFont.Inf
      .Vals(Key).VI, InfPick:ScreenInfo(Key, SubWindow:"PrefsFont", XP:
      180, YP:-575, Width:100, Height:30, Window:Board.PreferencesWindow,
      Items:BPComp.BoardLabelsFont.Sizes, PickerFunc:Board
      .UpdateBoardLabelsFont)), forKey:Key)

// add the game info sizes picker to the screen dictionary //
Key = "GameInfoSizes"; Objs.updateValue(Obj(Board.GameInfoFont.Inf
      .Vals(Key).VI, InfPick:ScreenInfo(Key, SubWindow:"PrefsFont", XP:
      180, YP:-420, Width:100, Height:30, Window:Board.PreferencesWindow,
      Items:Board.GameInfoFont.Sizes, PickerFunc:Board
      .UpdateGameInfoFont)), forKey:Key)

// add the move list sizes picker to the view dictionary //
Key = "MoveListSizes"; Objs.updateValue(Obj(Board.MoveListFont.Inf
      .Vals(Key).VI, InfPick:ScreenInfo(Key, SubWindow:"PrefsFont", XP:
      180, YP:-265, Width:100, Height:30, Window:Board.PreferencesWindow,
      Items:Board.MoveListFont.Sizes, PickerFunc:Board
      .UpdateMoveListFont)), forKey:Key)
```

```
      // add the game list sizes picker to the view dictionary //
      Key = "GameListSizes"; Objs.updateValue(Obj(Board.GameListFont.Inf
            .Vals(Key).VI, InfPick:ScreenInfo(Key, SubWindow:"PrefsFont", XP:
            180, YP:-110, Width:100, Height:30, Window:Board.PreferencesWindow,
            Items:Board.GameListFont.Sizes, PickerFunc:Board
            .UpdateGameListFont)), forKey:Key)

      // add the board labels title text to the screen dictionary //
      Key = "BoardLabelsFontTitle"; Objs.updateValue(Obj(InfText:
            ScreenInfo("Board Labels", SubWindow:"PrefsFont", XP:330, YP:-600,
            Width:200, Height:30, Window:Board.PreferencesWindow)), forKey:Key)

      // add the game info title text to the screen dictionary //
      Key = "GameInfoFontTitle"; Objs.updateValue(Obj(InfText:
            ScreenInfo("Game Info", SubWindow:"PrefsFont", XP:330, YP:-445,
            Width:200, Height:30, Window:Board.PreferencesWindow)), forKey:Key)

      // add the move list title text to the screen dictionary //
      Key = "MoveListFontTitle"; Objs.updateValue(Obj(InfText:
            ScreenInfo("Move List", SubWindow:"PrefsFont", XP:330, YP:-290,
            Width:200, Height:30, Window:Board.PreferencesWindow)), forKey:Key)

      // add the game list title text to the screen dictionary //
      Key = "GameListFontTitle"; Objs.updateValue(Obj(InfText:
            ScreenInfo("Game List", SubWindow:"PrefsFont", XP:330, YP:-135,
            Width:200, Height:30, Window:Board.PreferencesWindow)), forKey:Key)

      // add the event entered text field box to the screen dictionary //
      Key = "EventTitle"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("Event", Placeholder:"", Key:Key, XP:270, YP:-610,
            Width:500, Height:30, Window:Board.EnterInfoWindow, EnteredFunc:
            Attributes.UpdateEnteredText)), forKey:Key)

      // add the site location text field box to the screen dictionary //
      Key = "SiteLocation"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("Site", Placeholder:"", Key:Key, XP:270, YP:-575,
            Width:500, Height:30, Window:Board.EnterInfoWindow, EnteredFunc:
            Attributes.UpdateEnteredText)), forKey:Key)

      // add the date game played text to the screen dictionary //
      Key = "DateGamePlayedText"; Objs.updateValue(Obj(InfText:
            ScreenInfo("Date", SubWindow:"", XP:99, YP:-540, Width:50, Height:
            30, Window:Board.EnterInfoWindow)), forKey:Key)

      // add the date game played date picker to the screen dictionary //
      Key = "DateGamePlayed"; Objs.updateValue(Obj(Attributes.GetGameDate(Key),
            InfDte:ScreenInfo("Date", Key:Key, XP:174, YP:-540, Width:100,
            Height:30, Window:Board.EnterInfoWindow, DateFunc:Attributes
            .UpdateGameDate)), forKey:Key)

      // add the round number text field box to the screen dictionary //
      Key = "RoundNumber"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("Round", Placeholder:"", Key:Key, XP:70, YP:-505,
            Width:100, Height:30, Window:Board.EnterInfoWindow, EnteredFunc:
            Attributes.UpdateEnteredText)), forKey:Key)

      // add the white name text field box to the screen dictionary //
      Key = "WhiteName"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("White", Placeholder:"", Key:Key, XP:270, YP:-470,
            Width:500, Height:30, Window:Board.EnterInfoWindow, EnteredFunc:
            Attributes.UpdateEnteredText)), forKey:Key)

      // add the black name text field box to the screen dictionary //
      Key = "BlackName"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("Black", Placeholder:"", Key:Key, XP:270, YP:-435,
            Width:500, Height:30, Window:Board.EnterInfoWindow, EnteredFunc:
            Attributes.UpdateEnteredText)), forKey:Key)

      // add the end result text to the screen dictionary //
      Key = "EndResultText"; Objs.updateValue(Obj(InfText:ScreenInfo("Result",
            SubWindow:"", XP:95, YP:-400, Width:50, Height:30, Window:Board
            .EnterInfoWindow)), forKey:Key)

      // add the end result picker view to the screen dictionary //
      Key = "EndResult"; Objs.updateValue(Obj(Attributes.GetEndResult(Key),
            InfPick:ScreenInfo(Key, SubWindow:"", XP:179, YP:-400, Width:125,
            Height:30, Window:Board.EnterInfoWindow, Items:Attributes
            .EndResults, PickerFunc:Attributes.UpdateEndResult)), forKey:Key)

      // add the time control text field box to the screen dictionary //
      Key = "TimeControl"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("Time Control", Placeholder:"", Key:Key, XP:270,
            YP:-365, Width:500, Height:30, Window:Board.EnterInfoWindow,
            EnteredFunc:Attributes.UpdateEnteredText)), forKey:Key)

      // add the white elo text field box to the screen dictionary //
      Key = "WhiteElo"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("White Elo", Placeholder:"", Key:Key, XP:70, YP:
            -330, Width:100, Height:30, Window:Board.EnterInfoWindow,
            EnteredFunc:Attributes.UpdateEnteredText)), forKey:Key)

      // add the black elo text field box to the screen dictionary //
      Key = "BlackElo"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("Black Elo", Placeholder:"", Key:Key, XP:70, YP:
            -295, Width:100, Height:30, Window:Board.EnterInfoWindow,
            EnteredFunc:Attributes.UpdateEnteredText)), forKey:Key)

      // add the white uscf text field box to the screen dictionary //
      Key = "WhiteUSCF"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("White USCF", Placeholder:"", Key:Key, XP:70, YP:
            -260, Width:100, Height:30, Window:Board.EnterInfoWindow,
            EnteredFunc:Attributes.UpdateEnteredText)), forKey:Key)

      // add the black uscf text field box to the screen dictionary //
      Key = "BlackUSCF"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("Black USCF", Placeholder:"", Key:Key, XP:70, YP:
            -225, Width:100, Height:30, Window:Board.EnterInfoWindow,
            EnteredFunc:Attributes.UpdateEnteredText)), forKey:Key)

      // add the annotator text field box to the screen dictionary //
      Key = "Annotator"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("Annotator", Placeholder:"", Key:Key, XP:270, YP:
            -190, Width:500, Height:30, Window:Board.EnterInfoWindow,
            EnteredFunc:Attributes.UpdateEnteredText)), forKey:Key)

      // add the eco text field box to the screen dictionary //
      Key = "ECO"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS, InfEnt:
            ScreenInfo("ECO", Placeholder:"", Key:Key, XP:70, YP:-155, Width:
            100, Height:30, Window:Board.EnterInfoWindow, EnteredFunc:Attributes
            .UpdateEnteredText)), forKey:Key)

      // add the opening text field box to the screen dictionary //
      Key = "Opening"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS, InfEnt:
            ScreenInfo("Opening", Placeholder:"", Key:Key, XP:270, YP:-120,
            Width:500, Height:30, Window:Board.EnterInfoWindow, EnteredFunc:
            Attributes.UpdateEnteredText)), forKey:Key)

      // add the section text field box to the screen dictionary //
      Key = "Section"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS, InfEnt:
            ScreenInfo("Section", Placeholder:"", Key:Key, XP:270, YP:-85,
            Width:500, Height:30, Window:Board.EnterInfoWindow, EnteredFunc:
            Attributes.UpdateEnteredText)), forKey:Key)

      // add the board number text field box to the screen dictionary //
      Key = "BoardNumber"; Objs.updateValue(Obj(Attributes.Inf.Vals(Key).VS,
            InfEnt:ScreenInfo("Board", Placeholder:"", Key:Key, XP:70, YP:-50,
            Width:100, Height:30, Window:Board.EnterInfoWindow, EnteredFunc:
            Attributes.UpdateEnteredText)), forKey:Key)

      // add the game list table view to the screen dictionary //
      Key = "GameListTableView"; Objs.updateValue(Obj(InfLst:ScreenInfo(Key,
            Window:Board.GameListWindow, RefreshOrLoadFunc:Board
            .RefreshOrLoadFromList)), forKey:Key)

      // add the list toolbar to the game list dictionary //
      Key = "ListToolbar"; Objs.updateValue(Obj(0, InfBar:ScreenInfo(Board
            .GameListWindow, Selectable:false, BarButtons:ListBar,
            ToolbarFunc:Board.GameListAction)), forKey:Key)

      // add the preferences toolbar to the screen dictionary //
      Key = "PrefsToolbar"; Objs.updateValue(Obj(Board.Inf
            .Vals("PreferencesSelection").VI, InfBar:ScreenInfo(Board
            .PreferencesWindow, Selectable:true, BarButtons:PrefsBar,
            ToolbarFunc:Board.PreferencesAction)), forKey:Key)

      // add the enter game info pgn tabs toolbar to the screen dictionary //
      Key = "TagsToolbar"; Objs.updateValue(Obj(1, InfBar:ScreenInfo(Board
            .EnterInfoWindow, Selectable:false, BarButtons:TagsBar,
            ToolbarFunc:Board.ExitInfoWindow)), forKey:Key)
   }
}
```