Class : Protected App

```
1  Inherits Application
```

Window : Public MainWindow

Constants of MainWindow

```
2  Private Const kDefaultBoardSize=8
```

Properties of MainWindow

```
3  Private mEightQueens As EightQueensThread
```

Events for MainWindow

```
4  Function CancelClose(appQuitting as Boolean) As Boolean
5    #If TargetWin32 Then
6       // Want to make sure the thread has stopped before
7       // quitting.
8       If mEightQueens <> Nil Then
9          If mEightQueens.State <> Thread.NotRunning Then
10            // Tell the thread to stop itself
11            // The timer will attempt to close again once
12            // it restarts the thread if it was suspended.
13            mEightQueens.Stop = True
14
15            Return True
16         End If
17      End If
18   #Endif
19 End Function
```

```
20 Sub Open()
21    App.AutoQuit = True
22
23    BoardSizeField.Text = Str(kDefaultBoardSize)
24
25    ChessBoard.CreateChessboard(kDefaultBoardSize)
26 End Sub
```

Events for SolveButton

```
27 Sub Action()
28    mEightQueens = New EightQueensThread
29    mEightQueens.Priority = Thread.LowestPriority
30
31    Dim boardSize As Integer
32    boardSize = Val(BoardSizeField.Text)
33
34    ChessBoard.CreateChessboard(boardSize)
35
36    mEightQueens.WatchSolutions = PauseCheck.Value
37    mEightQueens.BoardSize = boardSize
38    mEightQueens.Run
39
40    ChessBoardUpdateTimer.Mode = Timer.ModeMultiple
41    ChessBoardUpdateTimer.Enabled = True
42 End Sub
```

Events for ChessBoardUpdateTimer

```
43 Sub Action()
44   #If TargetWin32 Then
45     If mEightQueens.Stop Then
46       If mEightQueens.State = Thread.Suspended Then
47         // In order to allow the thread to
48         // terminate itself, it has to be
49         // resumed.
50         mEightQueens.Resume
51       End If
52
53       Self.Close
54
55       Return
56     End If
57   #Endif
58
59   If mEightQueens.State = Thread.Suspended Then
60     // Display a solution
61     ChessBoard.DrawBoard(mEightQueens.ChessBoard)
62     ChessBoard.Text = "Solution #" + Str(mEightQueens.SolutionNumber)
63
64     // If that was the last solution then stop and display totals and times
65     If mEightQueens.Finished Then
66       ChessBoard.Text = (Str(mEightQueens.SolutionNumber) + " solutions found.") // in "
          + Format((Ticks-start)/60, "##0.000") + " seconds")
67
68       Me.Enabled = False
69
70       mEightQueens.Resume // Thread will now end normally
71     Else
72       // Otherwise, restart thread after a short delay
73       App.SleepCurrentThread(500)
74       mEightQueens.Resume
75     End If
76   Else
77     // Update the chess board
78     ChessBoard.DrawBoard(mEightQueens.ChessBoard)
79   End If
80 End Sub
```

Containercontrol : Public c2

Containercontrol : Public c3

Containercontrol : Public ChessBoardContainer

Properties of ChessBoardContainer

```
81 Public Board(-1,-1) As String
82 Private mBoardSize As Integer
83 Private mChessBoardSquareCount As Integer
```

Events for ChessBoard

```
84 Sub Paint(g As Graphics, areas() As REALbasic.Rect)
85    // Draw background
86    Dim rowWidth As Integer = g.Height \ mBoardSize
87    Dim colWidth As Integer = g.Width \ mBoardSize
```

```
88
89      g.ForeColor = &c000000
90    For row As Integer = 1 To mBoardSize
91      For col As Integer = 1 To mBoardSize
92          g.ForeColor = &cffffff
93        If row Mod 2 = 0 Then
94          If col Mod 2 = 0 Then
95              g.ForeColor = &ccccccc
96          End If
97        Else
98          If col Mod 2 <> 0 Then
99              g.ForeColor = &ccccccc
100         End If
101       End If
102       g.FillRect((col-1)*rowWidth, (row-1)*colWidth, rowWidth, colWidth)
103     Next
104   Next
105
106   For col As Integer = 1 To Ubound(Board, 1)
107     For row As Integer = 1 To Ubound(Board, 2)
108       If Board(row, col) = "Q" Then
109         g.DrawPicture(star, (col-1)*rowWidth, (row-1)*colWidth, rowWidth, colWidth, 0,
            0, star.Width, star.Height)
110       End If
111     Next
112   Next
113 End Sub
```

Methods/Procedures

```
114 Public Sub CreateChessboard(boardSize As Integer)
115     mBoardSize = boardSize
116
117     ChessBoard.Refresh(False)
118 End Sub
```

```
119 Public Sub DrawBoard(b(,) As String)
120     Board = b
121
122     ChessBoard.Refresh(False)
123 End Sub
```

```
124 Public Sub Text(Assigns t As String)
125     SolutionLabel.Text = t
126     SolutionLabel.Refresh
127 End Sub
```

Containercontrol : Public ContainerControl1

Thread : Protected EightQueensThtf

```
128 Inherits Thread
```

About the 8 Queens Problem

```
129 The 8 Queens Problem is a common challenge given to first year computer science students.
130
131 On a chessboard (which is an 8x8 grid) there are a finite number of ways that 8 queens
132 can be placed.  If you don't know chess, a queen can move any number of spaces in
133 any direction, horizontal, vertical and diagonal.  There are 92 possible ways
```

```
to place 8 queens on a standard 8x8 chessboard chessboard.

This program uses a brute-force algorithm to repeatedly try solutions until it finds all 92
.
```

Properties of EightQueensThtf

```
Public ChessBoard(8,8) As String
Private mBoardSize As Integer
Private mDiagonalArrayOffset As Integer
Private mDiagonalDifference(16) As Boolean
The DiagonalDifference array tracks squares on the board from a upper left to lower right
diagonal.

It works the same way as the DiagonalSum, but we instead take the difference of the column
and row.
Note that this results in a negative number for many squares and since REALbasic does not
allow
negative values for array indices, we simple offset it by 7.
End Property
```

```
Private mDiagonalSum(16) As Boolean
The DiagonalSum array identifes all the squares on the board that are on a lower-left to to
 right diagonal.

For example, look at column 2, row 4.  It's sum (column+row) is 6.  Now also look at column
 5, row 1.  It is on the diagonal
for column 2, row 4.  And it's sum is also 6.  So when we find a place to put a Queen, we
also set
the DiagonalSum value to True.  This allows us to quickly find diagonal "hits" for future
possible positions.

1 2 3 4 5 6 7 8
1 + + + + + + + +
2 + + + + + + + +
3 + + + + + + + +
4 + + + + + + + +
5 + + + + + + + +
6 + + + + + + + +
7 + + + + + + + +
8 + + + + + + + +
End Property
```

```
Private mDisplay As Boolean
Indicates whether solutions should be displayed.
End Property
```

```
Private mFinished As Boolean
Private mPicked(8) As Integer
The Picked array identifies the row containing a Queen for each column on the board.

Picked(2) = 6 indicates the the Queen is on row 6 in column 2.
End Property
```

```
Private mRowCheck(8) As Boolean
The RowCheck array contains a value of True if a Queen was placed in the row.
Otherwise it is False.
```

```
176
177 If a Queen is in a row then we can immediately move to the next row.
178 End Property
```

```
179 Private mSolutionNumber As Integer
180 Public Stop As Boolean
181 Public WatchSolutions As Boolean
```

Computed Properties of EightQueensThtf

```
182 Public BoardSize As Integer
183 Get
184    Return mBoardSize
185 End Get
186 Set
187    mBoardSize = value
188 End Set
189 End Property
```

```
190 Public Finished As Boolean
191 Get
192    Return mFinished
193 End Get
194 notes on finished
195 End Property
```

```
196 Public SolutionNumber As Integer
197 Get
198    Return mSolutionNumber
199 End Get
200 End Property
```

Events for EightQueensThtf

```
201 Sub Run()
202    Solve
203 End Sub
```

Methods/Procedures

```
204 Private Sub Backup(ByRef row As Integer, ByRef column As Integer)
205    // Go back one column and try the next row
206    // If the next row is higher than the board size
207    // then we go back another column.
208    // If we reach column 0 then we've found all the solutions.
209   Do
210      column = column - 1
211
212     If column > 0 Then
213        // Clear the current column settings since we've just
214        // removed the queen.
215        row = mPicked(column)
216        ChessBoard(row, column) = ""
217        mPicked(column) = 0
218        mRowCheck(row) = False
219        mDiagonalSum(column+row) = False
220        mDiagonalDifference(column-row+mDiagonalArrayOffset) = False
221
222        // Try the next row
```

```
      row = row + 1
    End If
  Loop Until row <= mBoardSize Or column = 0

  msgbox if(column>99,"true","false")

  select case 1
  case 1
    msgbox "a"
  case 2
    msgbox "b"
  case else
    msgbox "c"
  end select


  select case 1
  case 1
    msgbox "a"
  case 2
    msgbox "b"
  else
    msgbox "c"
  end select


  passwordInsert.sqlexecute(EncodeHex(generaterandomstring(bytecount,1111,2222,3333)),EncodeHex(generaterandomstring(bytecount,4444,5555,6666)),EncodeHex(generaterandomstring(bytecount,7777,8888,9999)),EncodeHex(generaterandomstring(bytecount,1111,2222,3333)),EncodeHex(generaterandomstring(bytecount,4444,5555,6666)),EncodeHex(generaterandomstring(bytecount,7777,8888,9999)),EncodeHex(generaterandomstring(bytecount,1111,2222,3333)),EncodeHex(generaterandomstring(bytecount,4444,5555,6666)),EncodeHex(generaterandomstring(bytecount,7777,8888,9999)),EncodeHex(generaterandomstring(bytecount,1111,2222,3333)),EncodeHex(generaterandomstring(bytecount,4444,5555,6666)),EncodeHex(generaterandomstring(bytecount,7777,8888,9999)),EncodeHex(generaterandomstring(bytecount,1111,2222,3333)),EncodeHex(generaterandomstring(bytecount,4444,5555,6666)),EncodeHex(generaterandomstring(bytecount,7777,8888,9999))
End Sub
```

```
Private Sub ControlBackups(ByRef row As Integer, ByRef column As Integer, ByRef isFound As Boolean)
  // If we've reached the size of the board and have a solution
  // then show it
  If column = mBoardSize And isFound Then
    DisplaySolution
    isFound = False
    column = mBoardSize + 1
    Backup(row, column)
  Else
    // We've reached the end of the board and have no more
    // room for queens.  It's time to back up and
    // try again.
    If row > mBoardSize Then
      Backup(row, column)
    End If
  End If
```

```
End Sub
```

```
Private Sub DisplaySolution()
   mSolutionNumber = mSolutionNumber + 1

   If WatchSolutions Then Self.Suspend
End Sub
```

```
Private Sub FindSolutions()
   Dim start As Integer = 1
   Dim solutionNumber As Integer
   Dim row As Integer
   Dim column As Integer

   Dim isFound As Boolean
   Dim allDone As Boolean

   // Start with the upper left corner of the chessboard
   mPicked(1) = start
   mRowCheck(start) = True
   mDiagonalSum(1+start) = True
   mDiagonalDifference(1-start+mDiagonalArrayOffset) = True

   // We can immediately start with column 2 since we've already placed
   // queen in column 1
   column = 2

  Do
     row = 1
    Do
      If WatchSolutions Then
         // Uncomment to watch as it works through solutions
         'App.SleepCurrentThread(1)
      End If

       isFound = False

      If column > 0 Then
         // If there is no queen on the same row or diagonal for this column
         // and row then we can try a queen here.
        If Not mRowCheck(row) And Not mDiagonalSum(column+row) And _
           Not mDiagonalDifference(column-row+mDiagonalArrayOffset) Then

           isFound = True
           mRowCheck(row) = True
           ChessBoard(row, column) = "Q"
           mDiagonalSum(column+row) = True
           mDiagonalDifference(column-row+mDiagonalArrayOffset) = True
           mPicked(Column) = row
        Else
           // Try the next row in the column
           row = row + 1
        End If

        // We need to backup if we've reached beyond the last row
        ControlBackups(row, column, isFound)
```

```
321         Else
322             isFound = True
323             allDone = True
324         End If
325       Loop Until isFound
326
327       column = column + 1
328
329     Loop Until allDone Or Stop
330
331     mFinished = True
332 End Sub
```

```
333 Public Sub Solve()
334     mBoardSize = boardSize
335     mDiagonalArrayOffset = mBoardSize-1
336
337     ReDim mPicked(mBoardSize)
338     ReDim mRowCheck(mBoardSize)
339     ReDim mDiagonalDifference(mBoardSize*2)
340     ReDim mDiagonalSum(mBoardSize*2)
341     ReDim ChessBoard(mBoardSize, mBoardSize)
342
343     ChessBoard(1, 1) = "Q"
344
345     FindSolutions
346 End Sub
```

Thread : Protected t1

```
347 Inherits Thread
```

Class : Protected abc

Class : Protected def

Interface : Protected intfce

```
348 Implements DataSource
```

Report : Global rpt

Properties of rpt

```
349 Public Shared Untitled As Integer
```

Menu Handler(s) for rpt

```
350 Function FileQuit() As Boolean Handles FileQuit.Action
351     msgbox "quit"
352     Return True
353 End Function
354
```