AUTODESK UNIVERSITY 2005
Walt Disney World Swan and Dolphin Resort
Orlando, Florida

Programming Tables to Do More

R. Robert Bell - MW Consulting Engineers

CP35-1 Tables are a welcome enhancement to AutoCAD. Do you wish that they would do more? You will be introduced to some functions created in VBA that make tables even more useful. Visual LISP functions will also be demonstrated.

About the Speaker:

Robert is the network administrator for MW Consulting Engineers, a consulting firm in Spokane, Washington, where he is responsible for the network and all AutoCAD customization. Robert has been writing AutoLISP code since AutoCAD v2.5, and VBA since it was introduced in Release 14. He has customized applications for the electrical/lighting, plumbing/piping, and HVAC disciplines. Robert has also developed applications for AutoCAD as a consultant. He is on the Board of Directors for AUGI and is active on Autodesk discussion groups.

RobertB@MWEngineers.com

Tables were a new feature to AutoCAD 2005. They have matured in AutoCAD 2006. However, there are tasks that we would like tables to do, that just are not possible in the normal user interface. Autodesk did provide an extensive programming interface for tables. This means that complicated tasks can be automated so that the user does not need to perform needlessly lengthy tasks.

This class will introduce you to the ActiveX interface for tables. This will be done by example, demonstrating programs that automate complex tasks such as:

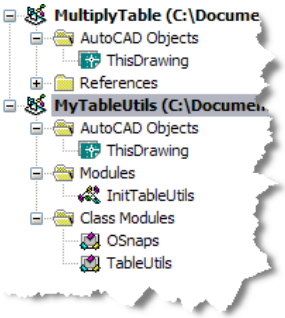
- Coping a selected range from one table to another
- Inserting a row, copying the format of the cells above
- Copy a table's current format to a new table style
- Setting an entire table to use a background color

Visual Basic will be the language used during the presentation, yet Visual LISP will be discussed and available for download.

This small Visual LISP function will be used during the class to run the Test procedures in the sample VBA projects.

```
(defun C:T () (command "._-VBARun" "Test"))
```

Table utility functions



Many of the samples will use certain functions repeatedly. These functions have been placed in a separate VBA project in a class module. This permits the code to be maintained in one place and referenced by multiple projects. A standard module in the MyTableUtility project provides a means to initialize an instance of the TableUtility object defined by the class module. There is also a separate private class module to support turning off running OSnaps during a table selection.

The following code should be placed in a Private class module named "OSnaps".

```
Option Explicit

Private orgOSnaps As Long

Private Sub Class_Initialize()
  orgOSnaps = ThisDrawing.GetVariable("OSMode")
  ThisDrawing.SetVariable "OSMode", orgOSnaps Or 16384 'turn running OSnaps off
End Sub

Private Sub Class_Terminate()
  ThisDrawing.SetVariable "OSMode", orgOSnaps
End Sub
```

The following code should be placed in a PublicNotCreatable class module named "TableUtils".

```
Option Explicit

Private Function KillCommand(Prompt As String) As String
  Dim cmdLen As Long
  cmdLen = Len("Command: ")
  Dim pmtLen As Long
  pmtLen = cmdLen - Len(Prompt)
  If pmtLen < 0 Then pmtLen = 0
  KillCommand = String(cmdLen, Chr(8)) & Prompt & String(pmtLen, Chr(32))
End Function
```

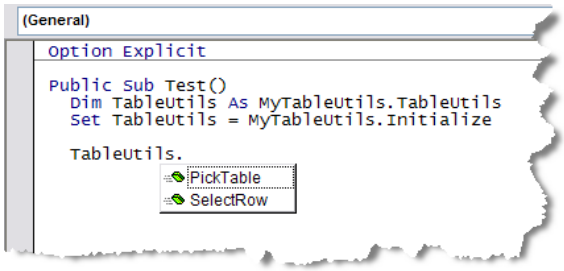
```
Public Function PickTable(Prompt As String) As Collection
  Dim acUtil As AcadUtility
  Set acUtil = ThisDrawing.Utility
  Set PickTable = New Collection
  Dim OSnapsOff As OSnaps
  Set OSnapsOff = New OSnaps
  Dim pickObj As AcadEntity
  Dim pickPt
  On Error GoTo Trap

GetPick:
  ThisDrawing.SetVariable "ErrNo", 0
  acUtil.GetEntity pickObj, pickPt, KillCommand(Prompt)

  If TypeOf pickObj Is AcadTable Then
    PickTable.Add pickObj
    PickTable.Add pickPt
  Else
    acUtil.Prompt KillCommand("0 found") & vbCrLf
    GoTo GetPick
  End If
  Set OSnapsOff = Nothing
Exit Function

Trap:
  Select Case Err.Number
    Case -2147352567
      Select Case ThisDrawing.GetVariable("ErrNo")
        Case 7 'misssed pick
          Resume
        Case 52 '<enter> or <esc> will goto end
        Case Else 'goto end
      End Select
    Case Else
      MsgBox Err.Number & vbCrLf & Err.Description, vbCritical, "PickTable"
  End Select
  Err.Clear
  Set PickTable = Nothing 'force a return of Nothing for calling procedures
End Function
```

```
Public Function SelectRow(Prompt As String) As Collection
  Dim TableInfo As Collection
  Set TableInfo = PickTable(Prompt)

  If Not (TableInfo Is Nothing) Then
    Dim myTable As AcadTable
    Set myTable = TableInfo(1)

    Dim pickPt As Variant
    pickPt = TableInfo(2)

    Set SelectRow = New Collection
    SelectRow.Add myTable

    Dim vecNorm(0 To 2) As Double
    vecNorm(0) = 0#: vecNorm(1) = 0#: vecNorm(2) = 1#

    Dim row As Long, column As Long
    Dim inTable As Boolean
    inTable = myTable.HitTest(pickPt, vecNorm, row, column)
    If Not (inTable) Then row = myTable.Rows - 1
    SelectRow.Add row
    SelectRow.Add column
  End If
End Function
```

The following code should be placed in a standard module named "InitTableUtils".

```
Option Explicit

Public Function Initialize() As TableUtils
  Set Initialize = New TableUtils
End Function
```

Once the external VBA project is saved, you can reference the .dvb file itself in another VBA project, just as you would with an ActiveX .dll or .tlb file. You then create an instance of the object by declaring a variable to the [ProjectName].[ClassName] and then calling the initialization procedure using [ProjectName].[FunctionName] as shown here:




```
Dim TableUtils As MyTableUtils.TableUtils
Set TableUtils = MyTableUtils.Initialize
```

Copy a range of cells from one table to another

Copying data from one table to another is a chore. The following project, although lengthy, makes it very easy to copy the data from one table and insert that data into another table. The user is asked to select a range of cells from one table ("GetTableData"). The user then needs to select the cell where the data is supposed to be inserted ("PlaceInDestination"). New rows are added to the destination table and the data is placed. If the user selects the table on the outside of a cell all the data is inserted below the last row of the table. If the user selects a cell too far to the right to accommodate all the data a warning is given.

```
Private Function SelectTable(FirstPoint As Variant, OtherPoint As Variant) As AcadTable
  ThisDrawing.SetVariable "NoMutt", 1
  Dim mySS As AcadSelectionSet
  Set mySS = ThisDrawing.SelectionSets.Add("Test")
  mySS.Select acSelectionSetCrossing, FirstPoint, OtherPoint
  If mySS.Count > 0 Then
    If TypeOf mySS.Item(0) Is AcadTable Then Set SelectTable = mySS.Item(0)
  End If
  mySS.Delete
  ThisDrawing.SetVariable "NoMutt", 0
End Function
```

```
Private Function GetCorner(StartPrompt As String, EndPrompt As String) As Collection
  Dim orgOSnaps As Long
  orgOSnaps = ThisDrawing.GetVariable("OSMode")
  ThisDrawing.SetVariable "OSMode", orgOSnaps Or 16384 'turn running OSnaps off

  Dim pickPt1 As Variant
  pickPt1 = GetPoint(StartPrompt)
  If HasElements(pickPt1) Then
    On Error GoTo Trap
    Dim pickPt2 As Variant
    pickPt2 = ThisDrawing.Utility.GetCorner(pickPt1, KillCommand(EndPrompt))
    If HasElements(pickPt2) Then
      Set GetCorner = New Collection
      GetCorner.Add pickPt1
      GetCorner.Add pickPt2
    End If
  End If
  ThisDrawing.SetVariable "OSMode", orgOSnaps
Exit Function
```

```
Trap:
  Select Case Err.Number
    Case -2147352567, -2145320928 '<enter> or <esc> will goto end
    Case Else
      MsgBox Err.Number & vbCrLf & Err.Description, vbCritical, "GetCorner"
  End Select
  Err.Clear
  ThisDrawing.SetVariable "OSMode", orgOSnaps
End Function
```

```
Private Function GetPoint(Prompt As String) As Variant
  On Error GoTo Trap
  Dim pickPt As Variant
  pickPt = ThisDrawing.Utility.GetPoint(Prompt:=KillCommand(Prompt))
  GetPoint = pickPt
Exit Function

Trap:
  Select Case Err.Number
    Case -2147352567, -2145320928 '<enter> or <esc> will goto end
    Case Else
      MsgBox Err.Number & vbCrLf & Err.Description, vbCritical, "GetPoint"
  End Select
  Err.Clear
End Function
```

```
Function HasElements(ByVal ArrayToCheck As Variant) As Boolean
  On Error Resume Next
  HasElements = (LBound(ArrayToCheck) <= UBound(ArrayToCheck))
  Err.Clear
End Function
```

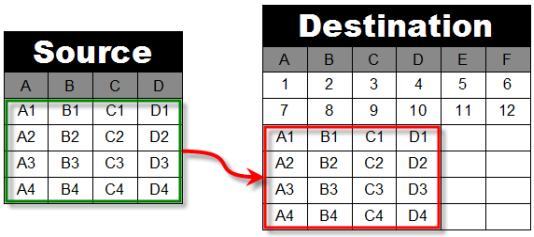
```
Private Function KillCommand(Prompt As String) As String
  Dim cmdLen As Long
  cmdLen = Len("Command: ")
  Dim pmtLen As Long
  pmtLen = cmdLen - Len(Prompt)
  If pmtLen < 0 Then pmtLen = 0
  KillCommand = String(cmdLen, Chr(8)) & Prompt & String(pmtLen, Chr(32))
End Function
```

```
Private Function GetTableData() As Variant
  Dim pmtStart As String, pmtEnd As String
  pmtStart = "Select starting cell: "
  pmtEnd = "Select ending cell: "

  Dim pickPts As Collection
  Set pickPts = GetCorner(pmtStart, pmtEnd)
  If Not (pickPts Is Nothing) Then
    Dim myTable As AcadTable
    Set myTable = SelectTable(pickPts(1), pickPts(2))
    If Not (myTable Is Nothing) Then
      Dim vecNorm(0 To 2) As Double
      vecNorm(0) = 0#: vecNorm(1) = 0#: vecNorm(2) = 1#
      Dim rowStart As Long, colStart As Long, rowEnd As Long, colEnd As Long
      myTable.SelectSubRegion _
        pickPts(1), pickPts(2), _
        vecNorm, vecNorm, _
        acTableSelectCrossing, False, _
        rowStart, rowEnd, _
        colStart, colEnd
```

```
      Dim rowCount As Long
      rowCount = rowEnd - rowStart
      Dim colCount As Long
      colCount = colEnd - colStart
      Dim TableData As Variant
      ReDim TableData(0 To rowCount, 0 To colCount)
      Dim rowCounter As Long
      Dim colCounter As Long
      For rowCounter = 0 To rowCount
        For colCounter = 0 To colCount
          TableData(rowCounter, colCounter) = _
            myTable.GetText(rowCounter + rowStart, colCounter + colStart)
        Next colCounter
      Next rowCounter
      GetTableData = TableData
    Else
      ThisDrawing.Utility.Prompt KillCommand("No table was selected.")
    End If
  End If
End Function
```

```
Private Sub PlaceInDestination(SourceTableData As Variant)
  Dim acUtil As AcadUtility
  Set acUtil = ThisDrawing.Utility

  Dim TableUtils As MyTableUtils.TableUtils
  Set TableUtils = MyTableUtils.Initialize

  Dim tableInfo As Collection
  Set tableInfo = TableUtils.SelectRow("Select row: ")
  Set TableUtils = Nothing

  If Not (tableInfo Is Nothing) Then
    Dim rowsCount As Long
    rowsCount = UBound(SourceTableData, 1) + 1
    Dim myTable As AcadTable
    Set myTable = tableInfo(1)
    Dim row As Long
    row = tableInfo(2)
    Dim column As Long
    column = tableInfo(3)

    myTable.InsertRows row + 1, myTable.GetRowHeight(row), rowsCount

    Dim maxCol As Long, srcCol As Long
    maxCol = myTable.Columns - column - 1
    srcCol = UBound(SourceTableData, 2)
    If srcCol > maxCol Then
      srcCol = maxCol
      acUtil.Prompt _
        KillCommand("Destination has fewer available columns, data truncated.")
    End If

    Dim rowCounter As Long, colCounter As Long
    For rowCounter = 0 To rowsCount - 1
      For colCounter = 0 To srcCol
        myTable.SetText _
          rowCounter + row + 1, _
          colCounter + column, _
          SourceTableData(rowCounter, colCounter)
      Next colCounter
    Next rowCounter
  End If
End Sub
```

Here is a procedure that you can use to test the above procedures.




```
Public Sub Test()
  ThisDrawing.Utility.Prompt vbCrLf

  Dim sourceData As Variant
  sourceData = GetTableData
  If HasElements(sourceData) Then PlaceInDestination sourceData
End Sub
```

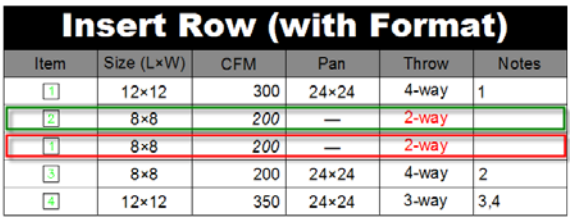
Insert a row copying the format of the cells above

Inserting rows in a formatted table can be an exercise in frustration. At least we are given the ability to inserts rows. However, the inserted cells take on the format of the table style's definition, not the format of the cells from the row where the insert was performed. The Match Cell tool helps, but is a manual tool, not something that is automatic. In addition, the Match Cell tool cannot match formats applied to the MText of a cell. The following code will give you the ability to select a row and have a row inserted below that row, applying the cell's format and its data.

```
Private Sub FormatRow(Table As AcadTable, SourceRow As Long, CopiedRow As Long)
  Dim cellText As String
  With Table
    Dim colCount As Long
    colCount = .Columns

    Dim colCounter As Long
    For colCounter = 0 To colCount - 1
      .SetCellType CopiedRow, colCounter, _
        .GetCellType(SourceRow, colCounter)

      Select Case .GetCellType(SourceRow, colCounter)
        Case acTextCell
          cellText = .GetText(SourceRow, colCounter)
          If cellText <> "" Then .SetText CopiedRow, colCounter, cellText
        Case acBlockCell
          .SetBlockTableRecordId CopiedRow, colCounter, _
            .GetBlockTableRecordId(SourceRow, colCounter), _
            .GetAutoScale(SourceRow, colCounter)
          .SetBlockRotation CopiedRow, colCounter, _
            .GetBlockRotation(SourceRow, colCounter)
          .SetBlockScale CopiedRow, colCounter, _
            .GetBlockScale(SourceRow, colCounter)
      End Select

      .SetCellAlignment CopiedRow, colCounter, _
        .GetCellAlignment(SourceRow, colCounter)
      .SetCellBackgroundColor CopiedRow, colCounter, _
        .GetCellBackgroundColor(SourceRow, colCounter)
      .SetCellBackgroundColorNone CopiedRow, colCounter, _
        .GetCellBackgroundColorNone(SourceRow, colCounter)
      .SetCellContentColor CopiedRow, colCounter, _
        .GetCellContentColor(SourceRow, colCounter)

      .SetCellGridColor CopiedRow, colCounter, acBottomMask, _
        .GetCellGridColor(SourceRow, colCounter, acBottomMask)
      .SetCellGridColor CopiedRow, colCounter, acLeftMask, _
        .GetCellGridColor(SourceRow, colCounter, acLeftMask)
      .SetCellGridColor CopiedRow, colCounter, acRightMask, _
        .GetCellGridColor(SourceRow, colCounter, acRightMask)
      .SetCellGridColor CopiedRow, colCounter, acTopMask, _
        .GetCellGridColor(SourceRow, colCounter, acTopMask)
```

```
        .SetCellGridLineWeight CopiedRow, colCounter, acBottomMask, _
          .GetCellGridLineWeight(SourceRow, colCounter, acBottomMask)
        .SetCellGridLineWeight CopiedRow, colCounter, acLeftMask, _
          .GetCellGridLineWeight(SourceRow, colCounter, acLeftMask)
        .SetCellGridLineWeight CopiedRow, colCounter, acRightMask, _
          .GetCellGridLineWeight(SourceRow, colCounter, acRightMask)
        .SetCellGridLineWeight CopiedRow, colCounter, acTopMask, _
          .GetCellGridLineWeight(SourceRow, colCounter, acTopMask)

        .SetCellGridVisibility CopiedRow, colCounter, acBottomMask, _
          .GetCellGridVisibility(SourceRow, colCounter, acBottomMask)
        .SetCellGridVisibility CopiedRow, colCounter, acLeftMask, _
          .GetCellGridVisibility(SourceRow, colCounter, acLeftMask)
        .SetCellGridVisibility CopiedRow, colCounter, acRightMask, _
          .GetCellGridVisibility(SourceRow, colCounter, acRightMask)
        .SetCellGridVisibility CopiedRow, colCounter, acTopMask, _
          .GetCellGridVisibility(SourceRow, colCounter, acTopMask)

        .SetCellTextHeight CopiedRow, colCounter, _
          .GetCellTextHeight(SourceRow, colCounter)
        .SetCellTextStyle CopiedRow, colCounter, _
          .GetCellTextStyle(SourceRow, colCounter)
        .SetTextRotation CopiedRow, colCounter, _
          .GetTextRotation(SourceRow, colCounter)
      Next colCounter
    End With
  End Sub
```

Here is a procedure that you can use to test the above procedure.




```
  Public Sub Test()
    ThisDrawing.Utility.Prompt vbCrLf

    Dim TableUtils As myTableUtils.TableUtils
    Set TableUtils = myTableUtils.Initialize

    Dim TableInfo As Collection
    Set TableInfo = TableUtils.SelectRow("Select row: ")
    Set TableUtils = Nothing

    If Not (TableInfo Is Nothing) Then
      Dim myTable As AcadTable
      Set myTable = TableInfo(1)

      Dim srcRow As Long
      srcRow = TableInfo(2)

      myTable.InsertRows srcRow + 1, myTable.GetRowHeight(srcRow), 1
      FormatRow myTable, srcRow, srcRow + 1
    End If
  End Sub
```

Copy a table's current format to a new table style

Although creating a table's style is usually done before the creation of the table, at times you may want to copy the format of a table that has been modified to a new table style. The process for transferring the format of a table into a style is not automatic and is therefore error-prone and tedious. This makes it a perfect candidate for automation.

A few moments of perusing the ActiveX object model reveals an interesting dilemma. *There is no Add method for a Table Style!* There is no TableStyles collection. Instead, table styles are defined and stored in a dictionary named ACAD_TableStyle. You can use the AddObject method on the dictionary to add a new table style to the dictionary.

```
Public Function CreateTableStyle(ByVal Name As String, _
                                 ByVal Description As String) As AcadTableStyle
  Dim tblDict As AcadDictionary
  Set tblDict = ThisDrawing.Database.Dictionaries.Item("ACAD_TableStyle")

  Dim myStyle As AcadTableStyle
  Set myStyle = tblDict.AddObject(Name, "AcDbTableStyle")

  myStyle.Description = Description
  Set CreateTableStyle = myStyle
End Function
```

All that remains is to write some code to read a table's format and apply the format to the elements of the table style. The trick is that you need to look at the individual cell properties and not the table's overall properties.

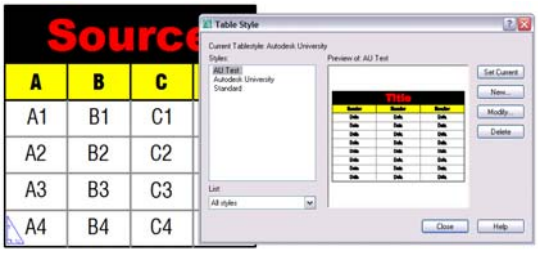
```
Public Function ModifyTableStyle(ByVal Name As String, _
                                 ByVal Description As String, _
                                 ByVal ReadTable As AcadTable) As Boolean
  Dim myStyle As AcadTableStyle
  If IsTableStyleFound(Name) Then
    Set myStyle = GetTableStyle(Name)
  Else
    Set myStyle = CreateTableStyle(Name, Description)
  End If

  With myStyle
    .FlowDirection = ReadTable.FlowDirection
    .HorzCellMargin = ReadTable.HorzCellMargin
    .VertCellMargin = ReadTable.VertCellMargin
    Dim WorkRow As Long
    For WorkRow = 0 To 2
        Select Case ReadTable.GetRowType(WorkRow)
          Case acTitleRow
            .TitleSuppressed = False
            SetRowFormat myStyle, ReadTable, acTitleRow, WorkRow
            .SetGridColor _
              acHorzTop + acHorzBottom + acVertLeft + acVertRight, _
              acTitleRow, _
              ReadTable.GetCellGridColor(WorkRow, 0, acTopMask)
            .SetGridLineWeight _
              acHorzTop + acHorzBottom + acVertLeft + acVertRight, _
              acTitleRow, _
              ReadTable.GetCellGridLineWeight(WorkRow, 0, acTopMask)
            .SetGridVisibility _
              acHorzTop + acHorzBottom + acVertLeft + acVertRight, _
              acTitleRow, _
              ReadTable.GetCellGridVisibility(WorkRow, 0, acTopMask)
          Case acHeaderRow
            .HeaderSuppressed = False
            SetRowFormat myStyle, ReadTable, acHeaderRow, WorkRow
            .SetGridColor _
              acHorzTop + acHorzBottom + acVertLeft + acVertRight, _
              acHeaderRow, _
              ReadTable.GetCellGridColor(WorkRow, 0, acTopMask)
            .SetGridLineWeight _
              acHorzTop + acHorzBottom + acVertLeft + acVertRight, _
              acHeaderRow, _
              ReadTable.GetCellGridLineWeight(WorkRow, 0, acTopMask)
            .SetGridVisibility _
              acHorzTop + acHorzBottom + acVertLeft + acVertRight, _
              acHeaderRow, _
              ReadTable.GetCellGridVisibility(WorkRow, 0, acTopMask)
          Case Else
```

```
            SetRowFormat myStyle, ReadTable, acDataRow, WorkRow
            .SetGridColor acHorzInside + acVertInside, acDataRow, _
              ReadTable.GetCellGridColor(WorkRow, 0, acBottomMask)
            .SetGridLineWeight acHorzInside + acVertInside, acDataRow, _
              ReadTable.GetCellGridLineWeight(WorkRow, 0, acBottomMask)
            .SetGridVisibility acHorzInside + acVertInside, acDataRow, _
              ReadTable.GetCellGridVisibility(WorkRow, 0, acBottomMask)
            Exit For
        End Select
    Next WorkRow
  End With
End Function
```

```
Private Function SetRowFormat(ByRef TableStyle As AcadTableStyle, _
                              ByVal ReadTable As AcadTable, _
                              ByVal RowType As Long, _
                              ByVal WorkRow As Long)
  With TableStyle
    .SetAlignment RowType, _
      ReadTable.GetCellAlignment(WorkRow, 0)
    .SetBackgroundColorNone RowType, _
      ReadTable.GetCellBackgroundColorNone(WorkRow, 0)
    .SetBackgroundColor RowType, _
      ReadTable.GetCellBackgroundColor(WorkRow, 0)
    .SetColor RowType, _
      ReadTable.GetCellContentColor(WorkRow, 0)
    .SetTextHeight RowType, _
      ReadTable.GetCellTextHeight(WorkRow, 0)
    .SetTextStyle RowType, _
      ReadTable.GetCellTextStyle(WorkRow, 0)
  End With
End Function
```

```
Private Function IsTableStyleFound(ByVal Name As String) As Boolean
  On Error Resume Next
  Dim testSty As AcadTableStyle
  Set testSty = GetTableStyle(Name)

  IsTableStyleFound = (Err.Number = 0)
End Function
```

```
Private Function GetTableStyle(ByVal Name As String) As AcadTableStyle
  Dim tblDict As AcadDictionary
  Set tblDict = ThisDrawing.Database.Dictionaries.Item("ACAD_TableStyle")

  Set GetTableStyle = tblDict.GetObject(Name)
End Function
```



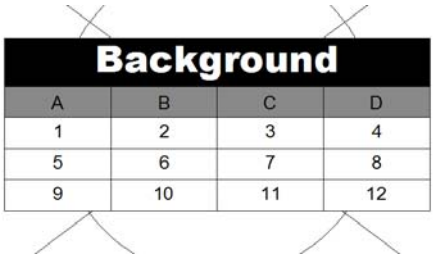
Here is a procedure that you can use to test the above procedures.

```
Public Sub Test()
  ThisDrawing.Utility.Prompt vbCrLf
  Dim TableUtils As MyTableUtils.TableUtils
  Set TableUtils = MyTableUtils.Initialize
  Dim TableInfo As Collection
  Set TableInfo = TableUtils.PickTable("Select table: ")
  Set TableUtils = Nothing
  If Not (TableInfo Is Nothing) Then
    Dim myTable As AcadTable
    Set myTable = TableInfo(1)
    ModifyTableStyle "AU Test", "AU code sample", myTable
  End If
End Sub
```

Applying a background mask, after the fact

Table styles can be set up to apply backgrounds before a table is even created. However, you may have some existing tables that you want to apply a neutral (white) background to without user intervention. The following simple procedure will apply a white (255, 255, 255) background to given tables. The results will look odd on non-white graphics backgrounds, yet plots correctly. In addition, objects located beneath the table will be masked.

```
Public Sub SetTableToBackground(Table As AcadTable)
  Dim newColor As AcadAcCmColor
  Set newColor = New AcadAcCmColor
  newColor.SetRGB 255, 255, 255
  With Table
    Dim row As Long, column As Long
    For row = 0 To .Rows - 1
      For column = 0 To .Columns - 1
        If .GetCellBackgroundColorNone(row, column) = True Then
          .SetCellBackgroundColorNone row, column, False
          .SetCellBackgroundColor row, column, newColor
        End If
      Next column
    Next row
  End With
  Set newColor = Nothing
End Sub
```



Here is a procedure that you can use to test the above procedure.

```
Public Sub Test()
  ThisDrawing.Utility.Prompt vbCrLf
  Dim TableUtils As MyTableUtils.TableUtils
  Set TableUtils = MyTableUtils.Initialize

  Dim TableInfo As Collection
  Set TableInfo = TableUtils.SelectRow("Select table: ")
  Set TableUtils = Nothing
  If Not (TableInfo Is Nothing) Then
    Dim myTable As AcadTable
    Set myTable = TableInfo(1)
    SetTableToBackground myTable
  End If
End Sub
```

Conclusion

The samples you have seen show that expanding the functionality of tables is possible via VBA. All of these samples can be translated into Visual LISP rather easily. Therefore, think outside the box and use the power of either VBA or Visual LISP to give tables the power you need to boost your productivity.

I hope you enjoyed this course. Please feel free to tell me what you thought of the course during this conference.