VERSION 1.0 CLASS

BEGIN

  MultiUse = -1  'True

END

Attribute VB\_Name = "cCell"

Attribute VB\_GlobalNameSpace = False

Attribute VB\_Creatable = False

Attribute VB\_PredeclaredId = False

Attribute VB\_Exposed = False

'gistThat@mcpher.com :do not modify this line - see ramblings.mcpher.com for details: updated on 8/18/2014 3:54:00 PM : from manifest:3414394 gist https://gist.github.com/brucemcpherson/3414216/raw/cCell.cls

' a data Cell - holds value at time of loading, or can be kept fresh if there might be formula updates

Option Explicit

' Version 2.04 -

'for more about this

' http://ramblings.mcpher.com/Home/excelquirks/classeslink/data-manipulation-classes

'to contact me

' http://groups.google.com/group/excel-ramblings

'reuse of code

' http://ramblings.mcpher.com/Home/excelquirks/codeuse

Private pValue As Variant                   ' value of cell when first loaded

Private pColumn As Long                     ' column number

Private pParent As cDataRow                 ' cDataRow to which this belongs

Public Property Get row() As Long

    row = pParent.row

End Property

Public Property Get column() As Long

    column = pColumn

End Property

Public Property Get parent() As cDataRow

    Set parent = pParent

End Property

Public Property Get myKey() As String

    myKey = makeKey(pParent.parent.headings(pColumn).toString)

End Property

Public Property Get where() As Range    ' return the range from whence it came

    If row = 0 Then

    ' its a heading

        Set where = pParent.where.Resize(1, 1).Offset(row, pColumn - 1)

    Else

        Set where = pParent.where.Resize(1, 1).Offset(, pColumn - 1)

    End If

End Property

Public Property Get refresh() As Variant ' refresh the current value and return it

    pValue = where.value

    refresh = pValue

End Property

Public Property Get toString(Optional sFormat As String = vbNullString, \_

            Optional followFormat As Boolean = False, \_

            Optional deLocalize As Boolean = False) As String ' Convert to a string, applying a format if supplied

    Dim s As String, os As String, ts As String

    If Len(sFormat) > 0 Then

        os = Format(value, sFormat)

    Else

        If followFormat Then

            s = where.NumberFormat

            If Len(s) > 0 And s <> "General" Then

                os = Format(value, s)

            Else

                os = CStr(value)

            End If

        Else

            os = CStr(value)

        End If

    End If

    If deLocalize Then

        If VarType(value) = vbDouble Or VarType(value) = vbCurrency Or VarType(value) = vbSingle Then

            ' commas to dots

            ts = Mid(CStr(1.1), 2, 1)

            os = Replace(os, ts, ".")

        ElseIf VarType(value) = vbBoolean Then

            If value Then

                os = "true"

            Else

                os = "false"

           End If

        End If

    End If

    toString = os

End Property

Public Property Get value() As Variant   ' return the value, refreshing it if necessary

    If pParent.parent.keepFresh Then

        value = refresh

    Else

        value = pValue

    End If

End Property

Public Property Let value(p As Variant)

    parent.parent.columns(pColumn).dirty = True

    If pParent.parent.keepFresh Then

        Commit p

    Else

        pValue = p

    End If

End Property

Public Function needSwap(cc As cCell, e As eSort) As Boolean

    ' this can be used from a sorting alogirthm

    Select Case e

        Case eSortAscending

            needSwap = LCase(toString) > LCase(cc.toString)

        Case eSortDescending

            needSwap = LCase(toString) < LCase(cc.toString)

        Case Else

            needSwap = False

    End Select

End Function

Public Function Commit(Optional p As Variant) As Variant

    Dim v As Variant

    If Not IsMissing(p) Then

        pValue = p

    End If

    where.value = pValue

    Commit = refresh

End Function

Public Function create(par As cDataRow, colNum As Long, rCell As Range, \_

            Optional v As Variant) As cCell        ' Fill the Cell up

    ' if v is specifed we knw the value without needing to access the sheet

    If IsMissing(v) Then

        pValue = rCell.value

    Else

        pValue = v

    End If

    pColumn = colNum

    Set pParent = par

    Set create = Me                     ' return for convenience

End Function

Public Sub tearDown()

    ' clean up

    Set pParent = Nothing

End Sub

VERSION 1.0 CLASS

BEGIN

  MultiUse = -1  'True

END

Attribute VB\_Name = "cDataRow"

Attribute VB\_GlobalNameSpace = False

Attribute VB\_Creatable = False

Attribute VB\_PredeclaredId = False

Attribute VB\_Exposed = False

'gistThat@mcpher.com :do not modify this line - see ramblings.mcpher.com for details: updated on 8/18/2014 3:54:03 PM : from manifest:3414394 gist https://gist.github.com/brucemcpherson/3414216/raw/cDataRow.cls

' a collection of data Cells representing one row of data

Option Explicit

'v 2.02

'for more about this

' http://ramblings.mcpher.com/Home/excelquirks/classeslink/data-manipulation-classes

'to contact me

' http://groups.google.com/group/excel-ramblings

'reuse of code

' http://ramblings.mcpher.com/Home/excelquirks/codeuse

Private pCollect As Collection                  ' a collection of data Cells - one for every column in this row

Private pWhere As Range

Private pParent As cDataSet

Private pRow As Long

Private pHidden As Boolean

Public Property Get hidden()

    hidden = pHidden

End Property

Public Property Get parent() As cDataSet

    Set parent = pParent

End Property

Public Property Get row() As Long

    row = pRow

End Property

Public Property Get columns() As Collection

    Set columns = pCollect

End Property

Public Property Get where() As Range

    Set where = pWhere

End Property

Public Property Get cell(sid As Variant, Optional complain As Boolean = False) As cCell

    Dim c As cCell

    Set c = exists(sid)

    If c Is Nothing And complain Then

        MsgBox (CStr(sid) & " is not a known column heading")

    End If

    Set cell = c

End Property

Public Property Get value(sid As Variant) As Variant

    Dim cc As cCell

    Set cc = cell(sid)

    If Not cc Is Nothing Then

        value = cc.value

    End If

End Property

Public Property Get values(Optional bIncludeKey = False) As Variant

    Dim cc As cCell

    ReDim a(1 To columns.count) As Variant

    For Each cc In columns

        If cc.column <> pParent.keyColumn Or bIncludeKey Then

            a(cc.column) = cc.value

        Else

            a(cc.column) = Empty

        End If

    Next cc

    values = a

End Property

Public Function find(v As Variant, Optional bIncludeKey = False) As cCell

    Dim cc As cCell

    For Each cc In columns

        If cc.column <> pParent.keyColumn Or bIncludeKey Then

            If makeKey(cc.value) = makeKey(v) Then

                Set find = cc

                Exit Function

            End If

        End If

    Next cc

End Function

Public Function max(Optional bIncludeKey = False) As Variant

    max = Application.WorksheetFunction.max(values(bIncludeKey))

End Function

Public Function min(Optional bIncludeKey = False) As Variant

    max = Application.WorksheetFunction.min(values(bIncludeKey))

End Function

Public Function refresh(Optional sid As Variant) As Variant

    Dim dt As cCell, v As Variant

    If IsMissing(sid) Then

        For Each dt In columns

            v = dt.refresh

        Next dt

    Else

        refresh = cell(sid).refresh

    End If

End Function

Public Sub Commit(Optional p As Variant, Optional sid As Variant)

    Dim dt As cCell

    If IsMissing(sid) Then

        For Each dt In columns

            dt.Commit p

        Next dt

    Else

      cell(sid).Commit p

    End If

End Sub

Public Property Get toString(sid As Variant, Optional sFormat As String = vbNullString) As String

    toString = cell(sid).toString(sFormat)

End Property

Public Function create(dset As cDataSet, rDataRow As Range, nRow As Long, \_

                            rv As Variant) As cDataRow

    Dim rCell As Range, dcell As cCell, hcell As cCell, hr As cHeadingRow, n As Long

    Dim r As Range, dc As cDataColumn

    Set pWhere = rDataRow

    Set pParent = dset

    pRow = nRow

    n = 0

    ' recordfilter

    pHidden = False

    If (pParent.recordFilter) Then

        pHidden = rDataRow.EntireRow.hidden

    End If

    If pRow = 0 Then        ' we are doing a headingrow

        For Each r In pWhere.Cells

            n = n + 1

            If IsEmpty(r) Then

                MsgBox ("unexpected blank heading cell at " & SAd(r))

                Exit Function

            End If

            Debug.Assert Not IsEmpty(r)

            Set dcell = New cCell

            With dcell

                pCollect.add .create(Me, n, r), makeKey(CStr(r.value))

            End With

        Next r

    Else

        Set hr = pParent.headingRow

        For Each hcell In hr.headings

            ' create a cell to hold it in

            Set rCell = rDataRow.Cells(1, hcell.column)

            Set dcell = New cCell

            dcell.create Me, hcell.column, rCell, rv(nRow - 1 + LBound(rv, 1), hcell.column - 1 + LBound(rv, 2))

            pCollect.add dcell

            ' set the type of column

            Set dc = pParent.columns(hcell.column)

            With dc

                If Not IsEmpty(rCell) Then

                    If .typeofColumn <> eTCmixed Then

                        If IsDate(rCell.value) Then

                            If .typeofColumn <> eTCdate Then

                                If .typeofColumn = eTCunknown Then

                                    .typeofColumn = eTCdate

                                Else

                                    .typeofColumn = eTCmixed

                                End If

                            End If

                        ElseIf IsNumeric(rCell.value) Then

                            If .typeofColumn <> eTCnumeric Then

                                If .typeofColumn = eTCunknown Then

                                    .typeofColumn = eTCnumeric

                                Else

                                    .typeofColumn = eTCmixed

                                End If

                            End If

                        Else

                            If .typeofColumn <> eTCtext Then

                                If .typeofColumn = eTCunknown Then

                                    .typeofColumn = eTCtext

                                Else

                                    .typeofColumn = eTCmixed

                                End If

                            End If

                        End If

                    End If

                End If

            End With

        Next hcell

    End If

    Set create = Me

End Function

Private Function exists(sid As Variant) As cCell

    On Error GoTo handle

    If VarType(sid) = vbLong Or VarType(sid) = vbInteger Then

        Set exists = pCollect(sid)

    Else

        Set exists = pCollect(pParent.headings(makeKey(CStr(sid))).column)

    End If

    Exit Function

handle:

    Set exists = Nothing

End Function

Public Sub tearDown()

    ' clean up

    Dim cc As cCell

    If Not pCollect Is Nothing Then

        For Each cc In columns

            cc.tearDown

        Next cc

        Set pCollect = Nothing

    End If

    Set pParent = Nothing

End Sub

Private Sub Class\_Initialize()

    Set pCollect = New Collection

End Sub

VERSION 1.0 CLASS

BEGIN

  MultiUse = -1  'True

END

Attribute VB\_Name = "cDataColumn"

Attribute VB\_GlobalNameSpace = False

Attribute VB\_Creatable = False

Attribute VB\_PredeclaredId = False

Attribute VB\_Exposed = False

'gistThat@mcpher.com :do not modify this line - see ramblings.mcpher.com for details: updated on 8/18/2014 4:47:42 PM : from manifest:3414394 gist https://gist.github.com/brucemcpherson/3414216/raw/cDataColumn.cls

' a collection of data Cells representing one column of data

' v2.05 -

Option Explicit

'for more about this

' http://ramblings.mcpher.com/Home/excelquirks/classeslink/data-manipulation-classes

'to contact me

' http://groups.google.com/group/excel-ramblings

'reuse of code

' http://ramblings.mcpher.com/Home/excelquirks/codeuse

Private pCollect As Collection                      ' a collection of data Cells - one for every row in this column

Private pWhere As Range

Private pParent As cDataSet

Private pColumn As Long

Private pTypeofColumn As eTypeofColumn

Private pHeadingCell As cCell    ' we can use this to find the heading for this column

Private pDirty As Boolean

Public Property Get googleType() As String

    Select Case pTypeofColumn

        Case eTCnumeric

            googleType = "number"

        Case eTCdate

            googleType = "date"

        Case Else

            googleType = "string"

    End Select

End Property

Public Property Get dirty() As Boolean

    dirty = pDirty

End Property

Public Property Let dirty(p As Boolean)

        pDirty = p

End Property

Public Property Get typeofColumn() As eTypeofColumn

    typeofColumn = pTypeofColumn

End Property

Public Property Let typeofColumn(p As eTypeofColumn)

    pTypeofColumn = p

End Property

Public Property Get column() As Long

    column = pColumn

End Property

Public Property Get rows() As Collection

    Set rows = pCollect

End Property

Public Property Get parent() As cDataSet

    Set parent = pParent

End Property

Public Property Get where() As Range

    If Not pWhere Is Nothing Then

        Set where = pWhere.Resize(pParent.rows.count)

    End If

End Property

Public Property Get cell(rowID As Variant) As cCell

    Set cell = pParent.cell(rowID, pHeadingCell.column)

End Property

Public Property Get value(rowID As Variant) As Variant

    value = cell(rowID).value

End Property

Public Function refresh(Optional rowID As Variant) As Variant

    Dim dt As cCell

    If IsMissing(rowID) Then

        For Each dt In rows

            refresh = dt.refresh

        Next dt

        refresh = Empty

    Else

        refresh = cell(rowID).refresh

    End If

End Function

Public Function filtered(v As Variant) As Collection

    ' this creates a filtered collection of cells for this column based on matching some value

    Dim c As Collection, cc As cCell

    Set c = New Collection

    For Each cc In rows

        ' this filter is in addition to any excel ones in operations

        If Not cc.parent.hidden And v = cc.value Then c.add cc

    Next cc

    Set filtered = c

End Function

Public Property Get uniqueValues(Optional es As eSort = eSortNone) As Collection

    ' return a collection of unique values for this column

    Dim cc As cCell

    Dim vUnique As Collection

    Set vUnique = New Collection

    For Each cc In rows

        If (Not cc.parent.hidden) Then

            If exists(vUnique, cc.toString) Is Nothing Then vUnique.add cc, CStr(cc.value)

        End If

    Next cc

    If es <> eSortNone Then SortColl vUnique, es

    Set uniqueValues = vUnique

End Property

Public Sub Commit(Optional p As Variant, Optional rowID As Variant)

    Dim dt As cCell, v As Variant

    If IsMissing(rowID) Then

        For Each dt In pCollect

            dt.Commit p

        Next dt

    Else

       cell(rowID).Commit p

    End If

End Sub

Public Property Get values() As Variant

    Dim cc As cCell

    ReDim a(1 To parent.visibleRowsCount) As Variant

    For Each cc In rows

        If Not cc.parent.hidden Then a(cc.row) = cc.value

    Next cc

    values = a

End Property

Public Function find(v As Variant) As cCell

    Dim cc As cCell

    For Each cc In rows

        If makeKey(cc.value) = makeKey(v) Then

            Set find = cc

            Exit Function

        End If

    Next cc

End Function

Public Function max() As Variant

    max = Application.WorksheetFunction.max(values)

End Function

Public Function min() As Variant

    min = Application.WorksheetFunction.min(values)

End Function

Public Property Get toString(rowNum As Long, Optional sFormat As String = vbNullString) As String

    toString = cell(rowNum).toString(sFormat)

End Property

Public Function create(dset As cDataSet, hcell As cCell, ncol As Long) As cDataColumn

    Dim rCell As Range, dcell As cCell

    pTypeofColumn = eTCunknown

    Set pParent = dset

    pColumn = ncol

    If Not pParent.where Is Nothing Then

        Set pWhere = hcell.where.Offset(1).Resize(dset.where.rows.count)

    End If

    Set pHeadingCell = hcell

    Set create = Me

End Function

Private Function exists(vCollect As Collection, sid As Variant) As cCell

    If Not vCollect Is Nothing Then

        On Error GoTo handle

        Set exists = vCollect(sid)

        Exit Function

    End If

handle:

    Set exists = Nothing

End Function

Public Sub tearDown()

    ' clean up

    Set pCollect = Nothing

    Set pParent = Nothing

End Sub

Private Sub Class\_Initialize()

    Set pCollect = New Collection

End Sub

VERSION 1.0 CLASS

BEGIN

MultiUse = -1 'True

END

Attribute VB\_Name = "cHeadingRow"

Attribute VB\_GlobalNameSpace = False

Attribute VB\_Creatable = False

Attribute VB\_PredeclaredId = False

Attribute VB\_Exposed = False

'gistThat@mcpher.com :do not modify this line - see ramblings.mcpher.com for details: updated on 8/18/2014 3:54:03 PM : from manifest:3414394 gist https://gist.github.com/brucemcpherson/3414216/raw/cHeadingRow.cls

' a collection of Cells that contain the headings associated with a dataset

' v2.03 - 3414216

Option Explicit

'for more about this

' http://ramblings.mcpher.com/Home/excelquirks/classeslink/data-manipulation-classes

'to contact me

' http://groups.google.com/group/excel-ramblings

'reuse of code

' http://ramblings.mcpher.com/Home/excelquirks/codeuse

Private pDataRow As cDataRow

Public Property Get parent() As cDataSet

Set parent = pDataRow.parent

End Property

Public Property Get dataRow() As cDataRow

Set dataRow = pDataRow

End Property

Public Property Get headings() As Collection

Set headings = pDataRow.columns

End Property

Public Property Get where() As Range

Set where = pDataRow.where

End Property

Public Function create(dset As cDataSet, rHeading As Range, Optional keepFresh As Boolean = False) As cHeadingRow

Dim rCell As Range, hcell As cCell, n As Long, dr As cDataRow

With pDataRow

.create dset, rHeading, 0, keepFresh

End With

Set create = Me

End Function

Public Function exists(s As String) As cCell

If headings.count > 0 Then

On Error GoTo handle

Set exists = headings(makeKey(s))

Exit Function

End If

handle:

Set exists = Nothing

End Function

Public Property Get headingList() As String

' return a comma separated list of the headings

Dim t As cStringChunker, cc As cCell

Set t = New cStringChunker

For Each cc In headings

t.add cc.toString & ","

Next cc

' remove final comma if there is one

headingList = t.chop.content

Set t = Nothing

End Property

Public Function validate(complain As Boolean, ParamArray args() As Variant) As Boolean

Dim i As Long, s As String

s = ""

For i = LBound(args) To UBound(args)

If exists(CStr(args(i))) Is Nothing Then

s = s & args(i) & ","

End If

Next i

If Len(s) = 0 Then

validate = True

Else

s = left(s, Len(s) - 1)

If complain Then

MsgBox "The following required columns are missing from dataset " & parent.Name & ":" & s

End If

End If

End Function

Public Sub tearDown()

' clean up

pDataRow.tearDown

Set pDataRow = Nothing

End Sub

Private Sub Class\_Initialize()

Set pDataRow = New cDataRow

End Sub

VERSION 1.0 CLASS

BEGIN

MultiUse = -1 'True

END

Attribute VB\_Name = "cADO"

Attribute VB\_GlobalNameSpace = False

Attribute VB\_Creatable = False

Attribute VB\_PredeclaredId = False

Attribute VB\_Exposed = False

Option Explicit

' we can use this class to return an ADO recordset from a closed Excel file

Private pConnection As ADODB.Connection

Private pRecordSet As ADODB.Recordset

Private pSQLFields As String

Private pSQLExtra As String

Private pSQL As String

Private pDataSource As String

Private pTable As String

Private pDset As cDataSet

Private pWhere As Range

Private pClearSheet As Boolean

Private pCreateDset As Boolean

Private peAdoConnection As eAdoConnections

Public Enum eAdoConnections

eAdoAuto

eAdoExcel2007

eAdoAccess2007

eAdoUnknown

End Enum

Public Property Get sql() As String

sql = pSQL

End Property

Public Property Get where() As Range

Set where = pWhere

End Property

Public Property Get dset() As cDataSet

Set dset = pDset

End Property

Public Function init(Optional rOutRange As Range = Nothing, \_

Optional sDataSource As String = vbNullString, \_

Optional bClearsheet As Boolean = True, \_

Optional bCreateDset As Boolean = True, \_

Optional eConnection As eAdoConnections = eAdoAuto, \_

Optional complain As Boolean = True) As cADO

Set pWhere = rOutRange

pCreateDset = bCreateDset

pClearSheet = bClearsheet

Set pConnection = New ADODB.Connection

Set pRecordSet = New ADODB.Recordset

pDataSource = sDataSource

peAdoConnection = eConnection

Set pDset = Nothing

If pDataSource = vbNullString Then pDataSource = ThisWorkbook.path & "\" & ThisWorkbook.Name

If peAdoConnection = eAdoAuto Then

peAdoConnection = tryToGetConnectionType

If peAdoConnection = eAdoUnknown Then

If complain Then MsgBox ("Dont know how to connect to " & pDataSource)

Set init = Nothing

Exit Function

End If

End If

Set init = Me

End Function

Public Function kill()

With pRecordSet

.Close

End With

With pConnection

.Close

End With

Set pRecordSet = Nothing

Set pConnection = Nothing

End Function

Private Function tryToGetConnectionType() As eAdoConnections

Dim p As Long

tryToGetConnectionType = eAdoUnknown

p = InStrRev(pDataSource, ".")

If p <> 0 Then

Select Case Mid(pDataSource, p + 1)

Case "xlsm", "xlsx", "xlsb"

tryToGetConnectionType = eAdoExcel2007

Case "accdb"

tryToGetConnectionType = eAdoAccess2007

End Select

End If

End Function

Public Function execute(Optional sTable As String = vbNullString, \_

Optional sSqlFields As String = "\*", \_

Optional sSqlExtra As String = vbNullString) As cADO

Dim fCol As ADODB.Field, r As Range, c As Long, w As Worksheet, cString As String

' CONNECT TO target datasource and execute sql

Set pConnection = New ADODB.Connection

pTable = sTable

If pTable = vbNullString Then pTable = ActiveSheet.Name

Select Case peAdoConnection

Case eAdoExcel2007

cString = "Provider=Microsoft.ACE.OLEDB.12.0;" & \_

"Data Source=" & pDataSource & ";" & \_

"Extended Properties=""Excel 12.0;HDR=Yes"";"

pSQL = Trim("select " & thisOrThat(sSqlFields, pSQLFields) & " from [" & \_

thisOrThat(sTable, pTable) & "$] " & thisOrThat(sSqlExtra, pSQLExtra))

Case eAdoAccess2007

cString = "Provider=Microsoft.ACE.OLEDB.12.0;" & \_

"Data Source=" & pDataSource & ";" & \_

"Persist Security Info=False;"

pSQL = Trim("select " & thisOrThat(sSqlFields, pSQLFields) & " from [" & \_

thisOrThat(sTable, pTable) & "] " & thisOrThat(sSqlExtra, pSQLExtra))

Case Else

Debug.Assert False

End Select

With pConnection

.Open cString

End With

Set pRecordSet = New ADODB.Recordset

With pRecordSet

.Open pSQL, pConnection, adOpenStatic, adLockOptimistic

' headings

If pWhere Is Nothing Then

Set w = Sheets.add

Set pWhere = w.Cells(1, 1)

End If

If pClearSheet Then pWhere.Worksheet.Cells.ClearContents

Set r = pWhere.Resize(1, 1)

For Each fCol In .Fields

r.value = fCol.Name

Set r = r.Offset(, 1)

Next fCol

Set r = pWhere.Resize(1, 1).Offset(1)

While Not .EOF

c = 0

For Each fCol In .Fields

r.Offset(, c).value = fCol.value

c = c + 1

Next fCol

Set r = r.Offset(1)

.MoveNext

Wend

' reset size of created data

Set pWhere = pWhere.Resize(r.row - pWhere.row, .Fields.count)

End With

' now let's create a new cDataSet

If pCreateDset Then

Set pDset = New cDataSet

With pDset

.populateData pWhere, , pTable, , , , , , , False

End With

End If

Set execute = Me

End Function

Private Function thisOrThat(sThis As String, sThat As String) As String

If sThis = vbNullString Then

thisOrThat = sThat

Else

thisOrThat = sThis

End If

End Function

Private Function createTable(tableName As String, cj As cJobject)

' drop existing version

' this is the only time i've ever used resume next

On Error Resume Next

pConnection.execute "DROP TABLE " & tableName

On Error GoTo 0

Dim jo As cJobject, u As String

u = vbNullString

For Each jo In cj.children

If (Len(u) > 0) Then u = u + ","

u = u & jo.child("name").toString & "," & jo.child("type").toString

Next jo

u = "CREATE TABLE " & tableName & "(" & u & ")"

End Function

Private Function insertIntoTable(ds) As cJobject

Dim c As String, dr As cDataRow, dc As cCell

For Each dr In ds.rows

c = vbNullString

For Each dc In dr.columns

If Len(c) > 0 Then c = c + ","

Next dc

Next dr

End Function