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 = "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 = "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 = "cDataSet"

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 3:54:01 PM : from manifest:3414394 gist https://gist.github.com/brucemcpherson/3414216/raw/cDataSet.cls

' class cDataSet

' v2.12 - 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

Option Compare Text

Private pCollect As Collection ' a collection of data rows - one for every row in the data

Private pCollectColumns As Collection ' a collection of data columns - one for every column in the data

Private pWhere As Range

Private pHeadingRow As cHeadingRow

Private pName As String

Private pisLab As Boolean

Private pKeepfresh As Boolean

Private pParent As cDataSets

Private pRecordFilter As Boolean

Private pLikely As Boolean

Const cJobName = "cDataSet"

Public Enum eJsonConv

eJsonConvPropertyNames

End Enum

Private pKeyColumn As Long

Public Property Get self() As cDataSet

Set self = Me

End Property

Public Property Get activeListObject() As ListObject

' this one checks for any intersection with a table and stores it

Dim o As ListObject

Set o = intersectListObject(headingRow.where)

If o Is Nothing Then Set o = intersectListObject(where)

Set activeListObject = o

End Property

Private Function intersectListObject(r As Range) As ListObject

Dim o As ListObject

If Not r Is Nothing Then

For Each o In r.Worksheet.ListObjects

If Not Intersect(o.Range, r) Is Nothing Then

Set intersectListObject = o

Exit Function

End If

Next o

End If

End Function

Public Function makeListObject(Optional sName As String = vbNullString) As ListObject

' creates a list object the to map the current dataset - will use the dataset name to generate a name if not given

If sName = vbNullString Then sName = "table\_" + self.Name

Set makeListObject = \_

self.where.Worksheet.ListObjects.add(xlSrcRange, self.headingRow.where.Resize(self.rows.count + 1), , xlYes)

makeListObject.Name = sName

End Function

Public Property Get visibleRowsCount() As Long

Dim n As Long, dr As cDataRow

If pRecordFilter Then

n = 0

For Each dr In rows

If Not dr.hidden Then n = n + 1

Next dr

visibleRowsCount = n

Else

visibleRowsCount = rows.count

End If

End Property

Public Property Get recordFilter() As Boolean

recordFilter = pRecordFilter

End Property

Public Property Get keyColumn() As Long

keyColumn = pKeyColumn

End Property

Public Property Get keepFresh() As Boolean

keepFresh = pKeepfresh

End Property

Public Property Get parent() As cDataSets

Set parent = pParent

End Property

Public Property Get Name() As String

Name = pName

End Property

Public Property Get rows() As Collection

Set rows = pCollect

End Property

Public Property Get columns() As Collection

Set columns = pCollectColumns

End Property

Public Property Get headings() As Collection

Set headings = pHeadingRow.headings

End Property

Public Property Get where() As Range

Set where = pWhere

End Property

Public Property Get headingRow() As cHeadingRow

Set headingRow = pHeadingRow

End Property

Public Property Set headingRow(p As cHeadingRow)

Set pHeadingRow = p

End Property

Public Property Get cell(rowID As Variant, sid As Variant) As cCell

Dim dr As cDataRow

Set dr = row(rowID)

If Not dr Is Nothing Then Set cell = dr.cell(sid)

End Property

Public Property Get isCellTrue(rowID As Variant, sid As Variant) As Boolean

Dim cc As cCell, s As String

Set cc = cell(rowID, sid)

isCellTrue = False

If (Not cc Is Nothing) Then

Select Case LCase(cc.toString)

Case "yes", "y", "1", "true"

isCellTrue = True

End Select

End If

End Property

Public Property Get value(rowID As Variant, sid As Variant, \_

Optional complain As Boolean = True) As Variant

On Error GoTo screwed

value = cell(rowID, sid).value

Exit Property

screwed:

MsgBox ("could not get value at row " & rowID & " column " & sid & " in dataset " & Name)

Exit Property

End Property

Public Function letValue(p As Variant, rowID As Variant, sid As Variant) As Variant

cell(rowID, sid).value = p

End Function

Public Property Get toString(rowID As Variant, sid As Variant) As String

toString = CStr(value(rowID, sid))

End Property

Public Property Get row(rowID As Variant) As cDataRow

If Not pisLab Then

If VarType(rowID) <> vbInteger And VarType(rowID) <> vbLong Then

MsgBox "Dataset " & pName & " must have labels enabled to use non-numeric labels"

Exit Property

End If

End If

Set row = exists(rowID)

End Property

Public Property Get column(sid As Variant) As cDataColumn

Set column = pCollectColumns(sid)

End Property

Public Property Get jObject(Optional jSonConv As eJsonConv = eJsonConvPropertyNames, \_

Optional datesToIso As Boolean = False, \_

Optional includeParseTypes As Boolean = False, \_

Optional includeDataSetName As Boolean = True, \_

Optional dataSetName As String = vbNullString) As cJobject

' convert dataset to a JSON string

Dim dr As cDataRow, dh As cCell, dc As cCell, cr As cJobject, ca As cJobject, d As Date, jName As String

' create serialization object

Dim cj As cJobject

Set cj = New cJobject

jName = cJobName

If dataSetName <> vbNullString Then jName = dataSetName

' so far only implemented the property names conversion

Debug.Assert jSonConv = eJsonConvPropertyNames

If includeDataSetName Then

cj.init Nothing, pName

Set cr = cj.add(jName).addArray

Else

Set cr = cj.init(Nothing).addArray

End If

For Each dr In rows

With cr.add

For Each dc In dr.columns

Set dh = headings(dc.column)

If columns(dc.column).googleType = "number" Then

.add dh.toString, dc.value

ElseIf datesToIso And columns(dc.column).googleType = "date" Then

If includeParseTypes Then

With .add(dh.toString)

.add "\_\_type", "Date"

.add "iso", toISODateTime(dc.value)

End With

Else

.add dh.toString, toISODateTime(dc.value)

End If

Else

.add dh.toString, dc.toString

End If

Next dc

End With

Next dr

' return from branch where data starts

If includeDataSetName Then

Set jObject = cj.child(jName)

Else

Set jObject = cr

End If

End Property

Public Function refresh(Optional rowID As Variant, Optional sid As Variant) As Variant

' this one can be a single cell refresh or more

Dim dr As cDataRow

refresh = Empty

If IsMissing(rowID) And IsMissing(sid) Then

For Each dr In rows

dr.refresh

Next dr

ElseIf IsMissing(rowID) Then

refresh = column(sid).refresh

ElseIf IsMissing(sid) Then

refresh = row(rowID).refresh

Else

refresh = cell(rowID, sid).refresh

End If

End Function

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

' this one can be a single cell refresh or more

Dim dr As cDataRow

If IsMissing(rowID) And IsMissing(sid) Then

For Each dr In rows

dr.Commit p

Next dr

ElseIf IsMissing(rowID) Then

column(sid).Commit p

ElseIf IsMissing(sid) Then

row(rowID).Commit p

Else

cell(rowID, sid).Commit p

End If

End Sub

Private Function create(rp As Range, \_

Optional sn As String = vbNullString, Optional blab As Boolean = False, \_

Optional keepFresh As Boolean = False, Optional stopAtFirstEmptyRow = True, \_

Optional sKey As String = vbNullString, Optional maxDataRows As Long = 0) As cDataSet

Dim dRow As cDataRow, dcol As cDataColumn, hcell As cCell, exitwhile As Boolean

Dim topRow As Long, nRow As Long, ncol As Long, m As Long, av As Variant

Dim rv As Variant, i As Long

pKeepfresh = keepFresh

If sn = vbNullString Then

pName = rp.Worksheet.Name

Else

pName = sn

End If

' take the whle thing or a maximum no of rows

m = rp.rows.count - 1

If maxDataRows > 0 And maxDataRows < m Then m = maxDataRows

If (m > 0) Then

Set pWhere = rp.Offset(1).Resize(m, headings.count)

End If

pName = makeKey(pName)

pisLab = blab

If pisLab Then

If sKey = vbNullString Then

pKeyColumn = 1

Else

pKeyColumn = headingRow.exists(sKey).column

End If

End If

' create the columns

ncol = 0

For Each hcell In headings

Set dcol = New cDataColumn

ncol = ncol + 1

dcol.create Me, hcell, ncol

pCollectColumns.add dcol, makeKey(hcell.value)

Next hcell

' get the shape of a blank delimited table

If (m > 0) Then

If stopAtFirstEmptyRow Then

Set pWhere = toEmptyRow(pWhere)

End If

' read in the whole lot at once

If Not pWhere Is Nothing Then

' excel doesnt return an array if range size is 1.

av = pWhere.value

If IsArray(av) Then

rv = av

Else

ReDim rv(1, 1)

rv(LBound(rv, 1), LBound(rv, 2)) = av

End If

For i = LBound(rv, 1) To UBound(rv, 1)

Set dRow = New cDataRow

dRow.create Me, pWhere.Offset(i - LBound(rv, 1)).Resize(1), i + 1 - LBound(rv, 1), rv

If pisLab Then

If exists(makeKey(dRow.cell(pKeyColumn).value)) Is Nothing Then

pCollect.add dRow, makeKey(dRow.cell(pKeyColumn).value)

Else

MsgBox ("Could not add duplicate key " + dRow.cell(pKeyColumn).toString + \_

" in data set " + pName + " column " + headings(pKeyColumn).toString + \_

" at " + SAd(dRow.where))

End If

Else

pCollect.add dRow

End If

For Each dcol In pCollectColumns

dcol.rows.add dRow.cell(dcol.column)

Next dcol

Next i

End If

Else

Set pWhere = Nothing

End If

Set create = Me

End Function

Public Function populateJSON(job As cJobject, rstart As Range, \_

Optional wClearContents As Boolean = True, \_

Optional stopAtFirstEmptyRow As Boolean = True) As cDataSet

Dim joRow As cJobject, joCol As cJobject, rm As Range

' take a json object and apply it to a range

If job Is Nothing Then

MsgBox "input json object not defined"

ElseIf Not job.isArrayRoot Then

MsgBox job.key & " must be a rowise array object"

Else

If wClearContents Then

rstart.Worksheet.Cells.ClearContents

End If

For Each joRow In job.children

For Each joCol In joRow.children

With joCol

Set rm = rstart.Cells(joRow.childIndex + 1, .childIndex)

rm.value = .value

rstart.Cells(1, .childIndex).value = .key

End With

Next joCol

Next joRow

' now do a normal populate

Set populateJSON = populateData(rstart.Resize(rm.row - rstart.row + 1, \_

rm.column - rstart.column + 1), \_

, , , , , , , , stopAtFirstEmptyRow)

End If

End Function

Public Function populateGoogleWire(sWire As String, rstart As Range, \_

Optional wClearContents As Boolean = True, \_

Optional stopAtFirstEmptyRow As Boolean = True) As cDataSet

Dim jo As cJobject, s As String, p As Long, e As Long, joc As cJobject, jc As cJobject, jr As cJobject, cr As cJobject

Dim jt As cJobject, v As Variant, aString As Variant, newWire As Boolean

Dim jStart As String

jStart = "table:"

p = InStr(1, sWire, jStart)

'there have been multiple versions of wire ...

If p = 0 Then

'try the other one

jStart = q & ("table") & q & ":"

p = InStr(1, sWire, jStart)

newWire = True

End If

' take a google wire string and apply it to a range

p = InStr(1, sWire, jStart)

e = Len(sWire) - 1

If p <= 0 Or e <= 0 Or p > e Then

MsgBox " did not find table definition data"

Exit Function

End If

If Mid(sWire, e, 2) <> ");" Then

MsgBox ("incomplete google wire message")

Exit Function

End If

' encode the 'table:' part to a cjobject

p = p + Len(jStart)

s = "{" & jStart & "[" & Mid(sWire, p, e - p - 1) & "]}"

' google protocol doesnt have quotes round the key of key value pairs,

' and i also need to convert date from javascript syntax new Date()

s = rxReplace("(new\sDate)(\()(\d+)(,)(\d+)(,)(\d+)(\))", s, "'$3/$5/$7'")

If Not newWire Then s = rxReplace("(\w+)(:)", s, "'$1':")

' this should return an object as follow

' {table:[ cols:[c:[{id:x,label:x,pattern:x,type:x}] , rows:[ c:[(v:x,f:x}] ]}

Set jo = New cJobject

Set jo = jo.deSerialize(s, eDeserializeGoogleWire)

'need to convert that to cdataset:[{label:"x",,,},{},,,]

'column labels can be extracted then from jo.child("1.cols.n.label") .. where 'n'= column number

Set joc = New cJobject

Set cr = joc.init(Nothing, cJobName).addArray

For Each jr In jo.child("1.rows").children

With cr.add

For Each jc In jo.child("1.cols").children

Set jt = jr.child("c").children(jc.childIndex)

' sometimes there is no "v" if a null value

If Not jt.childExists("v") Is Nothing Then

Set jt = jt.child("v")

End If

If jc.child("type").toString = "date" Then

' month starts at zero in javascript

aString = Split(jt.toString, "/")

If LBound(aString) <= UBound(aString) Then

If UBound(aString) - LBound(aString) <> 2 Then

Debug.Print jt.fullKey, jt.toString & " should have been a date"

v = jt.value

Else

v = DateSerial(CInt(aString(0)), CInt(aString(1)) + 1, CInt(aString(2)))

End If

Else

v = Empty

End If

Else

v = jt.value

End If

''Debug.Print jc.fullKey, jc.Child("type").toString, \_

'' jc.Child("id").toString, jt.toString, jc.Child("label").toString, v

.add jc.child("label").toString, v

Next jc

End With

Next jr

If joc.hasChildren Then

If joc.child(1).hasChildren Then

Set populateGoogleWire = populateJSON(joc, rstart, wClearContents, stopAtFirstEmptyRow)

cr.tearDown

joc.tearDown

Exit Function

End If

End If

MsgBox ("there was no actionable data - check that your google doc types reflect the data in the cells")

End Function

Public Function rePopulate() As cDataSet

' this repopulates and creates a new cdataset

Dim newSet As cDataSet, s As String

If pKeyColumn > 0 Then

s = headingRow.headings(pKeyColumn)

End If

Set newSet = New cDataSet

' delete it from parent collection

If Not pParent Is Nothing Then

pParent.dataSets.remove (pName)

End If

' recreate it with the same parameters as before

Set rePopulate = newSet.populateData(firstCell(headingRow.where), , pName, \_

pisLab, , , pLikely, s, , , pRecordFilter)

End Function

Private Sub Class\_Initialize()

Set pHeadingRow = New cHeadingRow

Set pCollect = New Collection

Set pCollectColumns = New Collection

End Sub

Public Function load(sheetName As String, \_

Optional parameterBlock As String = vbNullString) As cDataSet

' this is just a quick populateData with most common parameters

Set load = populateData(wholeSheet(sheetName), , , parameterBlock <> vbNullString, parameterBlock, , True)

End Function

Public Function populateData(Optional rstart As Range = Nothing, Optional keepFresh As Boolean = False, Optional sn As String = vbNullString, \_

Optional blab As Boolean = False, Optional blockstarts As String = vbNullString, \_

Optional ps As cDataSets, \_

Optional bLikely As Boolean = False, \_

Optional sKey As String = vbNullString, \_

Optional maxDataRows As Long = 0, \_

Optional stopAtFirstEmptyRow As Boolean = True, \_

Optional brecordFilter As Boolean = False) As cDataSet

Dim blockName As String, rp As Range, rInput As Range

pRecordFilter = brecordFilter

pLikely = bLikely

If rstart Is Nothing Then

Set rInput = getLikelyColumnRange

ElseIf bLikely Then

Set rInput = getLikelyColumnRange(rstart.Worksheet)

Else

Set rInput = rstart

End If

' this is about taking a block from the range rather than the whole range

blockName = makeKey(sn)

If blockstarts <> vbNullString Then

Set rp = cleanFind(blockstarts, rInput.Resize(, 1), True, True)

If rp Is Nothing Then

Exit Function

End If

If blockName = vbNullString Then

blockName = makeKey(blockstarts)

End If

If (bLikely Or stopAtFirstEmptyRow) Then

Set rp = toEmptyBox(rp.Resize(rInput.rows.count - rp.row + 1, rInput.columns.count))

Else

Set rp = toEmptyCol(rp.Resize(rInput.rows.count - rp.row + 1, rInput.columns.count))

End If

Else

Set rp = rInput

End If

' set up headings

pHeadingRow.create Me, rp.Resize(1)

' create dataset

create rp, blockName, blab, keepFresh, stopAtFirstEmptyRow, sKey, maxDataRows

Set populateData = Me

Set pParent = ps

If Not pParent Is Nothing Then pParent.dataSets.add Me, pName

End Function

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

Dim dr As cDataRow

ReDim a(1 To visibleRowsCount) As Variant

For Each dr In rows

If Not dr.hidden Then a(dr.row) = dr.values(bIncludeKey)

Next dr

values = a

End Property

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

Dim dr As cDataRow, cc As cCell

For Each dr In rows

Set cc = dr.find(v, bIncludeKey)

If Not cc Is Nothing Then

Set find = cc

Exit Function

End If

Next dr

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

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

End Function

Public Function flushDirtyColumns()

Dim dc As cDataColumn

For Each dc In columns

If dc.dirty Then

dc.Commit

dc.dirty = False

End If

Next dc

End Function

Public Function bigCommit(Optional rout As Range = Nothing, Optional clearWs As Boolean = False, \_

Optional headOrderArray As Variant = Empty, \_

Optional filterHead As String = vbNullString, Optional filterValue As Variant = Empty, \_

Optional filterApproximate As Boolean = True, \_

Optional outputHeadings As Boolean = True, Optional filterUpperValue) As Long

' this one does a quick bulk commit

Dim rTarget As Range, headOrder As Collection, hcell As cCell, nHeads As Long, s As String, j As Long

Dim dArray As Variant, dr As cDataRow, n As Long, i As Long, filterCol As Long, fArray As Variant

' get start of where we are putting this to

If rout Is Nothing Then

Set rTarget = headingRow.where

Else

Set rTarget = rout

End If

'possible that we clear the target worksheet frst

If clearWs Then rTarget.Worksheet.Cells.ClearContents

' its possible to specify only a subset of columns, or reorder them

If IsEmpty(headOrderArray) Then

' all columns are required

Set headOrder = headings

Else

' a subset or reordering is required

Set headOrder = New Collection

For nHeads = LBound(headOrderArray) To UBound(headOrderArray)

Set hcell = headingRow.exists(CStr(headOrderArray(nHeads)))

If Not hcell Is Nothing Then

headOrder.add hcell, makeKey(hcell.value)

Else

s = s & headOrderArray(nHeads) & ","

End If

Next nHeads

If Len(s) > 0 Then

MsgBox "These fields do not exist " & s

End If

End If

' is there a filter ?

filterCol = 0

If filterHead <> vbNullString Then

Set hcell = headingRow.exists(filterHead)

If hcell Is Nothing Then

MsgBox (filterHead & " does not exist to filter on..ignoring")

Else

filterCol = hcell.column

End If

End If

' now create the array

If headOrder.count > 0 Then

n = 0

If outputHeadings Then n = 1

ReDim dArray(1 To rows.count + n, 1 To headOrder.count)

Set rTarget = rTarget.Resize(pCollect.count + n, headOrder.count)

i = 0

If outputHeadings Then

' headings

For Each hcell In headOrder

i = i + 1

dArray(1, i) = hcell.value

Next hcell

End If

For Each dr In pCollect

If filterOk(dr, filterCol, filterValue, filterApproximate, filterUpperValue) Then

If Not recordFilter Or Not dr.hidden Then

n = n + 1

i = 0

For Each hcell In headOrder

i = i + 1

dArray(n, i) = dr.cell(hcell.column).value

Next hcell

End If

End If

Next dr

If filterCol <> 0 And n <> pCollect.count + 1 Then

Set rTarget = rTarget.Resize(n, headOrder.count)

ReDim fArray(1 To n, 1 To headOrder.count)

For i = 1 To n

For j = 1 To headOrder.count

fArray(i, j) = dArray(i, j)

Next j

Next i

dArray = Empty

rTarget = fArray

Else

rTarget = dArray

End If

End If

bigCommit = n

End Function

Private Function filterOk(dr As cDataRow, filterCol As Long, \_

filterValue As Variant, filterApproximate As Boolean, Optional filterUpperValue As Variant) As Boolean

Dim filterUpper As Variant

' added capability for ranged filter

If (IsMissing(filterUpperValue)) Then

filterUpper = filterValue

Else

filterUpper = filterUpperValue

End If

' note that filterApproximate is incompatible with a filter range

' you should set filterapproximate to false for the uppervalue to have an effect

filterOk = True

If filterCol <> 0 Then

With dr.cell(filterCol)

If filterApproximate Then

filterOk = (.value Like filterValue)

Else

filterOk = (.value <= filterUpper And .value >= filterValue)

End If

End With

End If

End Function

Private Function exists(sid As Variant) As cDataRow

On Error GoTo handle

Set exists = pCollect(sid)

Exit Function

handle:

Set exists = Nothing

End Function

Public Sub tearDown()

' clean up

Dim dr As cDataRow, dc As cDataColumn

If Not pCollect Is Nothing Then

For Each dr In rows

dr.tearDown

Next dr

Set pCollect = Nothing

End If

If Not pHeadingRow Is Nothing Then

pHeadingRow.tearDown

Set pHeadingRow = Nothing

End If

If Not pCollectColumns Is Nothing Then

For Each dc In columns

dc.tearDown

Next dc

Set pCollectColumns = Nothing

End If

Set pParent = Nothing

End Sub

‘\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

VERSION 1.0 CLASS

BEGIN

MultiUse = -1 'True

END

Attribute VB\_Name = "cDataSets"

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 3:54:01 PM : from manifest:3414394 gist https://gist.github.com/brucemcpherson/3414216/raw/cDataSets.cls

Option Explicit

' v2.01

'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

' CdataSets

Private pCollect As Collection

Private pName As String

Public Property Get dataSets() As Collection

Set dataSets = pCollect

End Property

Public Property Get dataSet(sn As String, Optional complain As Boolean = False) As cDataSet

Dim ds As cDataSet

Set ds = exists(sn)

If ds Is Nothing Then

If complain Then MsgBox ("data set " & sn & " doesnt exist")

End If

Set dataSet = ds

End Property

Public Property Get Name() As String

Name = pName

End Property

Public Function create(Optional sName As String = "DataSets") As cDataSets

pName = sName

Set create = Me

End Function

Public Function init(Optional rInput As Range = Nothing, Optional keepFresh As Boolean = False, \_

Optional sn As String = vbNullString, \_

Optional blab As Boolean = False, Optional blockstarts As String, \_

Optional bLikely As Boolean = False, \_

Optional sKey As String = vbNullString, \_

Optional respectFilter As Boolean = False) As cDataSet

Dim ds As cDataSet

Set ds = New cDataSet

With ds

.populateData rInput, keepFresh, sn, blab, blockstarts, Me, bLikely, sKey, , , respectFilter

End With

''pCollect.add ds, ds.name

Set init = ds

End Function

Private Function exists(sid As Variant) As cDataSet

On Error GoTo handle

Set exists = pCollect(sid)

Exit Function

handle:

Set exists = Nothing

End Function

Public Sub tearDown()

' clean up

Dim ds As cDataSet

If Not pCollect Is Nothing Then

For Each ds In dataSets

ds.tearDown

Next ds

Set pCollect = Nothing

End If

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 = "cMyClass"

Attribute VB\_GlobalNameSpace = False

Attribute VB\_Creatable = False

Attribute VB\_PredeclaredId = False

Attribute VB\_Exposed = False

Option Explicit

Private pkey As Long

Private pName As String

Private pChildren As Collection

Public Property Get key() As Long

key = pkey

End Property

Public Property Get Name() As String

Name = pName

End Property

Public Property Get children() As Collection

Set children = pChildren

End Property

Public Property Let key(p As Long)

pkey = p

End Property

Public Function init(k As Long, sName As String) As cMyClass

pkey = k

pName = sName

Set pChildren = New Collection

Set init = Me

End Function

'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

VERSION 1.0 CLASS

BEGIN

MultiUse = -1 'True

END

Attribute VB\_Name = "cMyClass2"

Attribute VB\_GlobalNameSpace = False

Attribute VB\_Creatable = False

Attribute VB\_PredeclaredId = False

Attribute VB\_Exposed = False

Option Explicit

Private pkey As Long

Private pName As String

Private pChildren As Dictionary

Public Property Get key() As Long

key = pkey

End Property

Public Property Get Name() As String

Name = pName

End Property

Public Property Get children() As Dictionary

Set children = pChildren

End Property

Public Property Let key(p As Long)

pkey = p

End Property

Public Function init(k As Long, sName As String) As cMyClass2

pkey = k

pName = sName

Set pChildren = New Dictionary

Set init = Me

End Function