## Buffering details from EJScreen_Tech_Doc_Version_2.2:

EJScreen allows a user to define a buffer, such as the circle that includes everything within 1 mile of a specific point. Non-circular, user-defined shapes also can be defined to represent buffers of any shape. A report summarizes the demographics of residents within this buffer, as well as the environmental indicators and EJ Index values within the buffer.

The summary within a buffer is designed to represent the average resident within the buffer, and also provides an estimate of the total population residing in the buffer. For example, the traffic proximity indicator for a buffer is the population-weighted average of all the traffic indicator values in the buffer. Similarly, the percent people of color would be a weighted average, which is the same as the overall percent people of color for all residents in the buffer.

Some block groups will be partly inside and partly outside a buffer, and any buffer analysis must estimate how much of each block group's population is inside the buffer. Areal apportionment of block groups is one standard method, but it assumes that population is evenly spread throughout a block group, which may be far from the actual distribution of residents. Areal apportionment of blocks would be even more accurate but extremely computationally intensive.

To provide the most accurate counts that are currently feasible for a screening tool, EJScreen uses an approach based on decennial Census block internal points. EJScreen estimates the fraction of the Census block group population that is inside the buffer by using *block*-level population counts from the decennial Census. These blocks provide data about where residents are at a higher resolution than block groups. Each block has an internal point defined by the Census Bureau, and the entire block population is counted as inside or outside the buffer depending on whether the block internal point is inside or outside. This assumption typically introduces relatively little error because blocks are so small relative to a typical buffer, so a small fraction of the total buffer population is in blocks that span an edge of the buffer. Also, any blocks along the edge of a buffer whose populations are close to 0 or 100% inside the buffer will be well represented by this assumption.

As long as users draw buffers much larger than a local block group, this method should represent the average person inside the buffer reasonably well.

The calculation of a value for the buffer is essentially the population-weighted average of the indicator values in the blocks included in the buffer, where each block uses the indicator values of the block group containing it. A block group is weighted based on the fraction of the current ACS block group population that is considered in the buffer. That fraction is estimated as the decennial Census block population divided by the decennial Census block group population. The formula below is used to estimate the population average of a raw indicator value in a buffer. This formula is simply a population-weighted average – it sums the population-weighted raw values, and then divides that sum by the total population in the buffer.

$$Value(A) = \sum_{\forall Blk,\, Blk \cap A} \frac{\frac{BlockPop}{BGPop} * BGACSPop * BG\_RawValue}{\sum_{\forall Blk, Blk \cap A} \frac{BlkPop}{BGPop} * BGACSPop}$$

"BlockPop" refers to the decennial Census block level population total (used here because the ACS does not provide block resolution), and "BG" indicates block group. "BGACSPop" is the block group estimated population count from the current ACS, which is often different than the decennial Census total for all blocks in the block group, because the ACS data used here is a composite estimate based on survey samples spanning five years, while the decennial Census is a full count at one point in time.

## Summary of .NET method to implement buffering and calculations

Note: Numbers in parentheses refer to the .NET samples below

To start the process a polygon is created to use for the EJ analysis. This can be either the same polygon that a user inputs, or a buffer on either a point, line, or polygon and a buffer distance that has been input (1). This polygon is then used to spatially query the point layer that represents block centroids, this gets all blocks within the input study area. Since EJ data is stored at the blockgroup level, block centroids are used since blocks are one level finer in detail than blockgroup and each blockgroup is comprised of one or more blocks (2). Each block centroid contains a field with the percentage weight it contributes to the overall population total. By iterating through the subset of blocks, this field is aggregated to produce a total weight each blockgroup contributes to the overall analysis (3). Then, for each blockgroup, EJ indicator fields are iterated and for each raw EJ indicator value, it is multiplied by the weight factor for the blockgroup that has been previously calculated. A running total of each of these indicator values is also stored for each indicator to be used as a denominator later to divide the result (4). Finally, each indicator is divided by the total sum for that indicator to calculate the final weighted total (5).

## Code samples to demonstrate how the EJ Screen indicator values are calculated using VB.NET

1) **If a polygon is input with 0 distance, use the polygon for the query, otherwise buffer the polygon with the given buffer distance to create the analysis polygon.**

```
If aoGeom.GeometryType =
ESRI.ArcGIS.Geometry.esriGeometryType.esriGeometryPolygon And distanceValue =
0.0# Then
   aoPoly = aoGeom
Else
   geomArrayOut = iGS.BufferGeodesic(aoGeom.SpatialReference,
aoGeom.SpatialReference, da, 0.0#, iGS.FindUnitsByWKID("EPSG", unitValue), False,
geomArrayIn)
   aoPoly = geomArrayOut.Element(0)
End If
```

2) **Use the analysis polygon to perform a spatial query on block centroid points. This returns all block centroids in the study area. Each centroid point has the Blockgroup ID it is part of and the population percentage it contains.**

```
Dim pSpatialFilter As ISpatialFilter = New SpatialFilter
       pSpatialFilter.Geometry = aoPoly
       pSpatialFilter.GeometryField = fcWeight.ShapeFieldName
       pSpatialFilter.SpatialRel =
esriSpatialRelEnum.esriSpatialRelIntersects
       fCurs = fcWeight.Search(pSpatialFilter, False)
```

3) **Iterate through the centroid features and create a lookup table that contains each unique Blockgroup ID and the sum of the total weighted percentage of all blocks for each Blockgroup.**

   For example,

   Blockgroup ID : percent weight

   510594327022: 0.75

   510594327023: 0.32

```
While Not feat Is Nothing
      cnt += 1
      Dim idVal As String =
DirectCast(feat.Value(feat.Fields.FindField(unitIDFieldInWeight)), String)
      Dim popattValObj As Object =
feat.Value(feat.Fields.FindField(popweightField))
      Dim popattValDbl As Double = Nothing
      If TypeOf (popattValObj) Is DBNull Then
         popattValDbl = 0 'set nulls to 0 so don't affect weighting
      Else
         popattValDbl = DirectCast(popattValObj, Double)
      End If
      If Not pDict.ContainsKey(idVal) Then
         Dim weightDict As New Dictionary(Of String, Double)
        weightDict(popweightField) = popattValDbl
         pDict.Add(idVal, weightDict)
       Else
         pDict.Item(idVal).Item(popweightField) += popattValDbl
       End If

       'set up FIPS pop keys if not added, add state fips and init to 0
       If Not pDictFipsPops.ContainsKey(idVal.Substring(0, 2)) Then
          pDictFipsPops.Add(idVal.Substring(0, 2), 0.0#)
       End If
      feat = fCurs.NextFeature()
End While
```

4) **Loop through the Blockgroup weight table and for each Blockgroup ID, query the EJ layer for matching EJ Blockgroup ID. Then loop each EJ indicator and multiply the raw value by the proportional weight for the Blockgroup.**

```
For Each idWeightpair In pDict
    pQuery = New QueryFilter
    'ej uses BG layer regardless of inputs, use a like clause on BG layer
for partial match if not BG
    If statlevelValue = "STATE" Or statlevelValue = "COUNTY" Or
statlevelValue = "TRACT" Then
       pQuery.WhereClause = unitIDFieldInStat & " LIKE '" &
idWeightpair.Key & "%'"
    Else
       pQuery.WhereClause = unitIDFieldInStat & " = '" & idWeightpair.Key
& "'"
    End If
    fCurs = fcStat.Search(pQuery, False)
    feat = fCurs.NextFeature()
    fCursState = fcStatState.Search(pQuery, False)
    featState = fCursState.NextFeature()
    'loop through EJ features
    While Not feat Is Nothing
        hasFeats = True
        featCount += 1
       'weight factor multiplied by ACS population, used to store pop per
2 digit state fips
        subWeightPop = (idWeightpair.Value.Item(popweightField) *
CDbl(feat.Value(feat.Fields.FindField("ACSTOTPOP"))))
        If CDbl(feat.Value(feat.Fields.FindField("ACSTOTPOP"))) = 0 Then
           statLayerZeroPopCount += 1
        End If
        'store this pop to matching fips for subtotals
        pDictFipsPops.Item(idWeightpair.Key.Substring(0, 2)) +=
subWeightPop
        'store running pop total
        totWeightPop += subWeightPop
       'get subweights for each unique denominator and increment its
totaller
       'check for null denominator, if null, set to zero
```

```
        For Each dkey As String In denomkeys
           'weight each denominator by pop weight, e.g. ACSTOTPOP for
     population based indicators (other denominators are used for certain
     indicators, other denominator fields include
     ACSTOTHU,ACSEDUCBAS,ACSTOTHH,ACSUNEMPBAS,ACSIPOVBAS). This is for
     the numerator of the EJ equation.
           denomFieldsSubWeights.Item(dkey) =
(idWeightpair.Value.Item(popweightField) *
CDbl(CheckNullReturnZero(feat.Value(feat.Fields.FindField(dkey)))))
           'increment the denominator running total, this is for summed
     denominator in the equation later.
           denomFieldsTotalWeights.Item(dkey) +=
(idWeightpair.Value.Item(popweightField) *
CDbl(CheckNullReturnZero(feat.Value(feat.Fields.FindField(dkey)))))
        Next
        'loop through each EJ indicator
        For Each sumItem In sumfields
       'if DB null skip incrementing
       If Not IsDBNull(feat.Value(feat.Fields.FindField(sumItem))) Then
          If fldhash.Item(sumItem) = nullDoubleValue Then
             fldhash.Item(sumItem) = 0.0# 'init this stat to 0 but keep -
99999 as default to test for all-null case
             Dim currDenom = denomFieldsNameLookup.Item(sumItem)
             'multiply raw value by current denominator value which is the
     weighted universe value for this indicator, e.g. weighted total
     population for pop based indicators.
             'Add to running total for this indicator this completes the
     numerator in the equation and performs the summation of values.
             fldhash.Item(sumItem) +=
CDbl(feat.Value(feat.Fields.FindField(sumItem))) *
denomFieldsSubWeights.Item(currDenom)
          End If
          If categoryfields.Item(sumItem) = "P_EJ2" Or
          categoryfields.Item(sumItem) = "P_EJ5" Then
             'add to State hash too
          If Not
               IsDBNull(featState.Value(featState.Fields.FindField(sumIte
          m))) Then
               If fldhashState.Item(sumItem) = nullDoubleValue Then
                  fldhashState.Item(sumItem) = 0.0#
                  'init this stat to 0 but keep -99999 as default to test
               for all-null case
               End If
               Dim currDenom = denomFieldsNameLookup.Item(sumItem)
               fldhashState.Item(sumItem) +=
CDbl(featState.Value(featState.Fields.FindField(sumItem))) *
denomFieldsSubWeights.Item(currDenom)
          End If
     End If
     Next
```

```
            feat = fCurs.NextFeature()
            featState = fCursState.NextFeature()
      End While
Next
```

5) **Iterate through the saved indicator calculations and divide by accumulated denominator total, this completes the EJ equation.**

```
For Each sumItem In pdfoutput
    Dim pdfField As Object = Nothing
    Dim rawPre As String = "
    'save raw vals and orig field name for lup steps, output raw values
with alias to the result
    Dim rawValSave As New Dictionary(Of String, Double)
    Dim rawValSaveState As New Dictionary(Of String, Double)
    Dim nullString As String = "N/A"
    If totWeightPop > 0 Then
       For Each sumItem In fldhash
          Dim currValue As Double = nullDoubleValue
          Dim currDemogTotalName As String =
          denomFieldsNameLookup.Item(sumItem.Key)
          If Not sumItem.Value = nullDoubleValue Then
             'catch division by 0
             If denomFieldsTotalWeights.Item(currDemogTotalName) > 0 Then
                'divide by total denominator
                 currValue = sumItem.Value /
          denomFieldsTotalWeights.Item(currDemogTotalName)
             Else
               currValue = 0.0#
             End If
          End If
          'format the output based on category and add to result string
     that is returned
          Select Case categoryfields.Item(sumItem.Key)
               Case "P_EJ2"
                    'skip output
                    'use below for test add to see raw vals for lookups
                    'rawPre = "RAW_EJ2_"
                    'result.AddString(rawPre &
          lookupfields.Item(sumItem.Key), sumItem.Value / totWeightPop)
               Case "P_EJ5"
                    'skip output
                    'use below for test add to see raw vals for lookups
                    'rawPre = "RAW_EJ5_"
                    'result.AddString(rawPre &
          lookupfields.Item(sumItem.Key), sumItem.Value / totWeightPop)
```

```
                Case "P_ENV"
                     rawPre = "RAW_E_"
                     'format display by signif digs
                     If Not currValue = nullDoubleValue Then
                        currValue = getEnvText(CStr(currValue),
           formatter.Item(sumItem.Key))
                        result.AddString(rawPre &
           lookupfields.Item(sumItem.Key), currValue)
                     Else
                        result.AddString(rawPre &
           lookupfields.Item(sumItem.Key), nullString)
                     End If
                Case "P_DEM"
                     rawPre = "RAW_D_"
                     If Not currValue = nullDoubleValue Then
                        currValue = roundNumber(currValue * 100, 0)
                        result.AddString(rawPre &
           lookupfields.Item(sumItem.Key), CStr(currValue) & "%")
                     Else
                        result.AddString(rawPre &
           lookupfields.Item(sumItem.Key), nullString)
                     End If
           End Select
      End If
Next
```