```
Friend Module ModPacker
    Structure Sequence
        Public Len As Integer           'Length of the sequence in bytes (0 based)
        Public Off As Integer           'Offset of Match sequence in bytes (1 based), 0 if Literal Sequence
        Public Nibbles As Integer
        Public TotalBits As Integer     'Total Bits in Buffer
    End Structure

#Const AllowShortMidMatches = True    'This would result in saving a few bytes per disk side but unpacking is slower...

    Public BytePtr As Integer           'Buffer Byte Stream Pointer
    Public BitPtr As Integer            'Buffer Bit Stream Pointer
    Public NibblePtr As Integer         'Buffer 4Bit Stream Pointer
    Public BitPos As Integer            'Bit Position in the Bit Stream byte
    Public BitsLeft As Integer          'Bits left in BitPtr

    Public TotalBits As Integer = 0

    Public TotLits, TotSM, TotNMM, TotFMM, TotNLM, TotFLM As Integer

    Private TransitionalBlock As Boolean
    Private LastBlockOfBundle As Boolean = False

    Private FirstLitOfBlock As Boolean = False      'If true, this is the first block of next file in same buffer, Lit Selector Bit NOT NEEEDED
    Private NextFileInBuffer As Boolean = False     'Indicates whether the next file is added to the same buffer

    Private BlockUnderIO As Integer = 0
    Private AdLoPos As Byte, AdHiPos As Byte

    Private ReadOnly MatchSelector As Integer = 1
    Private ReadOnly LitSelector As Integer = 0

    'Match offsets - stored 0-based
    Private ReadOnly MaxFarOffset As Integer = 65536    '0-based (257-65536, stored as 256-65535)
    Private ReadOnly MaxNearOffset As Integer = 256     '0-based (1-256, stored as 0-255)
    Private ReadOnly MaxShortOffset As Integer = 64     '0-based (1-64, stored as 0-63)

    Private MaxOffset As Integer = MaxNearOffset * 3   '3 is most optimal for size, loading and disk creating speed

    'Match lengths
    Private ReadOnly MaxLongLen As Byte = 255           '1-based (32-255, stored the same)
    Private ReadOnly MaxMidLen As Byte = 31             '1-based (2-31, stored the same)
    Private ReadOnly MaxShortLen As Byte = 3 + 1        '0-based (2-4, stored 1-3), cannot be 0 because it is preserved for EndTag

    'Private ReadOnly LongMatchTag As Byte = &HF8    'Could be changed to &H00, but this is more economical
    'Private ReadOnly NextFileTag As Byte = &HFC
    'Private ReadOnly EndTag As Byte = 0             'Could be changed to &HF8, but this is more economical (Number of EndTags > Number of LongMatchTags)

    Private ReadOnly NearLongMatchTag As Byte = &H_84
    Private ReadOnly FarLongMatchTag As Byte = &H_04
    Private ReadOnly EndOfBundleTag As Byte = &H_00
    Private ReadOnly EndOfBlockTag As Byte = &H_00
    Private ReadOnly NextFileTag As Byte = &H_80

    Private ReadOnly MaxLitLen As Integer = 16
```

```
56
57      Private MatchBytes As Integer = 0
58      Private MatchBits As Integer = 0
59      Private LitBits As Integer = 0
60      Private MLen As Integer = 0
61      Private MOff As Integer = 0
62
63      Private ReadOnly MaxBits As Integer = 2048
64      Private ReadOnly MaxLitPerBlock As Integer = 251 - 1     'Maximum number of literals that fits in
a block, LitCnt is 0-based
65      '256 - (AdLo, AdHi , 1 Bit, 1 Nibble, Number of Lits)
66
67      Private Seq() As Sequence           'Sequence array, to find the best sequence
68
69      Private SL(), SO(), NL(), NO(), FL(), FO(), FFL(), FFO() As Integer   'Short, Near, and Far
Lengths and Offsets
70
71      Private SI As Integer   'Sequence array index, Offset max and min for far matches
72      Private LitSI As Integer            'Sequence array index of last literal sequence
73      Private StartPtr As Integer
74
75      Public PartialFileIndex, PartialFileOffset As Integer
76      Private CurrentFileIndex As Integer
77      Private ReferenceFileStart As Integer
78      Private ReferenceUnderIO As Integer
79
80      'Private Cycles As Integer
81      'Private BitStreamBytes As Integer
82
83      '//---------------------------
84      '// DECODING MATCH BYTES
85      '//---------------------------
86      '//                        X-3          X-2          X-1      X                OFFSET (STORED AS)
LENGTH (STORED AS)
87      '// SHORT:                                 oooooLL     $01-$40 ($00-$3F)       $02-$04 ($01-$03)
88      '// NEAR MID:                          oooooooo 1LLLLL00         $01-$FF ($01-$FF)
$02-$1F ($02-$1F)
89      '// NEAR LONG:               oooooooo LLLLLLLL 10000100         $01-$FF ($01-$FF)       $20-$FF
($20-$FF)
90      '// FAR MID:                      oooooooo HHHHHHHH 0LLLLL00     $0100-$FFFF
$03-$1F ($03-$1F)
91      '// FAR LONG:   oooooooo LLLLLLLL HHHHHHHH 00000100     $0100-$FFFF             $20-$FF ($20-$FF)
92      '//
93      '// NEAR LONG MATCH:                       10000100
94      '// FAR LONG MATCG:                       00000100
95      '// END OF BLOCK:                                    00000000
96      '// NEXT FILE IN BUNDLE:                          10000000
97      '// END OF BUNDLE:                         00000000 10000100 (USE 00001000 instead??? far mid
length of $02 Is Not used)
98      '//                                                                             (far match check
would take 4 more cycles...)
99      '// LEGEND:
100     '// L - Match length
101     '// o - Match offset LSB
102     '// H - Match offset MSB (if offset is 16-bit)
103
104     Public Sub PackFile(PN As Byte(), FileIndex As Integer, Optional FA As String = "", Optional FUIO
As Boolean = False)
105         If DoOnErr Then On Error GoTo Err
106
107         '-------------------------------------------------------------------------------------------
-------------
108         'PROCESS FILE
109         '-------------------------------------------------------------------------------------------
```

```
------------

        MaxOffset = If(Packer = PackerTypes.Faster, MaxNearOffset, MaxNearOffset * 3)

        Prg = PN
        FileUnderIO = FUIO
        PrgAdd = Convert.ToInt32(FA, 16)
        PrgLen = Prg.Length

        ReDim SL(PrgLen - 1), SO(PrgLen - 1), NL(PrgLen - 1), NO(PrgLen - 1), FL(PrgLen - 1),
FO(PrgLen - 1)
        ReDim FFL(PrgLen - 1), FFO(PrgLen - 1)

        ReDim Seq(PrgLen)       'This is actually one element more in the array, to have starter
element with 0 values

        With Seq(0)             'Initialize first element of sequence - WAS Seq(1)!!!
            '.Len = 0           '1 Literal byte, Len is 0 based
            '.Off = 0           'Offset=0 -> literal sequence, Off is 1 based
            .TotalBits = 10     'LitLen bit + 8 bits, DO NOT CHANGE IT TO 9!!!
        End With

        '-------------------------------------------------------------------------------------------
-------------
        'SEARCH REFERENCE FILE FOR FAR MATCHES
        '-------------------------------------------------------------------------------------------
-------------

        CurrentFileIndex = FileIndex

        If Packer = PackerTypes.Better Then
            For I As Integer = 0 To VFiles.Count - 1
                ReferenceFile = VFiles(I).ToArray
                ReferenceFileStart = Convert.ToInt32(VFileAddrA(I), 16)

                SearchVirtualFile(PrgLen - 1, I, ReferenceFileStart + ReferenceFile.Length - 1,
ReferenceFileStart + 1)
            Next

            For I As Integer = 0 To PartialFileIndex - 1
                ReferenceFile = Prgs(I).ToArray
                ReferenceFileStart = Convert.ToInt32(FileAddrA(I), 16)

                SearchReferenceFile(PrgLen - 1, I, ReferenceFileStart + ReferenceFile.Length - 1,
ReferenceFileStart + 1)
            Next

            If PartialFileIndex > -1 Then
                ReferenceFile = Prgs(PartialFileIndex).ToArray
                ReferenceFileStart = Convert.ToInt32(FileAddrA(PartialFileIndex), 16)

                'Search partial file from offset to end of file
                SearchReferenceFile(PrgLen - 1, PartialFileIndex, ReferenceFileStart +
ReferenceFile.Length - 1, ReferenceFileStart + PartialFileOffset + 1)
            End If
        End If

        '-------------------------------------------------------------------------------------------
-------------
        'CALCULATE BEST SEQUENCE
        '-------------------------------------------------------------------------------------------
-------------

```

```
163         CalcBestSequence(PrgLen - 1, 1, True)     'SeqLowestIndex is 1 because Prg(0) is always 1 literal on its own, we need at lease 2 bytes for a match
164
165         '-------------------------------------------------------------------------------------------------------------
166         'DETECT BUFFER STATUS AND INITIALIZE COMPRESSION
167         '-------------------------------------------------------------------------------------------------------------
168
169         FirstLitOfBlock = True                                      'First block of next file in same buffer, Lit Selector Bit NOT NEEEDED
170
171         If BytePtr = 255 Then
172             NextFileInBuffer = False                                'This is the first file that is being added to an empty buffer
173         Else
174             NextFileInBuffer = True                                 'Next file is being added to buffer that already has data
175         End If
176
177         If NewBundle Then
178             TransitionalBlock = True                                'New bundle, this is a transitional block
179             BlockPtr = ByteSt.Count                                 'If this is a new bundle, store Block Counter Pointer
180             NewBundle = False
181         End If
182
183         Buffer(BytePtr) = (PrgAdd + PrgLen - 1) Mod 256             'Add Address Hi Byte
184         AdLoPos = BytePtr
185
186         BlockUnderIO = CheckIO(PrgLen - 1)                          'Check if last byte of block is under IO or in ZP
187
188         If BlockUnderIO = 1 Then
189             BytePtr -= 1                                            'And skip 1 byte (=0) for IO Flag
190         End If
191
192         Buffer(BytePtr - 1) = Int((PrgAdd + PrgLen - 1) / 256)      'Add Address Lo Byte
193         AdHiPos = BytePtr - 1
194
195         BytePtr -= 2
196         LastByte = BytePtr          'The first byte of the ByteStream after (BlockCnt and IO Flag and) Address Bytes (251..253)
197
198         '-------------------------------------------------------------------------------------------------------------
199         'COMPRESS FILE
200         '-------------------------------------------------------------------------------------------------------------
201
202         Pack()
203
204         Exit Sub
205 Err:
206         ErrCode = Err.Number
207         MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")
208
209     End Sub
210
211     Private Sub SearchReferenceFile(PrgMaxIndex As Integer, RefIndex As Integer, RefMaxAddress As Integer, RefMinAddress As Integer)
212         'PrgMaxIndex = relative address within Prg
```

```
        'RefMaxAddress, RefMinAddress = absolute addresses within reference file
        If FileIOA(RefIndex) = False Then
            'Reference file is not under I/O, check if it is IN I/O
            If ((RefMinAddress >= &HD000) AndAlso (RefMinAddress < &HE000)) OrElse
                   ((RefMaxAddress >= &HD000) AndAlso (RefMaxAddress < &HE000)) Then
                'This reference file is IN the I/O registers, SKIP anything between $d000-$dfff
                If RefMinAddress + 1 < &HCFFF Then
                    'Search from start to $cfff
                    FindFarMatches(PrgMaxIndex, 1, &HCFFF, RefMinAddress + 1)
                End If
                If RefMaxAddress > &HE000 Then
                    'Search from $e000 to end of file
                    FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, &HE000 + 1)
                End If
            Else
                FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, RefMinAddress + 1)
            End If
        Else
            FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, RefMinAddress + 1)
        End If

    End Sub

    Private Sub SearchVirtualFile(PrgMaxIndex As Integer, RefIndex As Integer, RefMaxAddress As
Integer, RefMinAddress As Integer)
        'PrgMaxIndex = relative address within Prg
        'RefMaxAddress, RefMinAddress = absolute addresses within reference file
        If VFileIOA(RefIndex) = False Then
            'Reference file is not under I/O, check if it is IN I/O
            If ((RefMinAddress >= &HD000) AndAlso (RefMinAddress < &HE000)) OrElse
                   ((RefMaxAddress >= &HD000) AndAlso (RefMaxAddress < &HE000)) Then
                'This reference file is IN the I/O registers, SKIP anything between $d000-$dfff
                If RefMinAddress + 1 < &HCFFF Then
                    'Search from start to $cfff
                    FindFarMatches(PrgMaxIndex, 1, &HCFFF, RefMinAddress + 1)
                End If
                If RefMaxAddress > &HE000 Then
                    'Search from $e000 to end of file
                    FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, &HE000 + 1)
                End If
            Else
                FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, RefMinAddress + 1)
            End If
        Else
            FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, RefMinAddress + 1)
        End If

    End Sub

    Private Sub FindMatches(SeqHighestIndex As Integer, SeqLowestIndex As Integer, Optional FirstRun
As Boolean = False)
        If DoOnErr Then On Error GoTo Err

        Dim MaxO, MaxLL, MaxSL As Integer

        '-------------------------------------------------------------------------------------------
-------------
        'FIND LONGEST SHORT AND NEAR MATCHES FOR EACH POSITION, AND FAR MATCHES WITH OFFSET < MAX.
1024
        '-------------------------------------------------------------------------------------------
-------------

        'Pos = Min>0 to Max value, direction of execution is arbitrary (could be Max to Min>0 Step -1)
```

```
271         For Pos As Integer = SeqLowestIndex To SeqHighestIndex          'Pos cannot be 0, Prg(0) is always literal as it is always 1 byte left
272
273             'Offset goes from 1 to max offset (cannot be 0)
274             MaxO = Math.Min(MaxOffset, SeqHighestIndex - Pos) 'MaxO=256 or less
275             'Match length goes from 1 to max length
276             MaxLL = Math.Min(Pos + 1, MaxLongLen)   'MaxLL = 255 or less
277             MaxSL = Math.Min(Pos + 1, MaxShortLen)  'MaxSL = 4 or less
278
279             If (FirstRun) OrElse (FO(Pos) + Pos > SeqHighestIndex) OrElse (NO(Pos) + Pos > SeqHighestIndex) OrElse (SO(Pos) + Pos > SeqHighestIndex) Then
280                 'Only run search for this Pos if this is the first pass or the previously found match has an offset beyond block
281
282                 'End If
283                 'If (FirstRun) OrElse (SL(Pos) > 0) OrElse (NL(Pos) > 0) OrElse (FL(Pos) > 0) Then
284                 'Only run search for this Pos if this is the first pass or we have previously found a match for it
285
286                 If SO(Pos) + Pos > SeqHighestIndex Then
287                     SO(Pos) = 0
288                     SL(Pos) = 0
289                 End If
290                 If NO(Pos) + Pos > SeqHighestIndex Then
291                     NO(Pos) = 0
292                     NL(Pos) = 0
293                 End If
294                 If FO(Pos) + Pos > SeqHighestIndex Then
295                     FO(Pos) = 0
296                     FL(Pos) = 0
297                 End If
298
299                 For O As Integer = 1 To MaxO                                    'O=1 to 1024 or less
300                     'Check if first byte matches at offset, if not go to next offset
301                     If Prg(Pos) = Prg(Pos + O) Then
302                         For L As Integer = 1 To MaxLL                           'L=1 to 255 or less
303                             'If L = MaxLL Then
304                             'GoTo Match
305                             If (L = MaxLL) OrElse (Prg(Pos - L) <> Prg(Pos + O - L)) Then
306                                 'Find the first position where there is NO match -> this will give us the absolute length of the match
307                                 'L=MatchLength + 1 here
308                                 If L >= 2 Then
309                                     If (O <= MaxShortOffset) AndAlso (SL(Pos) < MaxSL) AndAlso (SL(Pos) < L) Then
310                                         SL(Pos) = Math.Min(MaxSL, L)  'If(L > MaxShortLen, MaxShortLen, L)   'Short matches cannot be longer than 4 bytes
311                                         SO(Pos) = O       'Keep Offset 1-based
312                                     End If
313 #If AllowShortMidMatches Then
314                                     If (O <= MaxNearOffset) AndAlso (NL(Pos) < L) Then 'Allow short (2-byte) Mid Matches
315                                         NL(Pos) = L
316                                         NO(Pos) = O
317                                     End If
318 #Else
319                                     If (O <= MaxNearOffset) AndAlso (NL(Pos) < L) AndAlso (L > 2) Then 'Skip short (2-byte) Mid Matches
320                                         NL(Pos) = L
321                                         NO(Pos) = O
322                                     End If
323 #End If
324                                     If (O > MaxNearOffset) AndAlso (FL(Pos) < L) AndAlso (L > 2) Then
```

```
                                    FL(Pos) = L
                                    FO(Pos) = O
                                End If
                            End If
                            Exit For
                        End If
                    Next
                    'If both short and long matches maxed out, we can leave the loop and go to the next Prg position
                    If (NL(Pos) = MaxLL) AndAlso (SL(Pos) = MaxSL) Then
                        Exit For
                    End If
                End If
            Next
        End If
    Next

    Exit Sub
Err:
    ErrCode = Err.Number
    MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")

End Sub

Private Sub FindFarMatches(SeqMaxIndex As Integer, SeqMinIndex As Integer, RefMaxAddress As Integer, RefMinAddress As Integer)
    If DoOnErr Then On Error GoTo Err
    'SeqMqxIndex,SeqMinIndex = relative address within Prg
    'RefMaxAddress, RefMinAddress = absolute addresses within reference file

    Dim MaxO, MaxLL, MaxSL As Integer

    If FileUnderIO = False Then
        If (PrgAdd + SeqMinIndex >= &HD000) AndAlso (PrgAdd + SeqMinIndex <= &HDFFF) Then
            If PrgAdd + SeqMaxIndex <= &HDFFF Then
                'The entire file segment is under I/O -> skip it
                Exit Sub
            Else
                'Not the entire file segment is under I/O -> only check the part that is not
                SeqMinIndex = &HE000 - PrgAdd
            End If
        ElseIf (PrgAdd + SeqMaxIndex >= &HD000) AndAlso (PrgAdd + SeqMaxIndex <= &HDFFF) Then
            SeqMaxIndex = &HCFFF - PrgAdd
        End If
    End If

    '-------------------------------------------------------------------------------------------------
    'FIND LONGEST SHORT AND NEAR MATCHES FOR EACH POSITION, AND FAR MATCHES WITH OFFSET < MAX. 1024
    '-------------------------------------------------------------------------------------------------
    Dim RefMinAddressIndex As Integer = RefMinAddress - ReferenceFileStart
    Dim RefMaxAddressIndex As Integer = RefMaxAddress - ReferenceFileStart

    Dim OffsetBase = If(ReferenceFileStart >= PrgAdd, ReferenceFileStart - PrgAdd, ReferenceFileStart + &H10000 - PrgAdd)

    'Pos = Min>0 to Max value, direction of execution is arbitrary (could be Max to Min>0 Step -1)
    For Pos As Integer = SeqMinIndex To SeqMaxIndex         'Pos cannot be 0, Prg(0) is always literal as it is always 1 byte left

```

```
            'Offset goes from 1 to max offset (cannot be 0)
            'MaxO = Math.Min(MaxNearOffset, SeqMaxIndex - Pos) 'MaxO=256 or less
            'Match length goes from 1 to max length
            For O As Integer = RefMinAddressIndex To RefMaxAddressIndex
                'Check if first byte matches at offset, if not go to next offset
                If Prg(Pos) = ReferenceFile(O) Then
                    MaxLL = Math.Min(Math.Min(Pos + 1, MaxLongLen), O - RefMinAddressIndex + 1)
'MaxLL = 255 or less
                    For L As Integer = 1 To MaxLL                               'L=1 to 255 or less
                        If (L = MaxLL) OrElse (Prg(Pos - L) <> ReferenceFile(O - L)) Then
                            'Find the first position where there is NO match -> this will give us the
absolute length of the match
                            'L=MatchLength + 1 here
                            If L > 2 Then
                                If ((O - Pos) > MaxNearOffset) AndAlso (FL(Pos) < L) Then
                                    FFL(Pos) = L
                                    FFO(Pos) = O - Pos + OffsetBase
                                End If
                            End If
                            Exit For
                        End If
                    Next
                    'If far matches maxed out, we can leave the loop and go to the next Prg position
                    If FFL(Pos) = MaxLL Then
                        Exit For
                    End If
                End If
            Next
        Next

        Exit Sub
Err:
        ErrCode = Err.Number
        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")

    End Sub

    Private Sub CalcBestSequence(SeqHighestIndex As Integer, SeqLowestIndex As Integer, Optional
FirstRun As Boolean = False)
        If DoOnErr Then On Error GoTo Err

        FindMatches(SeqHighestIndex, SeqLowestIndex, FirstRun)

        '-------------------------------------------------------------------------------------------
-------------
        'FIND BEST SEQUENCE FOR EACH POSITION
        '-------------------------------------------------------------------------------------------
-------------

        For Pos As Integer = SeqLowestIndex To SeqHighestIndex

            'Start with second element, first has been initialized above
            Seq(Pos + 1).TotalBits = &HFFFFFF
            'Max block size=100 = $10000 bytes = $80000 bits, make default larger than this

            If (FL(Pos) <> 0) OrElse (FFL(Pos) <> 0) Then
                If FL(Pos) >= FFL(Pos) Then
                    CheckMatchSeq(FL(Pos), FO(Pos), Pos)
                ElseIf Pos < SeqHighestIndex Then   'The last byte of the block MUST be a literal, we
can't use Far Matches there
                    'If a reference is under I/O then this block also must be under I/O to be able to
copy the reference
```

```
                    'If ((PrgAdd + Pos + FFO(Pos) >= &HD000) AndAlso (PrgAdd + Pos + FFO(Pos) < &HE000)) OrElse
                    '((PrgAdd + Pos + FFO(Pos) - FFL(Pos) >= &HD000) AndAlso (PrgAdd + Pos + FFO(Pos) - FFL(Pos) < &HE000)) Then
                    'BlockUnderIO = 1
                    'End If
                    CheckMatchSeq(FFL(Pos), FFO(Pos), Pos)
                End If
            End If
            If NL(Pos) > 0 Then
                CheckMatchSeq(NL(Pos), NO(Pos), Pos)
            End If
            If SL(Pos) > 0 Then
                CheckMatchSeq(SL(Pos), SO(Pos), Pos)
            End If

            CheckLitSeq(Pos)

        Next

        Exit Sub
Err:
        ErrCode = Err.Number
        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")

    End Sub

    Private Sub CheckMatchSeq(SeqLen As Integer, SeqOff As Integer, Pos As Integer)
        If DoOnErr Then On Error GoTo Err
        Dim TotBits As Integer

        'If this is a far match then min len = 3, otherwise min len = 2
        Dim MinLen As Integer = If(SeqOff > MaxNearOffset, 3, 2)

        'Check all possible lengths
        For L As Integer = SeqLen To MinLen Step -1
            'Calculate MatchBits

            If (L <= MaxShortLen) AndAlso (SeqOff <= MaxShortOffset) Then
                MatchBits = 9
            ElseIf (L <= MaxMidLen) AndAlso (SeqOff <= MaxNearOffset) Then
                MatchBits = 17
            ElseIf (L > MaxMidLen) AndAlso (SeqOff > MaxNearOffset) Then
                MatchBits = 33
            Else
                MatchBits = 25
            End If

            'Calculate total bit count, independently of nibble status
            TotBits = Seq(Pos + 1 - L).TotalBits + MatchBits

            With Seq(Pos + 1)
                'See if total bit count is better than best version
                If TotBits < .TotalBits Then
                    'If better, update best version
                    .Len = L            'MatchLen is 1 based
                    .Off = SeqOff       'Off is 1 based
                    .Nibbles = Seq(Pos + 1 - L).Nibbles
                    .TotalBits = TotBits
                End If
            End With
```

```
        Next

        Exit Sub
Err:
        ErrCode = Err.Number
        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")

    End Sub

    Private Sub CheckLitSeq(Pos As Integer)
        If DoOnErr Then On Error GoTo Err

        Dim TotBits As Integer

        'Continue previous Lit sequence or start new sequence
        LitCnt = If(Seq(Pos).Off = 0, Seq(Pos).Len, -1)

        'Calculate literal bits for a presumtive LitCnt+1 value
        LitBits = Int((LitCnt + 1) / MaxLitPerBlock) * 13
        Select Case (LitCnt + 1) Mod MaxLitPerBlock
            Case 0
                LitBits += 1                        'Lits = 0    1 literal, 1 bit
            Case 1 To MaxLitLen - 1
                LitBits += 5                        'Lits = 1-15 2-16 literals, 5 bits
            Case Else
                LitBits += 13                       'Lits = 16-250       17-251 literals, 13 bits
        End Select

        'LITERALS ARE ALWAYS FOLLOWED BY MATCHES, SO TYPE SELECTOR BIT IS NOT NEEDED AFTER LITERALS AT ALL

        TotBits = Seq(Pos - LitCnt - 1).TotalBits + LitBits + ((LitCnt + 2) * 8)

        With Seq(Pos + 1)
            'See if total bit count is less than best version
            If TotBits < .TotalBits Then
                'and save it to sequence at Pos+1 (position is 1 based)
                .Len = LitCnt + 1       'LitCnt is 0 based, LitLen is 0 based
                .Off = 0                'An offset of 0 marks a literal sequence, match offset is 1 based
                .Nibbles = Seq(Pos - (LitCnt + 1)).Nibbles + If(LitBits > 1, 1, 0)
                .TotalBits = TotBits
            End If
        End With

        Exit Sub
Err:
        ErrCode = Err.Number
        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")

    End Sub

    Private Sub Pack()
        If DoOnErr Then On Error GoTo Err

        'Packing is done backwards

        Dim BufferFull As Boolean

        SI = PrgLen - 1
        StartPtr = SI
```

```
553
554         If PrgAdd + SI = &HE5FF Then
555             SI += 0
556         End If
557
558         'Cycles += 61 'From NextFile label
559         'BitStreamBytes = 0
560
561 Restart:
562         Do
563
564             If Seq(SI + 1).Off = 0 Then
565                 '-----------------------------------------------------------------
566                 'Literal sequence
567                 '-----------------------------------------------------------------
568                 LitCnt = Seq(SI + 1).Len                'LitCnt is 0 based
569                 LitSI = SI
570                 MLen = 0                                'Reset MLen - this is needed for accurate bit
    counting in sequencefits
571
572                 'The max number of literals that fit in a single buffer is 249 bytes
573                 'This bypasses longer literal sequences and improves compression speed
574                 BufferFull = False
575
576                 'Shortcut to bypass long literal sequences that wouldn't fit in the buffer anyway
577                 If LitCnt > BytePtr Then    'MaxLitPerBlock Then
578                     BufferFull = True
579                     LitCnt = BytePtr        'MaxLitPerBlock
580                 End If
581
582                 Do While LitCnt > -1
583                     If SequenceFits(LitCnt + 1, CalcLitBits(LitCnt), CheckIO(SI - LitCnt)) = True Then
584                         Exit Do
585                     End If
586                     LitCnt -= 1
587                     BufferFull = True
588                 Loop
589
590                 'Go to next element in sequence
591                 SI -= LitCnt + 1    'If nothing added to the buffer, LitCnt=-1+1=0
592
593                 If BufferFull = True Then
594                     AddLitSequence()
595                     CloseBuffer()   'The whole literal sequence did not fit, buffer is full, close it
596                 End If
597
598             Else
599                 '-----------------------------------------------------------------
600                 'Match sequence
601                 '-----------------------------------------------------------------
602
603                 BufferFull = False
604
605                 MLen = Seq(SI + 1).Len      '1 based
606                 MOff = Seq(SI + 1).Off      '1 based
607 Match:
608                 CalcMatchBytesAndBits(MLen, MOff)
609
610                 ReferenceUnderIO = 0
611
612                 If MatchBytes = 4 Then
613                     '-----------------------------------------------------------------
```

```
                    'Far Long Match - 4 match bytes + 0/1 match bit
                    '------------------------------------------------------------------

                    If (((PrgAdd + SI + MOff) Mod &H10000 >= &HD000) AndAlso ((PrgAdd + SI + MOff) Mod &H10000 < &HE000)) OrElse
                    (((PrgAdd + SI + MOff - MLen + 1) Mod &H10000 >= &HD000) AndAlso ((PrgAdd + SI + MOff - MLen + 1) Mod &H10000 < &HE000)) Then
                        If (PrgAdd + SI >= &HD000) AndAlso (PrgAdd + SI < &HE000 AndAlso (FileUnderIO = False)) OrElse
                                    (PrgAdd + SI - MLen + 1 >= &HD000) AndAlso (PrgAdd + SI - MLen + 1 < &HE000 AndAlso (FileUnderIO = False)) Then
                        Else
                            ReferenceUnderIO = 1
                        End If
                    End If

                    If SequenceFits(MatchBytes + LitCnt + 1, MatchBits + CalcLitBits(LitCnt), Math.Max(CheckIO(SI - MLen + 1), ReferenceUnderIO)) Then
                        AddLitSequence()
                        'Add far long match
                        AddFarLongMatch()
                    Else
                        MLen = MaxMidLen
                        BufferFull = True   'Buffer if full, we will need to close it
                        GoTo Check3Bytes
                    End If
                ElseIf MatchBytes = 3 Then
                    '------------------------------------------------------------------
                    'Far Mid Match or Near Long Match - 3 match bytes + 0/1 match bit
                    '------------------------------------------------------------------

                    If MOff > MaxNearOffset Then
                        If (((PrgAdd + SI + MOff) Mod &H10000 >= &HD000) AndAlso ((PrgAdd + SI + MOff) Mod &H10000 < &HE000)) OrElse
                        (((PrgAdd + SI + MOff - MLen + 1) Mod &H10000 >= &HD000) AndAlso ((PrgAdd + SI + MOff - MLen + 1) Mod &H10000 < &HE000)) Then
                            If (PrgAdd + SI >= &HD000) AndAlso (PrgAdd + SI < &HE000 AndAlso (FileUnderIO = False)) OrElse
                                        (PrgAdd + SI - MLen + 1 >= &HD000) AndAlso (PrgAdd + SI - MLen + 1 < &HE000 AndAlso (FileUnderIO = False)) Then
                            Else
                                ReferenceUnderIO = 1
                            End If
                        End If

                    End If

Check3Bytes:        If SequenceFits(MatchBytes + LitCnt + 1, MatchBits + CalcLitBits(LitCnt), Math.Max(CheckIO(SI - MLen + 1), ReferenceUnderIO)) Then
                        AddLitSequence()
                        'Add long match
                        If MOff > MaxNearOffset Then
                            'Add far mid match
                            AddFarMidMatch()
                        Else
                            'Add near long match
                            AddNearLongMatch()
                        End If
                    Else
                        MLen = MaxMidLen
                        BufferFull = True   'Buffer if full, we will need to close it
                        GoTo Check2Bytes
                    End If
                ElseIf MatchBytes = 2 Then
                    '------------------------------------------------------------------
```

```
668                     'Near Mid Match - 2 match bytes + 0/1 match bit
669                     '------------------------------------------------------------------
670 Check2Bytes:         If SequenceFits(MatchBytes + LitCnt + 1, MatchBits + CalcLitBits(LitCnt),
    CheckIO(SI - MLen + 1)) Then
671                         AddLitSequence()
672                         'Add mid match
673                         AddNearMidMatch()
674                     Else
675                         BufferFull = True
676                         If SO(SI) <> 0 Then
677                             MLen = SL(SI)   'SL and SO array indeces are 0 based
678                             MOff = SO(SI)
679                             GoTo CheckShort
680                         Else
681                             GoTo CheckLit
682                         End If  'Short vs Literal
683                     End If      'Mid vs Short
684                 Else
685                     '------------------------------------------------------------------
686                     'Short Match - 1 match byte + 0/1 match bit
687                     '------------------------------------------------------------------
688 CheckShort:         If SequenceFits(1 + LitCnt + 1, MatchBits + CalcLitBits(LitCnt), CheckIO(SI - MLen
    + 1)) Then
689                         AddLitSequence()
690                         'Add short match
691                         AddShortMatch()
692                     Else
693                         '------------------------------------------------------------------
694                         'Match does not fit, check if 1 literal byte fits
695                         '------------------------------------------------------------------
696                         BufferFull = True
697 CheckLit:               MLen = 0    'This is needed here for accurate Bit count calculation in
    sequencefits (indicates Literal, not Match)
698                         If SequenceFits(1 + LitCnt + 1, CalcLitBits(LitCnt + 1), CheckIO(SI - LitCnt))
    Then
699                             If LitCnt = -1 Then
700                                 'If no literals, current SI will be LitSI, else, do not change LitSi
701                                 LitSI = SI
702                             End If
703                             LitCnt += 1     '0 based, now add 1 for an additional literal (first byte
    of match that did not fit)
704                             SI -= 1         'Rest of LitCnt has been already subtracted from SI
705                         End If  'Literal vs nothing
706                     End If      'Short match vs literal
707                 End If          'Long, mid, or short match
708 Done:
709                 SI -= MLen
710
711                 If BufferFull Then
712                     AddLitSequence()
713                     CloseBuffer()
714                 End If
715             End If              'Lit vs match
716
717         Loop While SI >= 0
718
719         AddLitSequence()        'See if any remaining literals need to be added, space has been
    previously reserved for them
720
721         'KARAOKE BUG - fixed in Sparkle 2.2 - making sure that the first byte of the next bundle fits
    in the transitional block
722         Dim BytesNeededForNextBundle As Integer = Int(BitsNeededForNextBundle / 8)
723         If (LastBlockOfBundle) Then
```

```
724             LastBlockOfBundle = False
725             TransitionalBlock = True
726             If (SequenceFits(BytesNeededForNextBundle, 0) = False) Then 'Bits=0, MLen will be checked in SequenceFits
727                 'We have miscalculated the last block of the bundle, let's recompress it the conventional way!
728
729                 SI = StartPtr
730
731                 ResetBuffer()                       'Resets buffer variables
732                 NextFileInBuffer = False            'Reset Next File flag
733                 TransitionalBlock = False           'Only the first block of a bundle is a transitional block
734                 FirstLitOfBlock = True
735
736                 Buffer(BytePtr) = (PrgAdd + SI) Mod 256
737                 AdLoPos = BytePtr
738                 BlockUnderIO = CheckIO(SI)          'Check if last byte of prg could go under IO
739
740                 If BlockUnderIO = 1 Then
741                     BytePtr -= 1
742                 End If
743
744                 Buffer(BytePtr - 1) = Int((PrgAdd + SI) / 256) Mod 256
745                 AdHiPos = BytePtr - 1
746                 BytePtr -= 2
747                 LastByte = BytePtr              'LastByte = the first byte of the ByteStream after and Address Bytes (253 or 252 with blockCnt)
748                 CalcBestSequence(If(SI > 1, SI, 1), If(SI - MaxNearOffset > 1, SI - MaxNearOffset, 1))
749                 GoTo Restart
750             End If
751         End If
752
753         Exit Sub
754 Err:
755         ErrCode = Err.Number
756         MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")
757
758     End Sub
759
760     Private Sub CalcMatchBytesAndBits(Length As Integer, Offset As Integer) 'Match Length is 1 based
761         If DoOnErr Then On Error GoTo Err
762
763         If (Length <= MaxShortLen) AndAlso (Offset <= MaxShortOffset) Then
764             MatchBytes = 1
765         ElseIf (Length <= MaxMidLen) AndAlso (Offset <= MaxNearOffset) Then
766             MatchBytes = 2
767         ElseIf (Length > MaxMidLen) AndAlso (Offset > MaxNearOffset) Then
768             MatchBytes = 4
769         Else
770             MatchBytes = 3
771         End If
772
773         MatchBits = If(LitCnt = -1, 1, 0)
774
775         Exit Sub
776 Err:
777         ErrCode = Err.Number
778         MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")
779
780     End Sub
```

```
781
782     Private Function CalcLitBits(Lits As Integer) As Integer     'LitCnt is 0 based
783         If DoOnErr Then On Error GoTo Err
784
785         If Lits = -1 Then
786             CalcLitBits = 0                     'Lits = -1          no literals, 0 bit
787         ElseIf Lits = 0 Then
788             CalcLitBits = 2                     'Lits = 0           one literal, 1 bit
789         ElseIf Lits < MaxLitLen Then
790             CalcLitBits = 6                     'Lits = 1-15        2-16 literals, 5 bits
791         Else
792             CalcLitBits = 14                    'Lits = 15-250      17-251 literals, 13 bits
793         End If
794
795         'BUGFIX: The very first literal sequence of a file or block does not need a type selector bit
796         'As we always start with at least one literal byte
797         If (FirstLitOfBlock) AndAlso (CalcLitBits > 0) Then CalcLitBits -= 1
798
799         LitBits = CalcLitBits
800
801         Exit Function
802 Err:
803         ErrCode = Err.Number
804         MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
    + " Error")
805
806     End Function
807
808     Private Function SequenceFits(BytesToAdd As Integer, BitsToAdd As Integer, Optional
    SequenceUnderIO As Integer = 0) As Boolean
809         If DoOnErr Then On Error GoTo Err
810
811         Dim BytesFree As Integer = BytePtr      '1,2,3,...,BytePtr-1,BytePtr
812
813         'If this is a transitional block (including block 0 on disk) then we need 1 byte for block
    count (will be overwritten by Close Byte
814         If (TransitionalBlock) OrElse (BufferCnt = 0) Then
815             BytesFree -= 1
816         End If
817
818         Dim BitsFree As Integer = BitsLeft ' BitPos + If(BitPtr <> 0, 1, 0)    '0-8
819         'BitsFree = BitsLeft
820
821         'Add IO Byte ONLY if this is the first sequence in the block that goes under IO
822         BytesToAdd += If((BlockUnderIO = 0) AndAlso (SequenceUnderIO = 1), 1, 0)
823
824         'Check if we have literal sequences >1 which have bits stored in nibbles
825         'BUGFIX: first literal sequence of a block/file has one less bits than any other seuqences, so
    comparision must be made with 5 instead of 6
826         If BitsToAdd >= 5 Then
827             If NibblePtr = 0 Then 'If NibblePtr Points at buffer(0) then we need to add 1 byte for a
    new NibblePtr position in the buffer
828                 BytesFree -= 1
829             End If
830             BitsToAdd -= 4      '4 bits less to store in the BitPtr
831         End If
832
833         'Add Match/Close Bit if the last sequence was a match
834         BitsToAdd += If(MLen > 0, 1, 0)
835
836         BytesToAdd += Int(BitsToAdd / 8)
837         BitsToAdd = BitsToAdd Mod 8
838
```

```
        If BitsFree - BitsToAdd < 0 Then BytesToAdd += 1

        If BytesFree >= BytesToAdd Then
            'Check if sequence will fit within block size limits
            SequenceFits = True
            'Data will fit
            If (BlockUnderIO = 0) AndAlso (SequenceUnderIO = 1) Then
                'This is the first byte in the block that will go UIO, so lets update the buffer to include the IO flag
                For I As Integer = BytePtr To AdHiPos   'Move all data to the left in buffer, including AdHi
                    Buffer(I - 1) = Buffer(I)
                Next
                Buffer(AdHiPos) = 0                     'IO Flag to previous AdHi Position
                BytePtr -= 1                            'Update BytePtr to next empty position in buffer
                If NibblePtr > 0 Then NibblePtr -= 1    'Only update Nibble Pointer if it does not point to Byte(0)
                If (BitPtr > 0) AndAlso (BitPtr < AdHiPos) Then BitPtr -= 1          'BitPtr also needs to be moved BUT ONLY IF > 0 - BUG reported by Raistlin/G*P
                AdHiPos -= 1                            'Update AdHi Position in Buffer
                BlockUnderIO = 1                        'Set BlockUnderIO Flag
            End If
        Else
            SequenceFits = False
        End If

        Exit Function
Err:
        ErrCode = Err.Number
        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")

        SequenceFits = False

    End Function

    Private Sub AddMatchBit()
        If DoOnErr Then On Error GoTo Err

        If LitCnt = -1 Then
            AddBits(MatchSelector, 1)   'Last Literal Length was -1, we need the Match selector bit (1)
            'Cycles += 10
        End If

        LitCnt = -1

        Exit Sub
Err:
        ErrCode = Err.Number
        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")

    End Sub

    Private Sub AddFarLongMatch()
        If DoOnErr Then On Error GoTo Err

        '//                              X-3          X-2          X-1      X               OFFSET (STORED AS)     LENGTH (STORED AS)
        '//     FAR LONG:   oooooooo LLLLLLLL HHHHHHHH 00000100     $0100-$FFFF             $20-$FF ($20-$FF)

```

```
893         TotMatch += 1
894 
895         AddMatchBit()
896 
897         Buffer(BytePtr) = FarLongMatchTag                   'Long Match Flag = &H_04
898         Buffer(BytePtr - 1) = Int((MOff - 1) / 256)
899         Buffer(BytePtr - 2) = MLen
900         Buffer(BytePtr - 3) = (MOff - 1) Mod 256
901         BytePtr -= 4
902 
903         'Cycles += 77 + (15 * MLen)
904 
905         TotFLM += 1
906 
907         Exit Sub
908 Err:
909         ErrCode = Err.Number
910         MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
    + " Error")
911 
912     End Sub
913 
914 
915     Private Sub AddNearLongMatch()
916         If DoOnErr Then On Error GoTo Err
917 
918         '//                              X-3          X-2          X-1      X               OFFSET
    (STORED AS)     LENGTH (STORED AS)
919         '//     NEAR LONG:               oooooooo LLLLLLLL 10000100         $01-$100 ($00-$FF)
    $20-$FF ($20-$FF)
920 
921         TotMatch += 1
922 
923         AddMatchBit()
924 
925         Buffer(BytePtr) = NearLongMatchTag
926         Buffer(BytePtr - 1) = MLen
927         Buffer(BytePtr - 2) = MOff - 1
928         BytePtr -= 3
929 
930         'Cycles += 77 + (15 * MLen)
931 
932         TotNLM += 1
933 
934         Exit Sub
935 Err:
936         ErrCode = Err.Number
937         MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
    + " Error")
938 
939     End Sub
940 
941     Private Sub AddFarMidMatch()
942         If DoOnErr Then On Error GoTo Err
943 
944         '//                              X-3          X-2          X-1      X               OFFSET
    (STORED AS)         LENGTH (STORED AS)
945         '//     FAR MID:                    oooooooo HHHHHHHH 0LLLLL00     $0101-$FFFF ($0100-$FFFE)
    $03-$1F ($03-$1F)
946 
947         TotMatch += 1
948 
949         AddMatchBit()
```

```
950
951         Buffer(BytePtr) = MLen * 4            'Length of match (#$02-#$1f))
952         Buffer(BytePtr - 1) = Int((MOff - 1) / 256)
953         Buffer(BytePtr - 2) = (MOff - 1) Mod 256
954         BytePtr -= 3
955
956         'Cycles += 77 + (15 * MLen)
957
958         TotFMM += 1
959
960         Exit Sub
961 Err:
962         ErrCode = Err.Number
963         MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
    + " Error")
964
965     End Sub
966
967     Private Sub AddNearMidMatch()
968         If DoOnErr Then On Error GoTo Err
969
970         '//                              X-3          X-2          X-1      X               OFFSET
    (STORED AS)     LENGTH (STORED AS)
971         '//     NEAR MID:                         oooooooo 1LLLLL00         $01-$100 ($00-$FF)
    $02-$1F ($02-$1F)
972
973         TotMatch += 1
974
975         AddMatchBit()
976
977         Buffer(BytePtr) = &H_80 + (MLen * 4)    'Length of match (#$02-#$1f))
978         Buffer(BytePtr - 1) = MOff - 1          'Offset (1-256, stored as 0-255)
979         BytePtr -= 2
980
981         'Cycles += 67 + (15 * MLen)
982
983         TotNMM += 1
984
985         Exit Sub
986 Err:
987         ErrCode = Err.Number
988         MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
    + " Error")
989
990     End Sub
991
992     Private Sub AddShortMatch()
993         If DoOnErr Then On Error GoTo Err
994
995         '//                              X-3          X-2          X-1      X               OFFSET
    (STORED AS)     LENGTH (STORED AS)
996         '//     SHORT:                              oooooLL     $01-$40 ($00-$3F)       $02-$04
    ($01-$03)
997
998         TotMatch += 1
999
1000        AddMatchBit()
1001
1002        Buffer(BytePtr) = ((MOff - 1) * 4) + (MLen - 1)
1003        BytePtr -= 1
1004
1005        'Cycles += 54 + (15 * MLen)
1006
```

```
1007        TotSM += 1
1008
1009        Exit Sub
1010    Err:
1011        ErrCode = Err.Number
1012        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
    + " Error")
1013
1014    End Sub
1015
1016    Private Sub AddLitSequence()
1017        If DoOnErr Then On Error GoTo Err
1018
1019        If LitCnt = -1 Then Exit Sub
1020
1021        Dim Lits As Integer = LitCnt
1022
1023        If Lits >= MaxLitLen Then
1024            AddLitBits(MaxLitLen)
1025            'Then add number of literals as a byte
1026            Buffer(BytePtr) = Lits ' + 1
1027            BytePtr -= 1
1028        Else
1029            'Add literal bits for 1-15 literals
1030            AddLitBits(Lits)
1031        End If
1032
1033        'Then add literal bytes
1034        For I As Integer = 0 To Lits
1035            Buffer(BytePtr - I) = Prg(LitSI - I)
1036        Next
1037
1038        BytePtr -= Lits + 1
1039        LitSI -= Lits + 1
1040        Lits = -1
1041
1042        TotLits += 1
1043
1044        'DO NOT RESET LITCNT HERE, IT IS NEEDED AT THE SUBSEQUENT MATCH TO SEE IF A MATCHTAG IS
    NEEDED!!!
1045
1046        Exit Sub
1047    Err:
1048        ErrCode = Err.Number
1049        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
    + " Error")
1050
1051    End Sub
1052
1053    Private Sub AddLitBits(Lits As Integer)
1054        If DoOnErr Then On Error GoTo Err
1055
1056        'We are never adding more than MaxLitBit number of bits here
1057
1058        If Lits = -1 Then Exit Sub    'We only call this routine with LitCnt>-1
1059
1060        'This is only for statistics
1061        'TotLit += Int(Lits / (MaxLitLen + 1)) + 1
1062
1063        If FirstLitOfBlock = False Then
1064            AddBits(LitSelector, 1)              'Add Literal Selector if this is not the first
    (Literal) byte in the buffer
1065            'Cycles += 8
```

```
        Else
            FirstLitOfBlock = False
        End If

        Select Case Lits
            Case 0
                AddBits(0, 1)               'Add Literal Length Selector 0 - read no more bits
        'Cycles += 52
            Case 1 To MaxLitLen - 1
                AddBits(1, 1)               'Add Literal Length Selector 1 - read 4 more bits
                AddNibble(Lits)             'Add Literal Length: 01-0f, 4 bits (0001-1111)
        'Cycles += 62 + ((Lits + 1) * 15)
            Case MaxLitLen
                AddBits(1, 1)               'Add Literal Length Selector 1 - read 4 more bits
                AddNibble(0)                'Add Literal Length: 0, 4 bits (0000) - we will have a longer literal sequence
                'Cycles += 72 + ((Lits + 1) * 15)
        End Select

        'DO NOT RESET LitCnt HERE!!!

        Exit Sub
Err:
        ErrCode = Err.Number
        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")

    End Sub
    Private Sub AddNibble(Bit As Integer)
        If DoOnErr Then On Error GoTo Err

        If NibblePtr = 0 Then
            NibblePtr = BytePtr
            BytePtr -= 1
            Buffer(NibblePtr) = Bit
        Else
            Buffer(NibblePtr) += Bit * 16
            NibblePtr = 0
        End If

        Exit Sub
Err:
        ErrCode = Err.Number
        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")

    End Sub

    Private Sub AddBits(Bit As Integer, BCnt As Byte)
        If DoOnErr Then On Error GoTo Err

        For I As Integer = BCnt - 1 To 0 Step -1
            If BitPos < 0 Then
                BitPos += 8
                BitsLeft = 8
                BitPtr = BytePtr    'New BitPtr pos
                BytePtr -= 1        'and BytePtr pos
                'BitStreamBytes += 1 'Number of bitstream bytes in buffer
            End If
            If (Bit And 2 ^ I) <> 0 Then
                Buffer(BitPtr) = Buffer(BitPtr) Or 2 ^ BitPos
            End If
```

```
DecBitPos:
            BitPos -= 1
            BitsLeft -= 1
            If BitPos = 0 Then
                If (Buffer(BitPtr) Mod 2 = 1) Then
                    BitPos = -1
                    BitsLeft = 0
                End If
            End If
            'Very first BitPtr in buffer has a 1 in BitPos=0 (Token Bit) -> Skip It!!!
            'If (BitPtr = 0) And (BitPos = 0) Then BitPos = -1
        Next

        'MsgBox(BufferCnt.ToString + vbNewLine + BitPtr.ToString + vbNewLine + BitPos.ToString + vbNewLine + BitsLeft.ToString)

        Exit Sub
Err:
        ErrCode = Err.Number
        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")

    End Sub

    Public Function CloseBuffer() As Boolean
        If DoOnErr Then On Error GoTo Err

        CloseBuffer = True

        'Buffer(BytePtr) = EndTag            'Not needed, byte 0 will be overwritten to EndTag during loading
        AddMatchBit()

        'Cycles += 19
        'Cycles += BitStreamBytes * 14

        BlockCnt += 1
        BufferCnt += 1

        'This does not work here yet, Pack needs to be changed to a function
        'If BufferCnt > BlocksFree Then
        'MsgBox("Unable to add bundle to disk :(", vbOKOnly, "Not enough free space on disk")
        'GoTo NoDisk
        'End If

        UpdateByteStream()

        ResetBuffer()                       'Resets buffer variables

        NextFileInBuffer = False            'Reset Next File flag

        TransitionalBlock = False           'Only the first block of a bundle is a transitional block

        FirstLitOfBlock = True

        If SI < 0 Then Exit Function        'We have reached the end of the file -> exit

        'If we have not reached the end of the file, then update buffer

        '-------------------------------------------------------------------------------------------------------------
        '"COLOR BUG"
        'Compression bug related to the transitional block (i.e. finding the last block of a bundle) -
```

```
FIXED
        'Fix: add 5 or 6 bytes + 2 bits to the calculation to find the last block of a bundle
        '+2 new bundle tag, +2 NEXT Bundle address, +1 first literal byte of NEXT Bundle, +0/1 IO status of first literal byte of NEXT file
        '+1 literal bit, +1 match bit (may or may not be needed, but we don't know until the end...)
        '-------------------------------------------------------------------------------------------------------------------

        'Check if the first literal byte of the NEXT Bundle will go under I/O
        'Bits needed for next bundle is calculated in ModDisk:SortPart
        '(Next block = Second block) or (remaining bits of Last File in Bundle + Needed Bits fit in this block)

        'LETHARGY BUG - Bits Left need to be calculated from Seq(SI+1) and NOT Seq(SI)
        'Add 4 bits if number of nibbles is odd
        Dim BitsLeftInBundle As Integer = Seq(SI + 1).TotalBits + ((Seq(SI + 1).Nibbles Mod 2) * 4)

        'If the next block is the first one on a new track, no need to recalculate the sequence
        'As all previous blocks will be loaded from the previous track before this block gets loaded
        Dim NewTrack As Boolean = False
        If BufferCnt < (17 * 21) Then

            If BufferCnt Mod 21 = 0 Then NewTrack = True

        ElseIf BufferCnt < ((17 * 21) + (6 * 19)) Then

            If (BufferCnt - (17 * 21)) Mod 19 = 0 Then NewTrack = True

        ElseIf BufferCnt < ((17 * 21) + (6 * 19) + (6 * 18)) Then

            If (BufferCnt - (17 * 21) - (6 * 19)) Mod 18 = 0 Then NewTrack = True

        Else

            If (BufferCnt - (17 * 21) - (6 * 19) - (6 * 18)) Mod 17 = 0 Then NewTrack = True

        End If

        If (BitsLeftInBundle + BitsNeededForNextBundle + If(MLen = 0, 0, 1) + 8 <= ((LastByte - 1) * 8) + BitPos) AndAlso (LastFileOfBundle = True) AndAlso (NewBlock = False) Then
            'KARAOKE BUG - fixed in Sparkle 2.2
            'BitsLeftInBundle = bits left to be compressed in bundle, add 1 bit if MLen>0, add 8 bits for Block Count (to simulate 'TransitionalBlock=True')
            'If the result is less than the bits remaining free in the remaining data + the first byte of the next bundle should fit in the buffer
            'I.e. we have identified the last block of the bundle (=transitional block)
            LastBlockOfBundle = True
        Else
            LastBlockOfBundle = False
        End If

        If (LastBlockOfBundle) OrElse (NewTrack) OrElse (BlockCnt = 1) Then
            'Seq(SI+1).Bytes/Nibbles/Bits = to calculate remaining bits in file
            'BitsNeededForNextBundle (5-6 bytes + 1/2 bits)
            '+5/6 bytes +1/2 bits
            'LastByte-1: subtract close tag/block count = Byte(1)
            'Bits remaining in block: LastByte * 8 (+ remaining bits in last BitPtr (BitPos+1))
            'But we are trying to overcalculate here to avoid misidentification of the last block
            'Which would result in buggy decompression

            'This is the last block ONLY IF the remainder of the bundle + the next bundle's info fits!!!
            'AND THE NEXT Bundle IS NOT ALIGNED in which case the next block is the last one
```

```
            'Seg(SI).bit includes both the byte stream in bits and the bit stream (total bits needed to compress the remainder of the bundle)
            '+Close Tag: 8 bits
            '+BitsNeeded: 5-6 bytes for next bundle's info + 1 lit bit +/- 1 match bit (may or may not be needed, but we wouldn't know until the end)
            'For the 2nd and last blocks of a bundle and the first blocks on a new track only recalculate the first byte's sequence
            'If BlockCnt <> 1 Then MsgBox((BitsLeftInBundle + BitsNeededForNextBundle).ToString + vbNewLine + (Seq(SI + 1).TotalBits + BitsNeededForNextBundle).ToString + vbNewLine + ((LastByte - 1) * 8 + BitPos).ToString)

            If NewTrack Then

                If Packer = PackerTypes.Better Then

                    If CurrentFileIndex > PartialFileIndex Then

                        If PartialFileIndex > -1 Then
                            'Search the finished segment of partial file
                            ReferenceFile = Prgs(PartialFileIndex).ToArray
                            ReferenceFileStart = Convert.ToInt32(FileAddrA(PartialFileIndex), 16)

                            SearchReferenceFile(SI, PartialFileIndex, ReferenceFileStart + PartialFileOffset, ReferenceFileStart + 1)
                        End If

                        'Search any finished files on track (partial file < finished file < current file)
                        For I As Integer = PartialFileIndex + 1 To CurrentFileIndex - 1
                            ReferenceFile = Prgs(I).ToArray
                            ReferenceFileStart = Convert.ToInt32(FileAddrA(I), 16)

                            SearchReferenceFile(SI, I, ReferenceFileStart + ReferenceFile.Length - 1, ReferenceFileStart + 1)
                        Next

                        'Search the finished segment of this file
                        ReferenceFile = Prgs(CurrentFileIndex).ToArray
                        ReferenceFileStart = Convert.ToInt32(FileAddrA(CurrentFileIndex), 16)

                        SearchReferenceFile(SI, CurrentFileIndex, ReferenceFileStart + ReferenceFile.Length - 1, ReferenceFileStart + SI + 1)
                    Else
                        'Partial file = CurrentFile (same file, spanning multiple tracks)
                        ReferenceFile = Prgs(CurrentFileIndex).ToArray
                        ReferenceFileStart = Convert.ToInt32(FileAddrA(CurrentFileIndex), 16)

                        SearchReferenceFile(SI, CurrentFileIndex, ReferenceFileStart + PartialFileOffset, ReferenceFileStart + SI + 1)
                    End If

                    'The current file becomes the partial file, anything before it is now finished and can be used for search
                    PartialFileIndex = CurrentFileIndex
                    PartialFileOffset = SI

                End If

            End If

            'Only recalculate the very first byte's sequence
            CalcBestSequence(Math.Max(SI, 1), Math.Max(SI, 1)) '(If(SI > 1, SI, 1), If(SI > 1, SI, 1))
            If (BlockCnt <> 1) AndAlso (NewTrack = False) Then  'Do not recalculate the very first block and first blocks on each track
```

```
1291                'Last/Transitional block
1292                BitsLeftInBundle = Seq(SI + 1).TotalBits + ((Seq(SI + 1).Nibbles Mod 2) * 4)
1293                'If the new bit count does not fit in the buffer then this is NOT the last block -> recalc sequence
1294                'KARAOKE BUG - fixed in Sparkle 2.2
1295                If BitsLeftInBundle + BitsNeededForNextBundle + If(MLen = 0, 0, 1) + 8 > ((LastByte - 1) * 8) + BitPos Then
1296                    LastBlockOfBundle = False
1297                    GoTo CalcAll
1298                End If
1299            End If
1300        Else
1301            'For all other blocks recalculate the first 256*3 bytes' sequence (MaxNearOffset * 3)
1302CalcAll:
1303            CalcBestSequence(Math.Max(SI, 1), Math.Max(SI - MaxOffset, 1)) 'If(SI > 1, SI, 1), If(SI - MaxOffset > 1, SI - MaxOffset, 1))
1304        End If
1305
1306        '-------------------------------------------------------------------------------------------------
1307
1308        Buffer(BytePtr) = (PrgAdd + SI) Mod 256
1309        AdLoPos = BytePtr
1310
1311        BlockUnderIO = CheckIO(SI)          'Check if last byte of prg could go under IO
1312
1313        If BlockUnderIO = 1 Then
1314            BytePtr -= 1
1315        End If
1316
1317        Buffer(BytePtr - 1) = Int((PrgAdd + SI) / 256) Mod 256
1318        AdHiPos = BytePtr - 1
1319        BytePtr -= 2
1320        LastByte = BytePtr  'LastByte = the first byte of the ByteStream after and Address Bytes (253 or 252 with blockCnt)
1321
1322        StartPtr = SI
1323
1324        Exit Function
1325Err:
1326        ErrCode = Err.Number
1327        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")
1328NoDisk:
1329        CloseBuffer = False
1330
1331    End Function
1332
1333    Public Function CloseBundle(Optional NextFileIO As Integer = 0, Optional LastPartOnDisk As Boolean = False, Optional FromEditor As Boolean = False) As Boolean
1334        If DoOnErr Then On Error GoTo Err
1335
1336        CloseBundle = True
1337
1338        If NewBlock = True Then GoTo NewB   'The bundle will start in a new block
1339
1340        'ADDS NEW Bundle TAG (Long Match Tag + End Tag) TO THE END OF THE Bundle
1341
1342        '-------------------------------------------------------------------------------
1343        '"SPRITE BUG"
1344        'Compression bug related to the transitional block - FIXED
1345        'Fix: include NEXT file's I/O status in calculation of needed bytes
1346        '-------------------------------------------------------------------------------
```

```
1347
1348        'BYTES NEEDED: (1)Long Match Tag + (2)End Tag + (3)BitPtr + (4)AdLo + (5)AdHi + (6)1st Literal +/- (7)I/O flag
1349        Dim Bytes As Integer = 6 + NextFileIO
1350
1351        'THE FIRST LITERAL ALSO NEEDS A LITERAL BIT
1352        'DO NOT ADD MATCH BIT HERE, IT WILL BE ADDED IN SequenceFits()
1353        'Bug fixed based on CloseFile bug reported by Visage/Lethargy
1354        Dim Bits As Integer = 1
1355
1356        TransitionalBlock = True    'This is always a transitional block, unless close sequence does not fit, will add +1 for Block Count
1357
1358        If SequenceFits(Bytes, Bits) Then       'This will add the EndTag to the needed bytes
1359
1360            'Buffer has enough space for New Bundle Tag and New Bundle Info and first Literal byte (and IO flag if needed)
1361
1362            'If last sequence was a match (no literals) then add a match bit
1363            If (MLen > 0) OrElse (LitCnt = -1) Then AddBits(1, 1)
1364
1365 NextPart:  'Match Bit is not needed if this is the beginning of the next block
1366            FilesInBuffer += 1  'There is going to be more than 1 file in the buffer
1367
1368            Buffer(1) = EORtransform(0)
1369
1370            NibblePtr = 0
1371            Buffer(BytePtr) = NearLongMatchTag                          'Then add New File Match Tag
1372            Buffer(BytePtr - 1) = EndOfBundleTag
1373            BitPtr = BytePtr - 2
1374            Buffer(BitPtr) = &H1
1375            BytePtr -= 3
1376            BitPos = 7
1377            BitsLeft = 7
1378
1379            If LastPartOnDisk = True Then       'This will finish the disk
1380                Buffer(BytePtr) = BytePtr - 2   'Finish disk with a dummy literal byte that overwrites itself to reset LastX for next disk side
1381                Buffer(BytePtr - 1) = &H3       'New address is the next byte in buffer
1382                Buffer(BytePtr - 2) = &H0       'Dummy $00 Literal that overwrites itself
1383                LitCnt = 0                      'One (dummy) literal
1384                'AddLitBits()                   'NOT NEEDED, WE ARE IN THE MIDDLE OF THE BUFFER, 1ST BIT NEEDS TO BE OMITTED
1385                AddBits(0, 1)                  'ADD 2ND BIT SEPARATELY (0-BIT, TECHNCALLY, THIS IS NOT NEEDED SINCE THIS IS THE LAST BIT)
1386                '--------------------------------------------------------------------
1387                'Buffer(ByteCnt - 3) = &H0      'THIS IS THE END TAG, NOT NEEDED HERE, WILL BE ADDED WHEN BUFFER IS CLOSED
1388                'ByteCnt -= 4                                      '*BUGFIX, THANKS TO RAISTLIN/G*P FOR REPORTING
1389                '--------------------------------------------------------------------
1390                BytePtr -= 3
1391                If FromEditor = False Then
1392                    'Only if we are NOT in the Editor AND BundleNo<128
1393                    If BundleNo < 128 Then
1394                        DirBlocks((BundleNo * 4) + 3) = BitPtr
1395                        DirPtr(BundleNo) = BufferCnt
1396                    End If
1397                    'Save last, "dummy" bundle info
1398                    LastBitPtr = BitPtr
1399                    LastBufferCnt = BufferCnt
1400                    'BundleNo += 1
1401                End If
```

```
            End If

            'DO NOT CLOSE LAST BUFFER HERE, WE ARE GOING TO ADD NEXT Bundle TO LAST BUFFER
            If ByteSt.Count > BlockPtr + 255 Then     'Only save block count if block is already added to ByteSt
                ByteSt(BlockPtr + 1) = EORtransform(LastBlockCnt)   'New Block Count is ByteSt(BlockPtr+1) in buffer, not ByteSt(BlockPtr+255)
                LoaderBundles += 1
            End If
            'Debug.Print(Hex(BundleNo) + vbTab + LastBlockCnt.ToString)

            LitCnt = -1                                                 'Reset LitCnt here
        Else
NewB:       'Next File Info does not fit, so close buffer
            CloseBuffer()               'Adds EndTag and starts new buffer
            'Then add 1 dummy literal byte to new block (blocks must start with 1 literal, next bundle tag is a match tag)
            Buffer(255) = &HFD          'Dummy Address ($03fd* - first literal's address in buffer... (*NextPart above, will reserve BlockCnt)
            Buffer(254) = &H3           '...we are overwriting it with the same value
            Buffer(253) = &H0           'Dummy value, will be overwritten with itself
            LitCnt = 0
            AddLitBits(LitCnt)       'WE NEED THIS HERE, AS THIS IS THE BEGINNING OF THE BUFFER, AND 1ST BIT WILL BE CHANGED TO COMPRESSION BIT
            BytePtr = 252
            LastBlockCnt += 1

            If LastBlockCnt > 255 Then
                'Parts cannot be larger than 255 blocks compressed
                'There is some confusion here how PartCnt is used in the Editor and during Disk building...
                MsgBox("Bundle " + If(CompressBundleFromEditor = True, BundleCnt + 1, BundleCnt).ToString + " would need " + LastBlockCnt.ToString + " blocks on the disk." + vbNewLine + vbNewLine + "Bundles cannot be larger than 255 blocks!", vbOKOnly + vbCritical, "Bundle exceeds 255-block limit!")
                If CompressBundleFromEditor = False Then GoTo NoGo
            End If

            BlockCnt -= 1
            'THEN GOTO NEXT Bundle SECTION
            GoTo NextPart
        End If

        NewBlock = SetNewBlock        'NewBlock is true at closing the previous bundle, so first it just sets NewBlock2
        SetNewBlock = False            'And NewBlock2 will fire at the desired bundle

        'MsgBox(BundleCnt.ToString + vbNewLine + Hex(BitPtr))
        'DirBlocks((BundleCnt * 4) + 3) = BitPtr
        'DirPtr(BundleCnt) = BufferCnt
        'MsgBox(BundleCnt.ToString + vbNewLine + BufferCnt.ToString)
        Exit Function
Err:
        ErrCode = Err.Number
        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name + " Error")

NoGo:
        CloseBundle = False

    End Function

    Public Sub CloseFile()
        If DoOnErr Then On Error GoTo Err
```

```
1455
1456        'ADDS NEXT FILE TAG TO BUFFER
1457
1458        '4-5 bytes and 1-2 bits needed for NextFileTag, Address Bytes and first Lit byte (+1 more if
    UIO)
1459        'BYTES NEEDED: (1)End Tag + (2)AdLo + (3)AdHi + (4)1st Literal +/- (5)I/O FLAG of NEW FILE's
    1st literal
1460        'BUG reported by Raistlin/G*P
1461        Dim BytesNeededForNextFile As Integer = 4 + CheckIO(PrgLen - 1)
1462
1463        'THE FIRST LITERAL BYTE WILL ALSO NEED A LITERAL BIT
1464        'DO NOT check whether Match Bit is needed for new file - will be checked in Sequencefits()
1465        'BUG reported by Visage/Lethargy
1466        Dim BitsNeededForNextFile As Integer = 1
1467        'Type selector bit  (match vs literal) is not needed, the first byte of a file is always
    literal
1468        'So this is the literal length bit: 0 - 1 literal, 1 - more than 1 literals, would also need a
    Nibble...
1469        '...but here we only need to be able to fit 1 literal byte
1470
1471        NextFileInBuffer = True
1472
1473        If SequenceFits(BytesNeededForNextFile, BitsNeededForNextFile) Then   'DO NOT INCLUDE NEXT
    NEXT FILE'S IO STATUS HERE - IT WOULD RESULT IN AN UNWANTED I/O FLAG INSERTION
1474
1475            'Buffer has enough space for New File Match Tag and New File Info and first Literal byte
    (and I/O flag if needed)
1476
1477            'If last sequence was a match (no literals) then add a match bit
1478            If (MLen > 0) OrElse (LitCnt = -1) Then AddBits(MatchSelector, 1)
1479
1480            Buffer(BytePtr) = NextFileTag                           'Then add New File Match Tag
1481            BytePtr -= 1
1482            FirstLitOfBlock = True
1483        Else
1484            'Next File Info does not fit, so close buffer, next file will start in new block
1485            CloseBuffer()
1486        End If
1487
1488        Exit Sub
1489Err:
1490        ErrCode = Err.Number
1491        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
    + " Error")
1492
1493    End Sub
1494
1495    Public Sub ResetBuffer() 'CHANGE TO PUBLIC
1496        If DoOnErr Then On Error GoTo Err
1497
1498        ReDim Buffer(255)       'New empty buffer
1499
1500        'Initialize variables
1501
1502        FilesInBuffer = 1
1503
1504        BitPos = 7             'Reset Bit Position Counter (counts 8 bits backwards: 7-0)
1505
1506        BitPtr = 0
1507        Buffer(BitPtr) = &H1
1508        BitsLeft = 7
1509        NibblePtr = 0
1510        BytePtr = 255
```

```
1511
1512        'Cycles = 61             'From NextFile label
1513        'BitStreamBytes = 0
1514
1515        'DO NOT RESET LitCnt HERE!!! It is needed for match tag check
1516
1517        Exit Sub
1518 Err:
1519        ErrCode = Err.Number
1520        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
    + " Error")
1521
1522    End Sub
1523
1524    Public Function CheckIO(Offset As Integer, Optional NextFileUnderIO As Integer = -1) As Integer
1525        If DoOnErr Then On Error GoTo Err
1526
1527        Offset += PrgAdd
1528
1529        If Offset < 256 Then       'Are we loading to the Zero Page? If yes, we need to signal it by
    adding IO Flag
1530            CheckIO = 1
1531        ElseIf NextFileUnderIO > -1 Then
1532            CheckIO = If((Offset >= &HD000) AndAlso (Offset <= &HDFFF) AndAlso (NextFileUnderIO = 1),
    1, 0)
1533        Else
1534            CheckIO = If((Offset >= &HD000) AndAlso (Offset <= &HDFFF) AndAlso (FileUnderIO = True),
    1, 0)
1535        End If
1536
1537        Exit Function
1538 Err:
1539        ErrCode = Err.Number
1540        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
    + " Error")
1541
1542    End Function
1543
1544    Public Sub UpdateByteStream()   'THIS IS ALSO USED BY LZ4+RLE!!!
1545        If DoOnErr Then On Error GoTo Err
1546
1547        ReDim Preserve ByteSt(BufferCnt * 256 - 1)
1548
1549        For I = 0 To 255
1550            ByteSt((BufferCnt - 1) * 256 + I) = Buffer(I)
1551        Next
1552
1553        Exit Sub
1554 Err:
1555        ErrCode = Err.Number
1556        MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
    + " Error")
1557
1558    End Sub
1559
1560 End Module
1561
```