

```

1  Friend Module ModPacker
2      Structure Sequence
3          Public Len As Integer          'Length of the sequence in bytes (0 based)
4          Public Off As Integer          'Offset of Match sequence in bytes (1 based), 0 if Literal
Sequence
5          Public Nibbles As Integer
6          Public TotalBits As Integer    'Total Bits in Buffer
7      End Structure
8
9  #Const AllowShortMidMatches = True    'This would result in saving a few bytes per disk side but
unpacking is slower...
10
11     Public BytePtr As Integer           'Buffer Byte Stream Pointer
12     Public BitPtr As Integer            'Buffer Bit Stream Pointer
13     Public NibblePtr As Integer         'Buffer 4Bit Stream Pointer
14     Public BitPos As Integer            'Bit Position in the Bit Stream byte
15     Public BitsLeft As Integer          'Bits left in BitPtr
16
17     Public TotalBits As Integer = 0
18
19     Public TotLits, TotSM, TotNMM, TotFMM, TotNLM, TotFLM As Integer
20
21     Private TransitionalBlock As Boolean
22     Private LastBlockOfBundle As Boolean = False
23
24     Private FirstLitOfBlock As Boolean = False    'If true, this is the first block of next file in
same buffer, Lit Selector Bit NOT NEEDED
25     Private NextFileInBuffer As Boolean = False    'Indicates whether the next file is added to the
same buffer
26
27     Private BlockUnderIO As Integer = 0
28     Private AdLoPos As Byte, AdHiPos As Byte
29
30     Private ReadOnly MatchSelector As Integer = 1
31     Private ReadOnly LitSelector As Integer = 0
32
33     'Match offsets - stored 0-based
34     Private ReadOnly MaxFarOffset As Integer = 65536    '0-based (257-65536, stored as 256-65535)
35     Private ReadOnly MaxNearOffset As Integer = 256    '0-based (1-256, stored as 0-255)
36     Private ReadOnly MaxShortOffset As Integer = 64    '0-based (1-64, stored as 0-63)
37
38     Private MaxOffset As Integer = MaxNearOffset * 3    '3 is most optimal for size, loading and disk
creating speed
39
40     'Match lengths
41     Private ReadOnly MaxLongLen As Byte = 255          '1-based (32-255, stored the same)
42     Private ReadOnly MaxMidLen As Byte = 31            '1-based (2-31, stored the same)
43     Private ReadOnly MaxShortLen As Byte = 3 + 1        '0-based (2-4, stored 1-3), cannot be 0
because it is preserved for EndTag
44
45     'Private ReadOnly LongMatchTag As Byte = &HF8    'Could be changed to &H00, but this is more
economical
46     'Private ReadOnly NextFileTag As Byte = &HFC
47     'Private ReadOnly EndTag As Byte = 0              'Could be changed to &HF8, but this is more
economical (Number of EndTags > Number of LongMatchTags)
48
49     Private ReadOnly NearLongMatchTag As Byte = &H_84
50     Private ReadOnly FarLongMatchTag As Byte = &H_04
51     Private ReadOnly EndOfBundleTag As Byte = &H_00
52     Private ReadOnly EndOfBlockTag As Byte = &H_00
53     Private ReadOnly NextFileTag As Byte = &H_80
54
55     Private ReadOnly MaxLitLen As Integer = 16

```

```

56     Private MatchBytes As Integer = 0
57     Private MatchBits As Integer = 0
58     Private LitBits As Integer = 0
59     Private MLen As Integer = 0
60     Private MOff As Integer = 0
61
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     '//
87     'X-3          X-2          X-1          X          OFFSET (STORED AS)
88     'LENGTH (STORED AS)
89     '
90     '    SHORT:          ooooooLL      $01-$40 ($00-$3F)      $02-$04 ($01-$03)
91     '    NEAR MID:      oooooooooo 1LLLLL00      $01-$FF ($01-$FF)
92     '$02-$1F ($02-$1F)
93     '    NEAR LONG:      oooooooooo LLLLLLLL 10000100      $01-$FF ($01-$FF)      $20-$FF
94     '($20-$FF)
95     '    FAR MID:      oooooooooo HHHHHHHH 0LLLLL00      $0100-$FFFF
96     '$03-$1F ($03-$1F)
97     '    FAR LONG:      oooooooooo LLLLLLLL HHHHHHHH 00000100      $0100-$FFFF      $20-$FF ($20-$FF)
98     '
99     '    NEAR LONG MATCH:          10000100
100    '    FAR LONG MATCG:          00000100
101    '    END OF BLOCK:          00000000
102    '    NEXT FILE IN BUNDLE:          10000000
103    '    END OF BUNDLE:          00000000 10000100 (USE 00001000 instead??? far mid
length of $02 Is Not used)
104    '
105    '
106    '
107    '
108    '
109    '
110    '
111    '
112    '
113    '
114    '
115    '
116    '
117    '
118    '
119    '
120    '
121    '
122    '
123    '
124    '
125    '
126    '
127    '
128    '
129    '
130    '
131    '
132    '
133    '
134    '
135    '
136    '
137    '
138    '
139    '
140    '
141    '
142    '
143    '
144    '
145    '
146    '
147    '
148    '
149    '
150    '
151    '
152    '
153    '
154    '
155    '
156    '
157    '
158    '
159    '
160    '
161    '
162    '
163    '
164    '
165    '
166    '
167    '
168    '
169    '
170    '
171    '
172    '
173    '
174    '
175    '
176    '
177    '
178    '
179    '
180    '
181    '
182    '
183    '
184    '
185    '
186    '
187    '
188    '
189    '
190    '
191    '
192    '
193    '
194    '
195    '
196    '
197    '
198    '
199    '
200    '
201    '
202    '
203    '
204    '
205    '
206    '
207    '
208    '
209    '
210    '
211    '
212    '
213    '
214    '
215    '
216    '
217    '
218    '
219    '
220    '
221    '
222    '
223    '
224    '
225    '
226    '
227    '
228    '
229    '
230    '
231    '
232    '
233    '
234    '
235    '
236    '
237    '
238    '
239    '
240    '
241    '
242    '
243    '
244    '
245    '
246    '
247    '
248    '
249    '
250    '
251    '
252    '
253    '
254    '
255    '
256    '
257    '
258    '
259    '
260    '
261    '
262    '
263    '
264    '
265    '
266    '
267    '
268    '
269    '
270    '
271    '
272    '
273    '
274    '
275    '
276    '
277    '
278    '
279    '
280    '
281    '
282    '
283    '
284    '
285    '
286    '
287    '
288    '
289    '
290    '
291    '
292    '
293    '
294    '
295    '
296    '
297    '
298    '
299    '
300    '
301    '
302    '
303    '
304    '
305    '
306    '
307    '
308    '
309    '
310    '
311    '
312    '
313    '
314    '
315    '
316    '
317    '
318    '
319    '
320    '
321    '
322    '
323    '
324    '
325    '
326    '
327    '
328    '
329    '
330    '
331    '
332    '
333    '
334    '
335    '
336    '
337    '
338    '
339    '
340    '
341    '
342    '
343    '
344    '
345    '
346    '
347    '
348    '
349    '
350    '
351    '
352    '
353    '
354    '
355    '
356    '
357    '
358    '
359    '
360    '
361    '
362    '
363    '
364    '
365    '
366    '
367    '
368    '
369    '
370    '
371    '
372    '
373    '
374    '
375    '
376    '
377    '
378    '
379    '
380    '
381    '
382    '
383    '
384    '
385    '
386    '
387    '
388    '
389    '
390    '
391    '
392    '
393    '
394    '
395    '
396    '
397    '
398    '
399    '
400    '
401    '
402    '
403    '
404    '
405    '
406    '
407    '
408    '
409    '
410    '
411    '
412    '
413    '
414    '
415    '
416    '
417    '
418    '
419    '
420    '
421    '
422    '
423    '
424    '
425    '
426    '
427    '
428    '
429    '
430    '
431    '
432    '
433    '
434    '
435    '
436    '
437    '
438    '
439    '
440    '
441    '
442    '
443    '
444    '
445    '
446    '
447    '
448    '
449    '
450    '
451    '
452    '
453    '
454    '
455    '
456    '
457    '
458    '
459    '
460    '
461    '
462    '
463    '
464    '
465    '
466    '
467    '
468    '
469    '
470    '
471    '
472    '
473    '
474    '
475    '
476    '
477    '
478    '
479    '
480    '
481    '
482    '
483    '
484    '
485    '
486    '
487    '
488    '
489    '
490    '
491    '
492    '
493    '
494    '
495    '
496    '
497    '
498    '
499    '
500    '
501    '
502    '
503    '
504    '
505    '
506    '
507    '
508    '
509    '
510    '
511    '
512    '
513    '
514    '
515    '
516    '
517    '
518    '
519    '
520    '
521    '
522    '
523    '
524    '
525    '
526    '
527    '
528    '
529    '
530    '
531    '
532    '
533    '
534    '
535    '
536    '
537    '
538    '
539    '
540    '
541    '
542    '
543    '
544    '
545    '
546    '
547    '
548    '
549    '
550    '
551    '
552    '
553    '
554    '
555    '
556    '
557    '
558    '
559    '
560    '
561    '
562    '
563    '
564    '
565    '
566    '
567    '
568    '
569    '
570    '
571    '
572    '
573    '
574    '
575    '
576    '
577    '
578    '
579    '
580    '
581    '
582    '
583    '
584    '
585    '
586    '
587    '
588    '
589    '
590    '
591    '
592    '
593    '
594    '
595    '
596    '
597    '
598    '
599    '
600    '
601    '
602    '
603    '
604    '
605    '
606    '
607    '
608    '
609    '
610    '
611    '
612    '
613    '
614    '
615    '
616    '
617    '
618    '
619    '
620    '
621    '
622    '
623    '
624    '
625    '
626    '
627    '
628    '
629    '
630    '
631    '
632    '
633    '
634    '
635    '
636    '
637    '
638    '
639    '
640    '
641    '
642    '
643    '
644    '
645    '
646    '
647    '
648    '
649    '
650    '
651    '
652    '
653    '
654    '
655    '
656    '
657    '
658    '
659    '
660    '
661    '
662    '
663    '
664    '
665    '
666    '
667    '
668    '
669    '
670    '
671    '
672    '
673    '
674    '
675    '
676    '
677    '
678    '
679    '
680    '
681    '
682    '
683    '
684    '
685    '
686    '
687    '
688    '
689    '
690    '
691    '
692    '
693    '
694    '
695    '
696    '
697    '
698    '
699    '
700    '
701    '
702    '
703    '
704    '
705    '
706    '
707    '
708    '
709    '
710    '
711    '
712    '
713    '
714    '
715    '
716    '
717    '
718    '
719    '
720    '
721    '
722    '
723    '
724    '
725    '
726    '
727    '
728    '
729    '
730    '
731    '
732    '
733    '
734    '
735    '
736    '
737    '
738    '
739    '
740    '
741    '
742    '
743    '
744    '
745    '
746    '
747    '
748    '
749    '
750    '
751    '
752    '
753    '
754    '
755    '
756    '
757    '
758    '
759    '
760    '
761    '
762    '
763    '
764    '
765    '
766    '
767    '
768    '
769    '
770    '
771    '
772    '
773    '
774    '
775    '
776    '
777    '
778    '
779    '
780    '
781    '
782    '
783    '
784    '
785    '
786    '
787    '
788    '
789    '
790    '
791    '
792    '
793    '
794    '
795    '
796    '
797    '
798    '
799    '
800    '
801    '
802    '
803    '
804    '
805    '
806    '
807    '
808    '
809    '
810    '
811    '
812    '
813    '
814    '
815    '
816    '
817    '
818    '
819    '
820    '
821    '
822    '
823    '
824    '
825    '
826    '
827    '
828    '
829    '
830    '
831    '
832    '
833    '
834    '
835    '
836    '
837    '
838    '
839    '
840    '
841    '
842    '
843    '
844    '
845    '
846    '
847    '
848    '
849    '
850    '
851    '
852    '
853    '
854    '
855    '
856    '
857    '
858    '
859    '
860    '
861    '
862    '
863    '
864    '
865    '
866    '
867    '
868    '
869    '
870    '
871    '
872    '
873    '
874    '
875    '
876    '
877    '
878    '
879    '
880    '
881    '
882    '
883    '
884    '
885    '
886    '
887    '
888    '
889    '
890    '
891    '
892    '
893    '
894    '
895    '
896    '
897    '
898    '
899    '
900    '
901    '
902    '
903    '
904    '
905    '
906    '
907    '
908    '
909    '
910    '
911    '
912    '
913    '
914    '
915    '
916    '
917    '
918    '
919    '
920    '
921    '
922    '
923    '
924    '
925    '
926    '
927    '
928    '
929    '
930    '
931    '
932    '
933    '
934    '
935    '
936    '
937    '
938    '
939    '
940    '
941    '
942    '
943    '
944    '
945    '
946    '
947    '
948    '
949    '
950    '
951    '
952    '
953    '
954    '
955    '
956    '
957    '
958    '
959    '
960    '
961    '
962    '
963    '
964    '
965    '
966    '
967    '
968    '
969    '
970    '
971    '
972    '
973    '
974    '
975    '
976    '
977    '
978    '
979    '
980    '
981    '
982    '
983    '
984    '
985    '
986    '
987    '
988    '
989    '
990    '
991    '
992    '
993    '
994    '
995    '
996    '
997    '
998    '
999    '
1000    '
1001    '
1002    '
1003    '
1004    '
1005    '
1006    '
1007    '
1008    '
1009    '
1010    '
1011    '
1012    '
1013    '
1014    '
1015    '
1016    '
1017    '
1018    '
1019    '
1020    '
1021    '
1022    '
1023    '
1024    '
1025    '
1026    '
1027    '
1028    '
1029    '
1030    '
1031    '
1032    '
1033    '
1034    '
1035    '
1036    '
1037    '
1038    '
1039    '
1040    '
1041    '
1042    '
1043    '
1044    '
1045    '
1046    '
1047    '
1048    '
1049    '
1050    '
1051    '
1052    '
1053    '
1054    '
1055    '
1056    '
1057    '
1058    '
1059    '
1060    '
1061    '
1062    '
1063    '
1064    '
1065    '
1066    '
1067    '
1068    '
1069    '
1070    '
1071    '
1072    '
1073    '
1074    '
1075    '
1076    '
1077    '
1078    '
1079    '
1080    '
1081    '
1082    '
1083    '
1084    '
1085    '
1086    '
1087    '
1088    '
1089    '
1090    '
1091    '
1092    '
1093    '
1094    '
1095    '
1096    '
1097    '
1098    '
1099    '
1100    '
1101    '
1102    '
1103    '
1104    '
1105    '
1106    '
1107    '
1108    '
1109    '
1110    '
1111    '
1112    '
1113    '
1114    '
1115    '
1116    '
1117    '
1118    '
1119    '
1120    '
1121    '
1122    '
1123    '
1124    '
1125    '
1126    '
1127    '
1128    '
1129    '
1130    '
1131    '
1132    '
1133    '
1134    '
1135    '
1136    '
1137    '
1138    '
1139    '
1140    '
1141    '
1142    '
1143    '
1144    '
1145    '
1146    '
1147    '
1148    '
1149    '
1150    '
1151    '
1152    '
1153    '
1154    '
1155    '
1156    '
1157    '
1158    '
1159    '
1160    '
1161    '
1162    '
1163    '
1164    '
1165    '
1166    '
1167    '
1168    '
1169    '
1170    '
1171    '
1172    '
1173    '
1174    '
1175    '
1176    '
1177    '
1178    '
1179    '
1180    '
1181    '
1182    '
1183    '
1184    '
1185    '
1186    '
1187    '
1188    '
1189    '
1190    '
1191    '
1192    '
1193    '
1194    '
1195    '
1196    '
1197    '
1198    '
1199    '
1200    '
1201    '
1202    '
1203    '
1204    '
1205    '
1206    '
1207    '
1208    '
1209    '
1210    '
1211    '
1212    '
1213    '
1214    '
1215    '
1216    '
1217    '
1218    '
1219    '
1220    '
1221    '
1222    '
1223    '
1224    '
1225    '
1226    '
1227    '
1228    '
1229    '
1230    '
1231    '
1232    '
1233    '
1234    '
1235    '
1236    '
1237    '
1238    '
1239    '
1240    '
1241    '
1242    '
1243    '
1244    '
1245    '
1246    '
1247    '
1248    '
1249    '
1250    '
1251    '
1252    '
1253    '
1254    '
1255    '
1256    '
1257    '
1258    '
1259    '
1260    '
1261    '
1262    '
1263    '
1264    '
1265    '
1266    '
1267    '
1268    '
1269    '
1270    '
1271    '
1272    '
1273    '
1274    '
1275    '
1276    '
1277    '
1278    '
1279    '
1280    '
1281    '
1282    '
1283    '
1284    '
1285    '
1286    '
1287    '
1288    '
1289    '
1290    '
1291    '
1292    '
1293    '
1294    '
1295    '
1296    '
1297    '
1298    '
1299    '
1300    '
1301    '
1302    '
1303    '
1304    '
1305    '
1306    '
1307    '
1308    '
1309    '
1310    '
1311    '
1312    '
1313    '
1314    '
1315    '
1316    '
1317    '
1318    '
1319    '
1320    '
1321    '
1322    '
1323    '
1324    '
1325    '
1326    '
1327    '
1328    '
1329    '
1330    '
1331    '
1332    '
1333    '
1334    '
1335    '
1336    '
1337    '
1338    '
1339    '
1340    '
1341    '
1342    '
1343    '
1344    '
1345    '
1346    '
1347    '
1348    '
1349    '
1350    '
1351    '
1352    '
1353    '
1354    '
1355    '
1356    '
1357    '
1358    '
1359    '
1360    '
1361    '
1362    '
1363    '
1364    '
1365    '
1366    '
1367    '
1368    '
1369    '
1370    '
1371    '
1372    '
1373    '
1374    '
1375    '
1376    '
1377    '
1378    '
1379    '
1380    '
1381    '
1382    '
1383    '
1384    '
1385    '
1386    '
1387    '
1388    '
1389    '
1390    '
1391    '
1392    '
1393    '
1394    '
1395    '
1396    '
1397    '
1398    '
1399    '
1400    '
1401    '
1402    '
1403    '
1404    '
1405    '
1406    '
1407    '
1408    '
1409    '
1410    '
1411    '
1412    '
1413    '
1414    '
1415    '
1416    '
1417    '
1418    '
1419    '
1420    '
1421    '
1422    '
1423    '
1424    '
1425    '
1426    '
1427    '
1428    '
1429    '
1430    '
1431    '
1432    '
1433    '
1434    '
1435    '
1436    '
1437    '
1438    '
1439    '
1440    '
1441    '
1442    '
1443    '
1444    '
1445    '
1446    '
1447    '
1448    '
1449    '
1450    '
1451    '
1452    '
1453    '
1454    '
1455    '
1456    '
1457    '
1458    '
1459    '
1460    '
1461    '
1462    '
1463    '
1464    '
1465    '
1466    '
1467    '
1468    '
1469    '
1470    '
1471    '
1472    '
1473    '
1474    '
1475    '
1476    '
1477    '
1478    '
1479    '
1480    '
1481    '
1482    '
1483    '
1484    '
1485    '
1486    '
1487    '
1488    '
1489    '
1490    '
1491    '
1492    '
1493    '
1494    '
1495    '
1496    '
1497    '
1498    '
1499    '
1500    '
1501    '
1502    '
1503    '
1504    '
1505    '
1506    '
1507    '
1508    '
1509    '
1510    '
1511    '
1512    '
1513    '
1514    '
1515    '
1516    '
1517    '
1518    '
1519    '
1520    '
1521    '
1522    '
1523    '
1524    '
1525    '
1526    '
1527    '
1528    '
1529    '
1530    '
1531    '
1532    '
1533    '
1534    '
1535    '
1536    '
1537    '
1538    '
1539    '
1540    '
1541    '
1542    '
1543    '
1544    '
1545    '
1546    '
1547    '
1548    '
1549    '
1550    '
1551    '
1552    '
1553    '
1554    '
1555    '
1556    '
1557    '
1558    '
1559    '
1560    '
1561    '
1562    '
1563    '
1564    '
1565    '
1566    '
1567    '
1568    '
1569    '
1570    '
1571    '
1572    '
1573    '
1574    '
1575    '
1576    '
1577    '
1578    '
1579    '
1580    '
1581    '
1582    '
1583    '
1584    '
1585    '
1586    '
1587    '
1588    '
1589    '
1590    '
1591    '
1592    '
1593    '
1594    '
1595    '
1596    '
1597    '
1598    '
1599    '
1600    '
1601    '
1602    '
1603    '
1604    '
1605    '
1606    '
1607    '
1608    '
1609    '
1610    '
1611    '
1612    '
1613    '
1614    '
1615    '
1616    '
1617    '
1618    '
1619    '
1620    '
1621    '
1622    '
1623    '
1624    '
1625    '
1626    '
1627    '
1628    '
1629    '
1630    '
1631    '
1632    '
1633    '
1634    '
1635    '
1636    '
1637    '
1638    '
1639    '
1640    '
1641    '
1642    '
1643    '
1644    '
1645    '
1646    '
1647    '
1648    '
1649    '
1650    '
1651    '
1652    '
1653    '
1654    '
1655    '
1656    '
1657    '
1658    '
1659    '
1660    '
1661    '
1662    '
1663    '
1664    '
1665    '
1666    '
1667    '
1668    '
1669    '
1670    '
1671    '
1672    '
1673    '
1674    '
1675    '
1676    '
1677    '
1678    '
1679    '
1680    '
1681    '
1682    '
1683    '
1684    '
1685    '
1686    '
1687    '
1688    '
1689    '
1690    '
1691    '
1692    '
1693    '
1694    '
1695    '
1696    '
1697    '
1698    '
1699    '
1700    '
1701    '
1702    '
1703    '
1704    '
1705    '
1706    '
1707    '
1708    '
1709    '
1710    '
1711    '
1712    '
1713    '
1714    '
1715    '
1716    '
1717    '
1718    '
1719    '
1720    '
1721    '
1722    '
1723    '
1724    '
1725    '
1726    '
1727    '
1728    '
1729    '
1730    '
1731    '
1732    '
1733    '
1734    '
1735    '
1736    '
1737    '
1738    '
1739    '
1740    '
1741    '
1742    '
1743    '
1744    '
1745    '
1746    '
1747    '
1748    '
1749    '
1750    '
1751    '
1752    '
1753    '
1754    '
1755    '
1756    '
1757    '
1758    '
1759    '
1760    '
1761    '
1762    '
1763    '
1764    '
1765    '
1766    '
1767    '
1768    '
1769    '
1770    '
1771    '
1772    '
1773    '
1774    '
1775    '
1776    '
1777    '
1778    '
1779    '
1780    '
1781    '
1782    '
1783    '
1784    '
1785    '
1786    '
1787    '
1788    '
1789    '
1790    '
1791    '
1792    '
1793    '
1794    '
1795    '
1796    '
1797    '
1798    '
1799    '
1800    '
1801    '
1802    '
1803    '
1804    '
1805    '
1806    '
1807    '
1808    '
1809    '
1810    '
1811    '
1812    '
1813    '
1814    '
1815    '
1816    '
1817    '
1818    '
1819    '
1820    '
1821    '
1822    '
1823    '
1824    '
1825    '
1826    '
1827    '
1828    '
1829    '
1830    '
1831    '
1832    '
1833    '
1834    '
1835    '
1836    '
1837    '
1838    '
1839    '
1840    '
1841    '
1842    '
1843    '
1844    '
1845    '
1846    '
1847    '
1848    '
1849    '
1850    '
1851    '
1852    '
1853    '
1854    '
1855    '
1856    '
1857    '
1858    '
1859    '
1860    '
1861    '
1862    '
1863    '
1864    '
1865    '
1866    '
1867    '
1868    '
1869    '
1870    '
1871    '
1872    '
1873    '
1874    '
1875    '
1876    '
1877    '
1878    '
1879    '
1880    '
1881    '
1882    '
1883
```

```

110
111     MaxOffset = If(Packer = PackerTypes.Faster, MaxNearOffset, MaxNearOffset * 3)
112
113     Prg = PN
114     FileUnderIO = FUIO
115     PrgAdd = Convert.ToInt32(FA, 16)
116     PrgLen = Prg.Length
117
118     ReDim SL(PrgLen - 1), SO(PrgLen - 1), NL(PrgLen - 1), NO(PrgLen - 1), FL(PrgLen - 1),
FO(PrgLen - 1)
119     ReDim FFL(PrgLen - 1), FFO(PrgLen - 1)
120
121     ReDim Seq(PrgLen)          'This is actually one element more in the array, to have starter
element with 0 values
122
123     With Seq(0)                'Initialize first element of sequence - WAS Seq(1)!!!
124         '.Len = 0              '1 Literal byte, Len is 0 based
125         '.Off = 0              'Offset=0 -> literal sequence, Off is 1 based
126         '.TotalBits = 10      'LitLen bit + 8 bits, DO NOT CHANGE IT TO 9!!!
127     End With
128
129     '-----
-----
130     'SEARCH REFERENCE FILE FOR FAR MATCHES
131     '-----
-----
132
133     CurrentFileIndex = FileIndex
134
135     If Packer = PackerTypes.Better Then
136         For I As Integer = 0 To VFiles.Count - 1
137             ReferenceFile = VFiles(I).ToArray
138             ReferenceFileStart = Convert.ToInt32(VFileAddrA(I), 16)
139
140             SearchVirtualFile(PrgLen - 1, I, ReferenceFileStart + ReferenceFile.Length - 1,
ReferenceFileStart + 1)
141         Next
142
143         For I As Integer = 0 To PartialFileIndex - 1
144             ReferenceFile = Prgs(I).ToArray
145             ReferenceFileStart = Convert.ToInt32(FileAddrA(I), 16)
146
147             SearchReferenceFile(PrgLen - 1, I, ReferenceFileStart + ReferenceFile.Length - 1,
ReferenceFileStart + 1)
148         Next
149
150         If PartialFileIndex > -1 Then
151             ReferenceFile = Prgs(PartialFileIndex).ToArray
152             ReferenceFileStart = Convert.ToInt32(FileAddrA(PartialFileIndex), 16)
153
154             'Search partial file from offset to end of file
155             SearchReferenceFile(PrgLen - 1, PartialFileIndex, ReferenceFileStart +
ReferenceFile.Length - 1, ReferenceFileStart + PartialFileOffset + 1)
156         End If
157     End If
158
159     '-----
-----
160     'CALCULATE BEST SEQUENCE
161     '-----
-----
162

```

```

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 NEEDED
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

```

```

213 'RefMaxAddress, RefMinAddress = absolute addresses within reference file
214 If FileIOA(RefIndex) = False Then
215     'Reference file is not under I/O, check if it is IN I/O
216     If ((RefMinAddress >= &HD000) AndAlso (RefMinAddress < &HE000)) OrElse
217         ((RefMaxAddress >= &HD000) AndAlso (RefMaxAddress < &HE000)) Then
218         'This reference file is IN the I/O registers, SKIP anything between $d000-$dfff
219         If RefMinAddress + 1 < &HCFFF Then
220             'Search from start to $cfff
221             FindFarMatches(PrgMaxIndex, 1, &HCFFF, RefMinAddress + 1)
222         End If
223         If RefMaxAddress > &HE000 Then
224             'Search from $e000 to end of file
225             FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, &HE000 + 1)
226         End If
227     Else
228         FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, RefMinAddress + 1)
229     End If
230 Else
231     FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, RefMinAddress + 1)
232 End If
233
234 End Sub
235
236 Private Sub SearchVirtualFile(PrgMaxIndex As Integer, RefIndex As Integer, RefMaxAddress As
Integer, RefMinAddress As Integer)
237     'PrgMaxIndex = relative address within Prg
238     'RefMaxAddress, RefMinAddress = absolute addresses within reference file
239     If VFileIOA(RefIndex) = False Then
240         'Reference file is not under I/O, check if it is IN I/O
241         If ((RefMinAddress >= &HD000) AndAlso (RefMinAddress < &HE000)) OrElse
242             ((RefMaxAddress >= &HD000) AndAlso (RefMaxAddress < &HE000)) Then
243             'This reference file is IN the I/O registers, SKIP anything between $d000-$dfff
244             If RefMinAddress + 1 < &HCFFF Then
245                 'Search from start to $cfff
246                 FindFarMatches(PrgMaxIndex, 1, &HCFFF, RefMinAddress + 1)
247             End If
248             If RefMaxAddress > &HE000 Then
249                 'Search from $e000 to end of file
250                 FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, &HE000 + 1)
251             End If
252         Else
253             FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, RefMinAddress + 1)
254         End If
255     Else
256         FindFarMatches(PrgMaxIndex, 1, RefMaxAddress, RefMinAddress + 1)
257     End If
258
259 End Sub
260
261 Private Sub FindMatches(SeqHighestIndex As Integer, SeqLowestIndex As Integer, Optional FirstRun
As Boolean = False)
262     If DoOnErr Then On Error GoTo Err
263
264     Dim Max0, MaxLL, MaxSL As Integer
265
266     '-----
267     'FIND LONGEST SHORT AND NEAR MATCHES FOR EACH POSITION, AND FAR MATCHES WITH OFFSET < MAX.
1024
268     '-----
269
270     '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 0 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 (0 <= 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) = 0 'Keep Offset 1-based
312                     End If
313                 #If AllowShortMidMatches Then
314                     If (0 <= MaxNearOffset) AndAlso (NL(Pos) < L) Then 'Allow short
(2-byte) Mid Matches
315                         NL(Pos) = L
316                         NO(Pos) = 0
317                     End If
318                 #Else
319                     If (0 <= MaxNearOffset) AndAlso (NL(Pos) < L) AndAlso (L > 2) Then
'Skip short (2-byte) Mid Matches
320                         NL(Pos) = L
321                         NO(Pos) = 0
322                     End If
323                 #End If
324                 If (0 > MaxNearOffset) AndAlso (FL(Pos) < L) AndAlso (L > 2) Then

```

```

325                                     FL(Pos) = L
326                                     FO(Pos) = 0
327                                     End If
328                                     End If
329                                     Exit For
330                                 End If
331                            Next
332                            'If both short and long matches maxed out, we can leave the loop and go to the
next Prg position
333                                If (NL(Pos) = MaxLL) AndAlso (SL(Pos) = MaxSL) Then
334                                    Exit For
335                                End If
336                            End If
337                        Next
338                    End If
339                Next
340
341            Exit Sub
342
343        Err:
344            ErrCode = Err.Number
345            MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
346
347            End Sub
348
349        Private Sub FindFarMatches(SeqMaxIndex As Integer, SeqMinIndex As Integer, RefMaxAddress As
Integer, RefMinAddress As Integer)
350            If DoOnErr Then On Error GoTo Err
351            'SeqMqxIndex,SeqMinIndex = relative address within Prg
352            'RefMaxAddress, RefMinAddress = absolute addresses within reference file
353
354            Dim Max0, MaxLL, MaxSL As Integer
355
356            If FileUnderIO = False Then
357                If (PrgAdd + SeqMinIndex >= &HD000) AndAlso (PrgAdd + SeqMinIndex <= &HFFFF) Then
358                    If PrgAdd + SeqMaxIndex <= &HFFFF Then
359                        'The entire file segment is under I/O -> skip it
360                        Exit Sub
361                    Else
362                        'Not the entire file segment is under I/O -> only check the part that is not
363                        SeqMinIndex = &HE000 - PrgAdd
364                        End If
365                    ElseIf (PrgAdd + SeqMaxIndex >= &HD000) AndAlso (PrgAdd + SeqMaxIndex <= &HFFFF) Then
366                        SeqMaxIndex = &HCFFF - PrgAdd
367                        End If
368                    End If
369                '-----
370                'FIND LONGEST SHORT AND NEAR MATCHES FOR EACH POSITION, AND FAR MATCHES WITH OFFSET < MAX.
1024
371                '-----
372
373                Dim RefMinAddressIndex As Integer = RefMinAddress - ReferenceFileStart
374                Dim RefMaxAddressIndex As Integer = RefMaxAddress - ReferenceFileStart
375
376                Dim OffsetBase = If(ReferenceFileStart >= PrgAdd, ReferenceFileStart - PrgAdd,
ReferenceFileStart + &H10000 - PrgAdd)
377
378                'Pos = Min>0 to Max value, direction of execution is arbitrary (could be Max to Min>0 Step -1)
379                For Pos As Integer = SeqMinIndex To SeqMaxIndex 'Pos cannot be 0, Prg(0) is always
literal as it is always 1 byte left

```

```

380 'Offset goes from 1 to max offset (cannot be 0)
381 'Max0 = Math.Min(MaxNearOffset, SeqMaxIndex - Pos) 'Max0=256 or less
382 'Match length goes from 1 to max length
383 For 0 As Integer = RefMinAddressIndex To RefMaxAddressIndex
384 'Check if first byte matches at offset, if not go to next offset
385 If Prg(Pos) = ReferenceFile(0) Then
386     MaxLL = Math.Min(Math.Min(Pos + 1, MaxLongLen), 0 - RefMinAddressIndex + 1)
'MaxLL = 255 or less
387     For L As Integer = 1 To MaxLL 'L=1 to 255 or less
388         If (L = MaxLL) OrElse (Prg(Pos - L) <> ReferenceFile(0 - L)) Then
389             'Find the first position where there is NO match -> this will give us the
absolute length of the match
390             'L=MatchLength + 1 here
391             If L > 2 Then
392                 If ((0 - Pos) > MaxNearOffset) AndAlso (FL(Pos) < L) Then
393                     FFL(Pos) = L
394                     FFO(Pos) = 0 - Pos + OffsetBase
395                 End If
396             End If
397             Exit For
398         End If
399     Next
400     'If far matches maxed out, we can leave the loop and go to the next Prg position
401     If FFL(Pos) = MaxLL Then
402         Exit For
403     End If
404 End If
405 Next
406 Next
407
408 Exit Sub
409 Err:
410     ErrCode = Err.Number
411     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
412
413 End Sub
414
415 Private Sub CalcBestSequence(SeqHighestIndex As Integer, SeqLowestIndex As Integer, Optional
FirstRun As Boolean = False)
416     If DoOnError Then On Error GoTo Err
417
418     FindMatches(SeqHighestIndex, SeqLowestIndex, FirstRun)
419
420     '-----
421     'FIND BEST SEQUENCE FOR EACH POSITION
422     '-----
423
424     For Pos As Integer = SeqLowestIndex To SeqHighestIndex
425
426         'Start with second element, first has been initialized above
427         Seq(Pos + 1).TotalBits = &HFFFFFF
428         'Max block size=100 = $10000 bytes = $80000 bits, make default larger than this
429
430         If (FL(Pos) <> 0) OrElse (FFL(Pos) <> 0) Then
431             If FL(Pos) >= FFL(Pos) Then
432                 CheckMatchSeq(FL(Pos), FO(Pos), Pos)
433             ElseIf Pos < SeqHighestIndex Then 'The last byte of the block MUST be a literal, we
can't use Far Matches there
434                 'If a reference is under I/O then this block also must be under I/O to be able to
copy the reference

```



```

435 &HE000)) OrElse
436         '((PrgAdd + Pos + FFO(Pos) - FFL(Pos) >= &HD000) AndAlso (PrgAdd + Pos + FFO(Pos)
- FFL(Pos) < &HE000)) Then
437         'BlockUnderIO = 1
438         'End If
439         CheckMatchSeq(FFL(Pos), FFO(Pos), Pos)
440     End If
441 End If
442 If NL(Pos) > 0 Then
443     CheckMatchSeq(NL(Pos), NO(Pos), Pos)
444 End If
445 If SL(Pos) > 0 Then
446     CheckMatchSeq(SL(Pos), SO(Pos), Pos)
447 End If
448
449 CheckLitSeq(Pos)
450
451 Next
452
453 Exit Sub
454 Err:
455     ErrCode = Err.Number
456     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
457
458 End Sub
459
460 Private Sub CheckMatchSeq(SeqLen As Integer, SeqOff As Integer, Pos As Integer)
461     If DoOnError Then On Error GoTo Err
462     Dim TotBits As Integer
463
464     'If this is a far match then min len = 3, otherwise min len = 2
465     Dim MinLen As Integer = If(SeqOff > MaxNearOffset, 3, 2)
466
467     'Check all possible lengths
468     For L As Integer = SeqLen To MinLen Step -1
469         'Calculate MatchBits
470
471         If (L <= MaxShortLen) AndAlso (SeqOff <= MaxShortOffset) Then
472             MatchBits = 9
473         ElseIf (L <= MaxMidLen) AndAlso (SeqOff <= MaxNearOffset) Then
474             MatchBits = 17
475         ElseIf (L > MaxMidLen) AndAlso (SeqOff > MaxNearOffset) Then
476             MatchBits = 33
477         Else
478             MatchBits = 25
479         End If
480
481         'Calculate total bit count, independently of nibble status
482         TotBits = Seq(Pos + 1 - L).TotalBits + MatchBits
483
484         With Seq(Pos + 1)
485             'See if total bit count is better than best version
486             If TotBits < .TotalBits Then
487                 'If better, update best version
488                 .Len = L           'MatchLen is 1 based
489                 .Off = SeqOff      'Off is 1 based
490                 .Nibbles = Seq(Pos + 1 - L).Nibbles
491                 .TotalBits = TotBits
492             End If
493         End With

```

```

494     Next
495
496     Exit Sub
497 Err:
498     ErrCode = Err.Number
499     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
500
501 End Sub
502
503 Private Sub CheckLitSeq(Pos As Integer)
504     If DoOnError Then On Error GoTo Err
505
506     Dim TotBits As Integer
507
508     'Continue previous Lit sequence or start new sequence
509     LitCnt = If(Seq(Pos).Off = 0, Seq(Pos).Len, -1)
510
511     'Calculate literal bits for a presumptive LitCnt+1 value
512     LitBits = Int((LitCnt + 1) / MaxLitPerBlock) * 13
513     Select Case (LitCnt + 1) Mod MaxLitPerBlock
514         Case 0
515             LitBits += 1                                'Lits = 0    1 literal, 1 bit
516         Case 1 To MaxLitLen - 1
517             LitBits += 5                                'Lits = 1-15 2-16 literals, 5 bits
518         Case Else
519             LitBits += 13                               'Lits = 16-250    17-251 literals, 13 bits
520     End Select
521
522     'LITERALS ARE ALWAYS FOLLOWED BY MATCHES, SO TYPE SELECTOR BIT IS NOT NEEDED AFTER LITERALS AT
ALL
523
524     TotBits = Seq(Pos - LitCnt - 1).TotalBits + LitBits + ((LitCnt + 2) * 8)
525
526     With Seq(Pos + 1)
527         'See if total bit count is less than best version
528         If TotBits < .TotalBits Then
529             'and save it to sequence at Pos+1 (position is 1 based)
530             .Len = LitCnt + 1          'LitCnt is 0 based, LitLen is 0 based
531             .Off = 0                  'An offset of 0 marks a literal sequence, match offset is 1
based
532             .Nibbles = Seq(Pos - (LitCnt + 1)).Nibbles + If(LitBits > 1, 1, 0)
533             .TotalBits = TotBits
534         End If
535     End With
536
537     Exit Sub
538 Err:
539     ErrCode = Err.Number
540     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
541
542 End Sub
543
544 Private Sub Pack()
545     If DoOnError Then On Error GoTo Err
546
547     'Packing is done backwards
548
549     Dim BufferFull As Boolean
550
551     SI = PrgLen - 1
552     StartPtr = SI

```

```

553     If PrgAdd + SI = &HE5FF Then
554         SI += 0
555     End If
556
557     'Cycles += 61 'From NextFile label
558     'BitStreamBytes = 0
559
560 Restart:
561     Do
562
563         If Seq(SI + 1).Off = 0 Then
564             '-----
565             'Literal sequence
566             '-----
567             LitCnt = Seq(SI + 1).Len           'LitCnt is 0 based
568             LitSI = SI
569             MLen = 0                           'Reset MLen - this is needed for accurate bit
570 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                 '-----

```

```

614 'Far Long Match - 4 match bytes + 0/1 match bit
615 '-----
616
617 If (((PrgAdd + SI + MOff) Mod &H10000 >= &HD000) AndAlso ((PrgAdd + SI + MOff) Mod
&H10000 < &HE000)) OrElse
618 (((PrgAdd + SI + MOff - MLen + 1) Mod &H10000 >= &HD000) AndAlso ((PrgAdd + SI +
MOff - MLen + 1) Mod &H10000 < &HE000)) Then
619 If (PrgAdd + SI >= &HD000) AndAlso (PrgAdd + SI < &HE000 AndAlso (FileUnderIO
= False)) OrElse
620 (PrgAdd + SI - MLen + 1 >= &HD000) AndAlso (PrgAdd + SI - MLen + 1 <
&HE000 AndAlso (FileUnderIO = False)) Then
621 Else
622 ReferenceUnderIO = 1
623 End If
624 End If
625
626 If SequenceFits(MatchBytes + LitCnt + 1, MatchBits + CalcLitBits(LitCnt),
Math.Max(CheckIO(SI - MLen + 1), ReferenceUnderIO)) Then
627 AddLitSequence()
628 'Add far long match
629 AddFarLongMatch()
630 Else
631 MLen = MaxMidLen
632 BufferFull = True 'Buffer if full, we will need to close it
633 GoTo Check3Bytes
634 End If
635 ElseIf MatchBytes = 3 Then
636 '-----
637 'Far Mid Match or Near Long Match - 3 match bytes + 0/1 match bit
638 '-----
639
640 If MOff > MaxNearOffset Then
641 If (((PrgAdd + SI + MOff) Mod &H10000 >= &HD000) AndAlso ((PrgAdd + SI + MOff)
Mod &H10000 < &HE000)) OrElse
642 (((PrgAdd + SI + MOff - MLen + 1) Mod &H10000 >= &HD000) AndAlso ((PrgAdd + SI
+ MOff - MLen + 1) Mod &H10000 < &HE000)) Then
643 If (PrgAdd + SI >= &HD000) AndAlso (PrgAdd + SI < &HE000 AndAlso
(FileUnderIO = False)) OrElse
644 (PrgAdd + SI - MLen + 1 >= &HD000) AndAlso (PrgAdd + SI - MLen + 1 <
&HE000 AndAlso (FileUnderIO = False)) Then
645 Else
646 ReferenceUnderIO = 1
647 End If
648 End If
649 End If
650
651 Check3Bytes: If SequenceFits(MatchBytes + LitCnt + 1, MatchBits + CalcLitBits(LitCnt),
Math.Max(CheckIO(SI - MLen + 1), ReferenceUnderIO)) Then
652 AddLitSequence()
653 'Add long match
654 If MOff > MaxNearOffset Then
655 'Add far mid match
656 AddFarMidMatch()
657 Else
658 'Add near long match
659 AddNearLongMatch()
660 End If
661 Else
662 MLen = MaxMidLen
663 BufferFull = True 'Buffer if full, we will need to close it
664 GoTo Check2Bytes
665 End If
666 ElseIf MatchBytes = 2 Then
667 '-----

```

```

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     Then If SequenceFits(1 + LitCnt + 1, CalcLitBits(LitCnt + 1), CheckIO(SI - LitCnt))
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 DoOnError 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 Private Function CalcLitBits(Lits As Integer) As Integer 'LitCnt is 0 based
782     If DoOnErr Then On Error GoTo Err
783
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 sequences, so
comparison 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

```

```

839     If BitsFree - BitsToAdd < 0 Then BytesToAdd += 1
840
841     If BytesFree >= BytesToAdd Then
842         'Check if sequence will fit within block size limits
843         SequenceFits = True
844         'Data will fit
845         If (BlockUnderIO = 0) AndAlso (SequenceUnderIO = 1) Then
846             'This is the first byte in the block that will go UIO, so lets update the buffer to
include the IO flag
847             For I As Integer = BytePtr To AdHiPos 'Move all data to the left in buffer,
including AdHi
848                 Buffer(I - 1) = Buffer(I)
849             Next
850             Buffer(AdHiPos) = 0 'IO Flag to previous AdHi Position
851             BytePtr -= 1 'Update BytePtr to next empty position in
buffer
852             If NibblePtr > 0 Then NibblePtr -= 1 'Only update Nibble Pointer if it does not
point to Byte(0)
853             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
854             AdHiPos -= 1 'Update AdHi Position in Buffer
855             BlockUnderIO = 1 'Set BlockUnderIO Flag
856         End If
857     Else
858         SequenceFits = False
859     End If
860
861     Exit Function
862 Err:
863     ErrCode = Err.Number
864     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
865
866     SequenceFits = False
867
868 End Function
869
870 Private Sub AddMatchBit()
871     If DoOnErr Then On Error GoTo Err
872
873     If LitCnt = -1 Then
874         AddBits(MatchSelector, 1) 'Last Literal Length was -1, we need the Match selector bit
(1)
875         'Cycles += 10
876     End If
877
878     LitCnt = -1
879
880     Exit Sub
881 Err:
882     ErrCode = Err.Number
883     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
884
885 End Sub
886
887 Private Sub AddFarLongMatch()
888     If DoOnErr Then On Error GoTo Err
889
890     '//
(STORED AS)          LENGTH (STORED AS)          X-3          X-2          X-1          X          OFFSET
891     '//          FAR LONG:  00000000 LLLLLLLL HHHHHHHH 00000100          $0100-$FFFF          $20-$FF
($20-$FF)
892

```



```

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 DoOnError Then On Error GoTo Err
917
918         '//
(STORED AS)      LENGTH (STORED AS)      X-3      X-2      X-1      X      OFFSET
919         '//      NEAR LONG:              00000000 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 DoOnError Then On Error GoTo Err
943
944         '//
(STORED AS)      LENGTH (STORED AS)      X-3      X-2      X-1      X      OFFSET
945         '//      FAR MID:                00000000 HHHHHHHH 0LLLLL00      $0101-$FFFF ($0100-$FFFE)
$03-$1F ($03-$1F)
946
947         TotMatch += 1
948
949         AddMatchBit()

```

```
950         Buffer(BytePtr) = MLen * 4           'Length of match (#$02-$1f))
951         Buffer(BytePtr - 1) = Int((MOff - 1) / 256)
952         Buffer(BytePtr - 2) = (MOff - 1) Mod 256
953         BytePtr -= 3
954
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     '///
971     (STORED AS)      LENGTH (STORED AS)      X-3      X-2      X-1      X      OFFSET
972     '///      NEAR MID:      ooooooooo 1LLLLL00      $01-$100 ($00-$FF)
973     $02-$1F ($02-$1F)
974
975     TotMatch += 1
976
977     AddMatchBit()
978
979     Buffer(BytePtr) = &H_80 + (MLen * 4)           'Length of match (#$02-$1f))
980     Buffer(BytePtr - 1) = MOff - 1           'Offset (1-256, stored as 0-255)
981     BytePtr -= 2
982
983     'Cycles += 67 + (15 * MLen)
984
985     TotNMM += 1
986
987 Exit Sub
988 Err:
989     ErrCode = Err.Number
990     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
991
992 End Sub
993
994 Private Sub AddShortMatch()
995     If DoOnErr Then On Error GoTo Err
996
997     '///
998     (STORED AS)      LENGTH (STORED AS)      X-3      X-2      X-1      X      OFFSET
999     '///      SHORT:      ooooooLL      $01-$40 ($00-$3F)      $02-$04
1000     ($01-$03)
1001
1002     TotMatch += 1
1003
1004     AddMatchBit()
1005
1006     Buffer(BytePtr) = ((MOff - 1) * 4) + (MLen - 1)
1007     BytePtr -= 1
1008
1009     'Cycles += 54 + (15 * MLen)
```

```

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

```

```

1066 Else
1067     FirstLitOfBlock = False
1068 End If
1069
1070 Select Case Lits
1071     Case 0
1072         AddBits(0, 1) 'Add Literal Length Selector 0 - read no more bits
1073         'Cycles += 52
1074         Case 1 To MaxLitLen - 1
1075             AddBits(1, 1) 'Add Literal Length Selector 1 - read 4 more bits
1076             AddNibble(Lits) 'Add Literal Length: 01-0f, 4 bits (0001-1111)
1077             'Cycles += 62 + ((Lits + 1) * 15)
1078             Case MaxLitLen
1079                 AddBits(1, 1) 'Add Literal Length Selector 1 - read 4 more bits
1080                 AddNibble(0) 'Add Literal Length: 0, 4 bits (0000) - we will have a
longer literal sequence
1081                 'Cycles += 72 + ((Lits + 1) * 15)
1082         End Select
1083
1084         'DO NOT RESET LitCnt HERE!!!
1085
1086     Exit Sub
1087 Err:
1088     ErrCode = Err.Number
1089     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
1090
1091 End Sub
1092 Private Sub AddNibble(Bit As Integer)
1093     If DoOnError Then On Error GoTo Err
1094
1095     If NibblePtr = 0 Then
1096         NibblePtr = BytePtr
1097         BytePtr -= 1
1098         Buffer(NibblePtr) = Bit
1099     Else
1100         Buffer(NibblePtr) += Bit * 16
1101         NibblePtr = 0
1102     End If
1103
1104     Exit Sub
1105 Err:
1106     ErrCode = Err.Number
1107     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
1108
1109 End Sub
1110
1111 Private Sub AddBits(Bit As Integer, BCnt As Byte)
1112     If DoOnError Then On Error GoTo Err
1113
1114     For I As Integer = BCnt - 1 To 0 Step -1
1115         If BitPos < 0 Then
1116             BitPos += 8
1117             BitsLeft = 8
1118             BitPtr = BytePtr 'New BitPtr pos
1119             BytePtr -= 1 'and BytePtr pos
1120             'BitStreamBytes += 1 'Number of bitstream bytes in buffer
1121         End If
1122         If (Bit And 2 ^ I) <> 0 Then
1123             Buffer(BitPtr) = Buffer(BitPtr) Or 2 ^ BitPos
1124         End If

```

```

1125 DecBitPos:
1126     BitPos -= 1
1127     BitsLeft -= 1
1128     If BitPos = 0 Then
1129         If (Buffer(BitPtr) Mod 2 = 1) Then
1130             BitPos = -1
1131             BitsLeft = 0
1132         End If
1133     End If
1134     'Very first BitPtr in buffer has a 1 in BitPos=0 (Token Bit) -> Skip It!!!
1135     'If (BitPtr = 0) And (BitPos = 0) Then BitPos = -1
1136 Next
1137
1138     'MsgBox(BufferCnt.ToString + vbNewLine + BitPtr.ToString + vbNewLine + BitPos.ToString +
vbNewLine + BitsLeft.ToString)
1139
1140 Exit Sub
1141 Err:
1142     ErrCode = Err.Number
1143     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
1144
1145 End Sub
1146
1147 Public Function CloseBuffer() As Boolean
1148     If DoOnErr Then On Error GoTo Err
1149
1150     CloseBuffer = True
1151
1152     'Buffer(BytePtr) = EndTag                'Not needed, byte 0 will be overwritten to EndTag during
loading
1153     AddMatchBit()
1154
1155     'Cycles += 19
1156     'Cycles += BitStreamBytes * 14
1157
1158     BlockCnt += 1
1159     BufferCnt += 1
1160
1161     'This does not work here yet, Pack needs to be changed to a function
1162     'If BufferCnt > BlocksFree Then
1163     'MsgBox("Unable to add bundle to disk :", vbOKOnly, "Not enough free space on disk")
1164     'GoTo NoDisk
1165     'End If
1166
1167     UpdateByteStream()
1168
1169     ResetBuffer()                'Resets buffer variables
1170
1171     NextFileInBuffer = False     'Reset Next File flag
1172
1173     TransitionalBlock = False   'Only the first block of a bundle is a transitional block
1174
1175     FirstLitOfBlock = True
1176
1177     If SI < 0 Then Exit Function 'We have reached the end of the file -> exit
1178
1179     'If we have not reached the end of the file, then update buffer
1180
1181     '-----
-----
1182     '"COLOR BUG"
1183     'Compression bug related to the transitional block (i.e. finding the last block of a bundle) -

```

FIXED

```
1184 'Fix: add 5 or 6 bytes + 2 bits to the calculation to find the last block of a bundle
1185 '+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
1186 '+1 literal bit, +1 match bit (may or may not be needed, but we don't know until the end...)
1187 '-----
-----
1188
1189 'Check if the first literal byte of the NEXT Bundle will go under I/O
1190 'Bits needed for next bundle is calculated in ModDisk:SortPart
1191 '(Next block = Second block) or (remaining bits of Last File in Bundle + Needed Bits fit in
this block)
1192
1193 'LETHARGY BUG - Bits Left need to be calculated from Seq(SI+1) and NOT Seq(SI)
1194 'Add 4 bits if number of nibbles is odd
1195 Dim BitsLeftInBundle As Integer = Seq(SI + 1).TotalBits + ((Seq(SI + 1).Nibbles Mod 2) * 4)
1196
1197 'If the next block is the first one on a new track, no need to recalculate the sequence
1198 'As all previous blocks will be loaded from the previous track before this block gets loaded
1199 Dim NewTrack As Boolean = False
1200 If BufferCnt < (17 * 21) Then
1201
1202     If BufferCnt Mod 21 = 0 Then NewTrack = True
1203
1204 ElseIf BufferCnt < ((17 * 21) + (6 * 19)) Then
1205
1206     If (BufferCnt - (17 * 21)) Mod 19 = 0 Then NewTrack = True
1207
1208 ElseIf BufferCnt < ((17 * 21) + (6 * 19) + (6 * 18)) Then
1209
1210     If (BufferCnt - (17 * 21) - (6 * 19)) Mod 18 = 0 Then NewTrack = True
1211
1212 Else
1213
1214     If (BufferCnt - (17 * 21) - (6 * 19) - (6 * 18)) Mod 17 = 0 Then NewTrack = True
1215
1216 End If
1217
1218 If (BitsLeftInBundle + BitsNeededForNextBundle + If(MLen = 0, 0, 1) + 8 <= ((LastByte - 1) *
8) + BitPos) AndAlso (LastFileOfBundle = True) AndAlso (NewBlock = False) Then
1219 'KARAOKE BUG - fixed in Sparkle 2.2
1220 'BitsLeftInBundle = bits left to be compressed in bundle, add 1 bit if MLen>0, add 8 bits
for Block Count (to simulate 'TransitionalBlock=True')
1221 '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
1222 'I.e. we have identified the last block of the bundle (=transitional block)
1223 LastBlockOfBundle = True
1224 Else
1225 LastBlockOfBundle = False
1226 End If
1227
1228 If (LastBlockOfBundle) OrElse (NewTrack) OrElse (BlockCnt = 1) Then
1229 'Seq(SI+1).Bytes/Nibbles/Bits = to calculate remaining bits in file
1230 'BitsNeededForNextBundle (5-6 bytes + 1/2 bits)
1231 '+5/6 bytes +1/2 bits
1232 'LastByte-1: subtract close tag/block count = Byte(1)
1233 'Bits remaining in block: LastByte * 8 (+ remaining bits in last BitPtr (BitPos+1))
1234 'But we are trying to overcalculate here to avoid misidentification of the last block
1235 'Which would result in buggy decompression
1236
1237 'This is the last block ONLY IF the remainder of the bundle + the next bundle's info
fits!!!
1238 'AND THE NEXT Bundle IS NOT ALIGNED in which case the next block is the last one
```

```

1239 'Seg(SI).bit includes both the byte stream in bits and the bit stream (total bits needed
to compress the remainder of the bundle)
1240 '+Close Tag: 8 bits
1241 '+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)
1242 'For the 2nd and last blocks of a bundle and the first blocks on a new track only
recalculate the first byte's sequence
1243 'If BlockCnt <> 1 Then MsgBox((BitsLeftInBundle + BitsNeededForNextBundle).ToString +
vbNewLine + (Seq(SI + 1).TotalBits + BitsNeededForNextBundle).ToString + vbNewLine + ((LastByte - 1) *
8 + BitPos).ToString)
1244
1245     If NewTrack Then
1246
1247         If Packer = PackerTypes.Better Then
1248
1249             If CurrentFileIndex > PartialFileIndex Then
1250
1251                 If PartialFileIndex > -1 Then
1252                     'Search the finished segment of partial file
1253                     ReferenceFile = Prgs(PartialFileIndex).ToArray
1254                     ReferenceFileStart = Convert.ToInt32(FileAddrA(PartialFileIndex), 16)
1255
1256                     SearchReferenceFile(SI, PartialFileIndex, ReferenceFileStart +
PartialFileOffset, ReferenceFileStart + 1)
1257                 End If
1258
1259                 'Search any finished files on track (partial file < finished file < current
file)
1260                 For I As Integer = PartialFileIndex + 1 To CurrentFileIndex - 1
1261                     ReferenceFile = Prgs(I).ToArray
1262                     ReferenceFileStart = Convert.ToInt32(FileAddrA(I), 16)
1263
1264                     SearchReferenceFile(SI, I, ReferenceFileStart + ReferenceFile.Length - 1,
ReferenceFileStart + 1)
1265                 Next
1266
1267                 'Search the finished segment of this file
1268                 ReferenceFile = Prgs(CurrentFileIndex).ToArray
1269                 ReferenceFileStart = Convert.ToInt32(FileAddrA(CurrentFileIndex), 16)
1270
1271                 SearchReferenceFile(SI, CurrentFileIndex, ReferenceFileStart +
ReferenceFile.Length - 1, ReferenceFileStart + SI + 1)
1272             Else
1273                 'Partial file = CurrentFile (same file, spanning multiple tracks)
1274                 ReferenceFile = Prgs(CurrentFileIndex).ToArray
1275                 ReferenceFileStart = Convert.ToInt32(FileAddrA(CurrentFileIndex), 16)
1276
1277                 SearchReferenceFile(SI, CurrentFileIndex, ReferenceFileStart +
PartialFileOffset, ReferenceFileStart + SI + 1)
1278             End If
1279
1280             'The current file becomes the partial file, anything before it is now finished and
can be used for search
1281             PartialFileIndex = CurrentFileIndex
1282             PartialFileOffset = SI
1283
1284         End If
1285
1286     End If
1287
1288     'Only recalculate the very first byte's sequence
1289     CalcBestSequence(Math.Max(SI, 1), Math.Max(SI, 1)) '(If(SI > 1, SI, 1), If(SI > 1, SI, 1))
1290     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)
1302 CalcAll:
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
1325 Err:
1326     ErrCode = Err.Number
1327     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
1328 NoDisk:
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 DoOnError 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 'BYTES NEEDED: (1)Long Match Tag + (2)End Tag + (3)BitPtr + (4)AdLo + (5)AdHi + (6)1st Literal
1348 +/- (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, TECHNICALY, 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

```

```

1402         End If
1403
1404         'DO NOT CLOSE LAST BUFFER HERE, WE ARE GOING TO ADD NEXT Bundle TO LAST BUFFER
1405         If ByteSt.Count > BlockPtr + 255 Then 'Only save block count if block is already added
to ByteSt
1406             ByteSt(BlockPtr + 1) = EORtransform>LastBlockCnt) 'New Block Count is
ByteSt(BlockPtr+1) in buffer, not ByteSt(BlockPtr+255)
1407             LoaderBundles += 1
1408         End If
1409         'Debug.Print(Hex(BundleNo) + vbTab + LastBlockCnt.ToString)
1410
1411         LitCnt = -1 'Reset LitCnt here
1412     Else
1413 NewB: 'Next File Info does not fit, so close buffer
1414         CloseBuffer() 'Adds EndTag and starts new buffer
1415         'Then add 1 dummy literal byte to new block (blocks must start with 1 literal, next bundle
tag is a match tag)
1416         Buffer(255) = &HFD 'Dummy Address ($03fd* - first literal's address in buffer...
(*NextPart above, will reserve BlockCnt)
1417         Buffer(254) = &H3 '...we are overwriting it with the same value
1418         Buffer(253) = &H0 'Dummy value, will be overwritten with itself
1419         LitCnt = 0
1420         AddLitBits(LitCnt) 'WE NEED THIS HERE, AS THIS IS THE BEGINNING OF THE BUFFER, AND
1ST BIT WILL BE CHANGED TO COMPRESSION BIT
1421         BytePtr = 252
1422         LastBlockCnt += 1
1423
1424         If LastBlockCnt > 255 Then
1425             'Parts cannot be larger than 255 blocks compressed
1426             'There is some confusion here how PartCnt is used in the Editor and during Disk
building...
1427             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!")
1428             If CompressBundleFromEditor = False Then GoTo NoGo
1429         End If
1430
1431         BlockCnt -= 1
1432         'THEN GOTO NEXT Bundle SECTION
1433         GoTo NextPart
1434     End If
1435
1436     NewBlock = SetNewBlock 'NewBlock is true at closing the previous bundle, so first it
just sets NewBlock2
1437     SetNewBlock = False 'And NewBlock2 will fire at the desired bundle
1438
1439     'MsgBox(BundleCnt.ToString + vbNewLine + Hex(BitPtr))
1440     'DirBlocks((BundleCnt * 4) + 3) = BitPtr
1441     'DirPtr(BundleCnt) = BufferCnt
1442     'MsgBox(BundleCnt.ToString + vbNewLine + BufferCnt.ToString)
1443     Exit Function
1444 Err:
1445     ErrCode = Err.Number
1446     MsgBox(ErrorToString(), vbOKOnly + vbExclamation, Reflection.MethodBase.GetCurrentMethod.Name
+ " Error")
1447
1448 NoGo:
1449     CloseBundle = False
1450
1451 End Function
1452
1453 Public Sub CloseFile()
1454     If DoOnError Then On Error GoTo Err

```

```

1455         'ADDS NEXT FILE TAG TO BUFFER
1456
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
1489 Err:
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         'Cycles = 61             'From NextFile label
1512         'BitStreamBytes = 0
1513
1514         'DO NOT RESET LitCnt HERE!!! It is needed for match tag check
1515
1516         Exit Sub
1517
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 DoOnError 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 <= &HFFFF) AndAlso (NextFileUnderIO = 1),
1, 0)
1533     Else
1534         CheckIO = If((Offset >= &HD000) AndAlso (Offset <= &HFFFF) 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 DoOnError 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

```