Commit e10a045

IgorEisberg authored on Dec 12, 2025 · 17 / 17 · Verified

refactor: Improve renaming/injection of resources in stripped APKs (#4041)

```
* refactor: Impove renaming/injection of resources in stripped APKs

* Strict entry spec naming ensures that invalid entry names (incl. attempts of directory
traversal) are replaced.
* Inject generic entries for missing resources to ensure that stripped APKs can be recompiled
without any changes.
  Normally missing attributes or missing ID resources for enum/flag attribute items.
* Default resolve mode changed to KEEP (instead of REMOVE), otherwise stripped APKs can't be
recompiled.
  DUMMY mode can still be used to include unused dummy resources (rarely useful).
  REMOVE mode can still be used to ignore missing resources altogether (rarely useful).
  Output for non-stripped APKs is already identical between all of the modes.
* The rest is just style and micro-optimization.

* unused imports

* hyphen is allowed in resource name

* tweak for performance

* redundant check

* ensure attrs support formats based on usage

Android doesn't verify that a value encoded in an binary XML was encoded with a format that's
supported by the attribute.
The attr's "format" attribute is only enforced by aapt2 during build-time, so some obfuscators
manipulate it in a way that doesn't match with actual usage.
Let the binary XML decoder add missing formats whenever necessary.
No change in output for APKs that weren't obfuscated in this specific way.

* handle edge case: enum/flags that also support integer format

Technically integer format for enum/flags is allowed, but rarely used.
We have to handle this scenario properly. An example from framework-res.apk:

<attr name="numColumns" format="integer" min="0">
    <enum name="auto_fit" value="-1" />
</attr>

We solve this by checking if an attribute value has any matching symbols via
```

```
nameAttr.hasSymbolsForValue(value).

This commit also improves caching for ResEnum/ResFlags symbols to avoid conversion of raw
integer values to symbols more than once.
The caching is done lazily to avoid unnecessary overheads in case certain defined attribute
were never used (i.e. never formatted a value).

This commit changes nothing in any known APK, just handles a theoretical edge case.
```

main (#4041) · v3.0.2 ••• v3.0.0

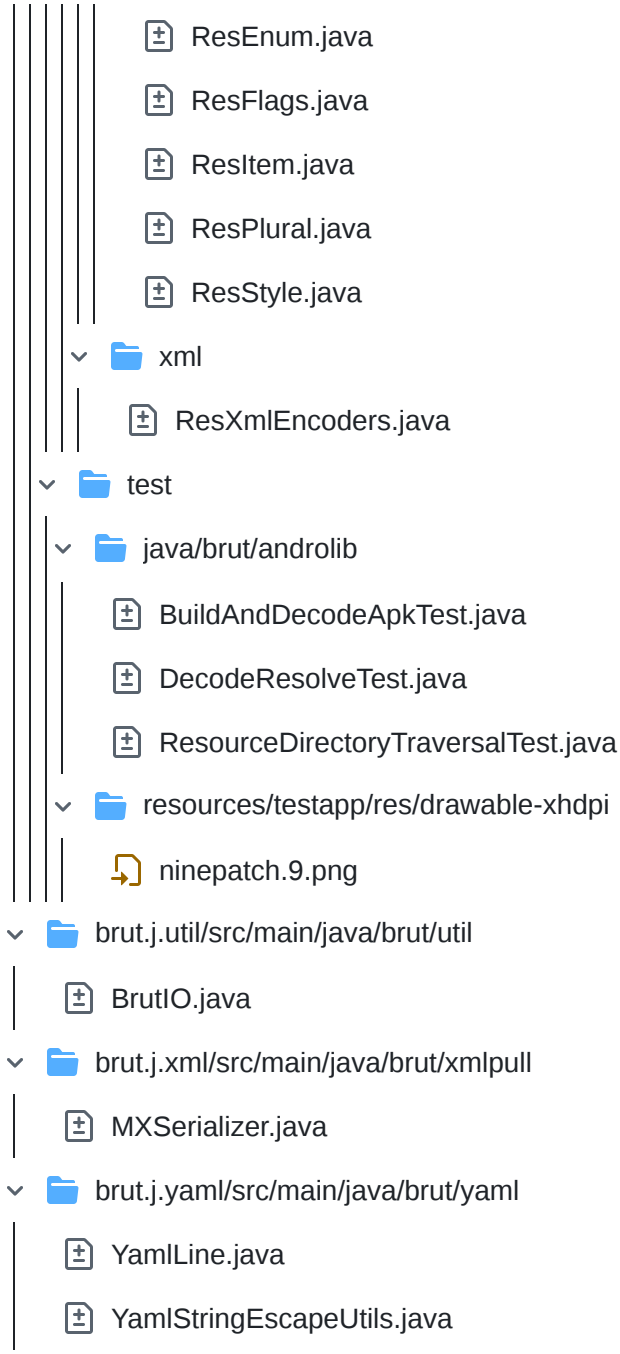
1 parent 0df4f39 commit e10a045

24 files changed +549 -310 lines changed

Top

Filter files…

- brut.apktool
  - apktool-cli/src/main/java/brut/apktool
    - Main.java
  - apktool-lib/src
    - main/java/brut/androlib
      - Config.java
      - res
        - ResourcesDecoder.java
        - decoder
          - BinaryResourceParser.java
          - BinaryXmlResourceParser.java
          - ResFileDecoder.java
        - data
          - StyledString.java
        - table
          - ResEntrySpec.java
        - value
          - ResAttribute.java
          - ResBag.java




```
@@ -460,15 +460,15 @@ private static void cmdDecode(String[] args) throws
AndrolibException {
460  460                        case "remove":
461  461                            config.setDecodeResolve(Config.DecodeResolve.REMOVE);
462  462                            break;
463      -                     case "dummy":
464      -                         config.setDecodeResolve(Config.DecodeResolve.DUMMY);
```

```
465       -                         break;
466  463                       case "keep":
467  464                           config.setDecodeResolve(Config.DecodeResolve.KEEP);
468  465                           break;
     466  +                       case "dummy":
     467  +                           config.setDecodeResolve(Config.DecodeResolve.DUMMY);
     468  +                           break;
469  469                       default:
470  470                           System.err.println("Unknown resolve resources mode: " +
                                mode);
471       -                         System.err.println("Expect: 'remove', 'dummy' or
```

...ol-cli/src/main/java/brut/apktool/Main.java

```
                                'dummy'.");
472  472                           System.exit(1);
473  473                           return;
474  474                   }
```

```
     @@ -19,7 +19,7 @@
19   19    public class Config {
20   20        public enum DecodeSources { NONE, FULL, ONLY_MAIN_CLASSES }
21   21        public enum DecodeResources { NONE, FULL, ONLY_MANIFEST }
22       -    public enum DecodeResolve { REMOVE, DUMMY, KEEP }
     22  +    public enum DecodeResolve { REMOVE, KEEP, DUMMY }
23   23        public enum DecodeAssets { NONE, FULL }
24   24
25   25        private final String mVersion;
     @@ -64,7 +64,7 @@ public Config(String version) {
64   64            mDecodeSources = DecodeSources.FULL;
65   65            mBaksmaliDebugMode = true;
66   66            mDecodeResources = DecodeResources.FULL;
67       -        mDecodeResolve = DecodeResolve.REMOVE;
     67  +        mDecodeResolve = DecodeResolve.KEEP;
68   68            mKeepBrokenResources = false;
69   69            mAnalysisMode = false;
```

...lib/src/main/java/brut/androlib/Config.java

```
@@ -24,6 +24,7 @@
24  24  import brut.androlib.meta.VersionInfo;
25  25  import brut.androlib.res.decoder.*;
26  26  import brut.androlib.res.table.*;
    27 +import brut.androlib.res.table.value.ResBag;
27  28  import brut.androlib.res.table.value.ResFileReference;
28  29  import brut.androlib.res.xml.ResXmlUtils;
29  30  import brut.androlib.res.xml.ValuesXmlSerializable;
@@ -86,15 +87,24 @@ public void decodeResources(File apkDir) throws
    AndrolibException {
86  87          }
87  88
88  89          ResPackage pkg = mTable.getMainPackage();
89     -        Map<ResType, List<ResEntry>> valuesEntries = new HashMap<>();
90  90
91     -        LOGGER.info("Decoding file-resources...");
92     -        for (ResEntry entry : pkg.listEntries()) {
    91 +        LOGGER.info("Decoding value resources...");
    92 +        for (ResEntry entry : new ArrayList<>(pkg.listEntries())) {
    93 +            if (entry.getValue() instanceof ResBag) {
    94 +                ((ResBag) entry.getValue()).resolveKeys();
    95 +            }
    96 +        }
    97 +
    98 +        LOGGER.info("Decoding file resources...");
    99 +        for (ResEntry entry : new ArrayList<>(pkg.listEntries())) {
100 101             if (entry.getValue() instanceof ResFileReference) {
101 102                 fileDecoder.decode(entry, inDir, outDir, mResFileMapping);
102 103             }
103    -            // ResFileDecoder may have replaced an invalid file reference,
104    -            // so we use "if" here rather than "else if".
    104 +        }
    105 +
    106 +        LOGGER.info("Generating values XMLs...");
    107 +        Map<ResType, List<ResEntry>> valuesEntries = new HashMap<>();
    108 +        for (ResEntry entry : pkg.listEntries()) {
105 109             if (entry.getValue() instanceof ValuesXmlSerializable) {
106 110                 ResType type = entry.getType();
```

```
100  110                    List<ResEntry> entries = valuesEntries.get(type);
     @@ -105,8 +115,6 @@ public void decodeResources(File apkDir) throws
     AndrolibException {
105  115                    entries.add(entry);
106  116                }
107  117            }
108       -
109       -          LOGGER.info("Decoding values */* XMLs...");
110  118            for (Map.Entry<ResType, List<ResEntry>> entry :
```

...ava/brut/androlib/res/ResourcesDecoder.java

```
               serial);
112  120            }
```

```
     @@ -812,8 +812,20 @@ private void injectMissingEntrySpecs() throws
     AndrolibException {
812  812            }
813  813
814  814            for (ResId id : mMissingEntrySpecs) {
     815  +             ResTypeSpec typeSpec = mPackage.getTypeSpec(id.getTypeId());
     816  +             ResValue value;
     817  +             switch (typeSpec.getName()) {
     818  +                 case "attr":
     819  +                 case "^attr-private":
     820  +                     value = ResAttribute.DEFAULT;
     821  +                     break;
     822  +                 default:
     823  +                     value = ResReference.NULL;
     824  +                     break;
     825  +             }
     826  +
815  827                mPackage.addEntrySpec(id, ResEntrySpec.DUMMY_PREFIX + id);
816       -            mPackage.addEntry(id, ResConfig.DEFAULT, ResReference.NULL);
     828  +            mPackage.addEntry(id, ResConfig.DEFAULT, value);
```

...rolib/res/decoder/BinaryResourceParser.java

```
819  831    }
```

```
@@ -18,6 +18,7 @@
18 18
19 19  import android.content.res.XmlResourceParser;
20 20  import android.util.TypedValue;
   21 +import brut.androlib.Config;
21 22  import brut.androlib.exceptions.AndrolibException;
22 23  import brut.androlib.exceptions.FrameworkNotFoundException;
23 24  import brut.androlib.exceptions.UndefinedResObjectException;
@@ -61,6 +62,7 @@ public class BinaryXmlResourceParser implements
XmlResourceParser {
61 62      private static final int ATTRIBUTE_IX_VALUE_DATA = 4; // data
62 63      private static final int ATTRIBUTE_LENGTH = 5;
63 64
   65 +    private static final int ANDROID_PKG_ID = 0x01;
64 66      private static final int PRIVATE_PKG_ID = 0x7F;
65 67
66 68      private final ResTable mTable;
@@ -215,10 +217,11 @@ public char[] getTextCharacters(int[]
holderForStartAndLength) {
215 217          if (text == null) {
216 218              return null;
217 219          }
   220 +        int len = text.length();
218 221          holderForStartAndLength[0] = 0;
219     -        holderForStartAndLength[1] = text.length();
220     -        char[] chars = new char[text.length()];
221     -        text.getChars(0, text.length(), chars, 0);
   222 +        holderForStartAndLength[1] = len;
   223 +        char[] chars = new char[len];
   224 +        text.getChars(0, len, chars, 0);
222 225          return chars;
223 226      }
224 227
@@ -310,20 +313,24 @@ public String getAttributeNamespace(int index) {
310 313          int offset = getAttributeOffset(index);
311 314          int namespace = mAttributes[offset + ATTRIBUTE_IX_NAMESPACE_URI];
312 315
```

```
                316 +           ResId nameId = ResId.of(getAttributeNameResource(index));
                317 +           int pkgId = nameId.getPackageId();
                318 +
313             319             // #2972 - If the namespace index is -1, the attribute is not present, but if the attribute is from system
314             320             // we can resolve it to the default namespace. This may prove to be too aggressive as we scope the entire
315             321             // system namespace, but it is better than not resolving it at all.
316             -           ResId id = ResId.of(getAttributeNameResource(index));
317             -           int pkgId = id.getPackageId();
318             -           if (namespace == -1 && pkgId == 1) {
319             -               return ANDROID_RES_NS;
320             -           }
321             322             if (namespace == -1) {
                323 +               if (pkgId == PRIVATE_PKG_ID) {
                324 +                   return getNonDefaultNamespaceUri(offset);
                325 +               }
                326 +               if (pkgId == ANDROID_PKG_ID) {
                327 +                   return ANDROID_RES_NS;
                328 +               }
322             329                 return "";
323             330             }
324             331
325             332             // Minifiers like removing the namespace, so we will default to default namespace
326             -           // unless the pkgId of the resource is private. We will grab the non-standard one.
                333 +           // unless the package ID of the resource is private. We will grab the non-standard one.
327             334             String value = mStringPool.getString(namespace);
328             335             if (value != null && !value.isEmpty()) {
329             336                 return value;
                    @@ -375,23 +382,50 @@ public String getAttributeName(int index) {
375             382             // Are improperly decoded when trusting the string pool.
376             383             // Leveraging the resource map allows us to get the proper value.
377             384             // <item android:state_enabled="true" app:d2="false" app:d3="true">
378             -           String resourceMapValue;
379             385             try {
380             -               resourceMapValue = decodeFromResourceId(nameId);
```

```
      386 +              String resourceMapValue = decodeFromResourceId(nameId);
      387 +              if (resourceMapValue != null) {
      388 +                  return resourceMapValue;
      389 +              }
386   390               } catch (AndrolibException ignored) {
382         -             resourceMapValue = null;
383         -         }
384         -         if (resourceMapValue != null) {
385         -             return resourceMapValue;
386   391               }
387   392
      393 +         // Couldn't decode from resource map, fall back to string pool.
388   394           String stringPoolValue = mStringPool.getString(name);
389         -         if (stringPoolValue != null) {
390         -             return stringPoolValue;
      395 +         if (stringPoolValue == null) {
      396 +             stringPoolValue = "";
391   397           }
392   398
393         -         // If it was not found in either, then we have a bogus resource.
394         -         return ResEntrySpec.MISSING_PREFIX + nameId;
      399 +         // In certain optimized apps, some attributes's specs are removed
          despite being used.
      400 +         // Inject a generic spec for the attribute, otherwise we can't rebuild.
      401 +         if (nameId != ResId.NULL) {
      402 +             Config config = mTable.getConfig();
      403 +             boolean removeUnresolved = config.getDecodeResolve() ==
          Config.DecodeResolve.REMOVE;
      404 +             try {
      405 +                 ResPackage pkg = mTable.getMainPackage();
      406 +
      407 +                 // #2836 - Skip item if the resource cannot be identified.
      408 +                 if (removeUnresolved || nameId.getPackageId() != pkg.getId()) {
      409 +                     LOGGER.warning(String.format(
      410 +                         "null attr reference: ns=%s, name=%s, id=%s",
      411 +                         getAttributePrefix(index), stringPoolValue, nameId));
      412 +                     return stringPoolValue;
      413 +                 }
      414 +
      415 +                 if (stringPoolValue.isEmpty()) {
```

```
                    stringPoolValue = ResEntrySpec.MISSING_PREFIX + nameId;
                }
                pkg.addEntrySpec(nameId, stringPoolValue);
                pkg.addEntry(nameId, ResConfig.DEFAULT, ResAttribute.DEFAULT);
            } catch (AndrolibException ex) {
                setFirstError(ex);
                LOGGER.warning(String.format(
                    "Could not add missing attr: ns=%s, name=%s, id=%s",
                    getAttributePrefix(index), stringPoolValue, nameId));
            }
        }

        return stringPoolValue;
    }

    private String decodeFromResourceId(ResId resId) throws AndrolibException {
```

```
@@ -425,29 +459,48 @@ public String getAttributeValue(int index) {
            try {
                String stringPoolValue = valueRaw != -1
                    ?
ResXmlEncoders.escapeXmlChars(mStringPool.getString(valueRaw)) : null;
-               String resourceMapValue = null;
+               String rawValue = stringPoolValue;

-               // Ensure we only track down obfuscated values for
reference/attribute type values. Otherwise, we might
-               // spam lookups against resource table for invalid IDs.
+               boolean isExplicitType = valueType != TypedValue.TYPE_NULL;
                if (valueType == TypedValue.TYPE_REFERENCE
                        || valueType == TypedValue.TYPE_DYNAMIC_REFERENCE
                        || valueType == TypedValue.TYPE_ATTRIBUTE
                        || valueType == TypedValue.TYPE_DYNAMIC_ATTRIBUTE) {
-                   resourceMapValue = decodeFromResourceId(ResId.of(valueData));
+                   // Explicit reference format is optional.
+                   isExplicitType = false;
+                   // Android prefers the resource map value over what the string
pool has.
+                   // We only track down obfuscated values for reference/attribute
type values.
```

```
                473 +                // Otherwise, we might spam lookups against resource table for invalid IDs.
                474 +                String resourceMapValue = decodeFromResourceId(ResId.of(valueData));
                475 +                if (stringPoolValue != null && resourceMapValue != null) {
                476 +                    // Handle a value with a format of "@yyy/xxx", but avoid "@yyy/zzz:xxx"
                477 +                    int slashPos = stringPoolValue.lastIndexOf('/');
                478 +                    if (slashPos != -1) {
                479 +                        int colonPos = stringPoolValue.lastIndexOf(':');
                480 +                        if (colonPos == -1) {
                481 +                            rawValue = stringPoolValue.substring(0, slashPos) + "/" + resourceMapValue;
                482 +                        }
                483 +                    } else if (!stringPoolValue.equals(resourceMapValue)) {
                484 +                        rawValue = resourceMapValue;
                485 +                    }
                486 +                }
437 487             }
438 488
439 489             // Try to decode from resource table.
440 490             ResId nameId = ResId.of(getAttributeNameResource(index));
441     -             ResItem value = ResItem.parse(mTable.getCurrentPackage(), valueType, valueData,
442     -                 getPreferredString(stringPoolValue, resourceMapValue));
                491 +             ResItem value = ResItem.parse(mTable.getCurrentPackage(), valueType, valueData, rawValue);
443 492             if (nameId != ResId.NULL) {
444 493                 try {
445 494                     // We need the attribute entry's value to format this value.
446 495                     ResEntrySpec nameSpec = mTable.getEntrySpec(nameId);
447     -                     ResValue nameDefValue = mTable.getDefaultEntry(nameId).getValue();
448     -
449     -                     if (nameDefValue instanceof ResAttribute) {
450     -                         decoded = ((ResAttribute) nameDefValue).formatValue(value, false);
                496 +                     ResValue nameValue = mTable.getDefaultEntry(nameId).getValue();
```

24 files changed +549 -310 lines changed ↑ Top Search within code

```
499 +                        ResAttribute nameAttr = (ResAttribute) nameValue;
500 +                        if (isExplicitType &&
    !nameAttr.hasSymbolsForValue(value)) {
501 +                            nameAttr.addType(valueType);
502 +                        }
503 +                        decoded = nameAttr.formatValue(value, false);
451 504                  } else {
452 505                      LOGGER.warning("Unexpected attribute name: " +
    nameSpec);
453 506                  }
```

...ib/res/decoder/BinaryXmlResourceParser.java

```
678 731             return -1;
679 732         }
680 733
681     -     private static String getPreferredString(String stringPoolValue, String
    resourceMapValue) {
682     -         String value = stringPoolValue;
683     -
684     -         if (stringPoolValue != null && resourceMapValue != null) {
685     -             int slashPos = stringPoolValue.lastIndexOf('/');
686     -             int colonPos = stringPoolValue.lastIndexOf(':');
687     -
688     -             // Handle a value with a format of "@yyy/xxx", but avoid
    "@yyy/zzz:xxx"
689     -             if (slashPos != -1) {
690     -                 if (colonPos == -1) {
691     -                     String type = stringPoolValue.substring(0, slashPos);
692     -                     value = type + "/" + resourceMapValue;
693     -                 }
694     -             } else if (!stringPoolValue.equals(resourceMapValue)) {
695     -                 value = resourceMapValue;
696     -             }
697     -         }
698     -         return value;
699     -     }
700     -
701 734         private void resetEventInfo() {
```

```
702  735            mEvent = -1;
703  736            mLineNumber = -1;
```

...ut/androlib/res/decoder/ResFileDecoder.java

```
@@ -100,13 +100,8 @@ public void decode(ResEntry entry, Directory inDir,
Directory outDir, Map<String
100  100            }
101  101
102  102            // Generate output file path from entry.
103       -          String outResPath = entry.getTypeName() +
entry.getConfig().getQualifiers() + "/" + entry.getName();
104       -          if (BrutIO.detectPossibleDirectoryTraversal(outResPath)) {
105       -              LOGGER.warning("Potentially malicious file path: " + outResPath +
", using instead: " + inResPath);
106       -              outResPath = inResPath;
107       -          } else if (!ext.isEmpty()) {
108       -              outResPath += "." + ext;
109       -          }
     103  +          String outResPath = entry.getTypeName() +
entry.getConfig().getQualifiers() + "/" + entry.getName()
     104  +                  + (ext.isEmpty() ? "" : "." + ext);
110  105
111  106            // Map output path to original path if it's different.
112  107            String outFileName = "res/" + outResPath;
```

...androlib/res/decoder/data/StyledString.java

```
@@ -65,7 +65,8 @@ private String decode() {
65   65                 return mDecodedText;
66   66             }
67   67
68        -          mBuffer = new StringBuilder(mText.length() * 2);
     68   +          int len = mText.length();
     69   +          mBuffer = new StringBuilder(len * 2);
69   70             mLastOffset = 0;
70   71
71   72             // Recurse top-level tags.
@@ -75,7 +76,7 @@ private String decode() {
```

```
75  76            }
76  77
77  78            // Write the remaining encoded raw text.
78      -         if (mLastOffset < mText.length()) {
    79  +         if (mLastOffset < len) {
79  80
                  mBuffer.append(ResXmlEncoders.escapeXmlChars(mText.substring(mLastOffset)));
80  81            }
81  82
                  @@ -117,10 +118,11 @@ private void decodeIterate(PeekingIterator<Span> it) {
117 118           }
118 119
119 120           // Write encoded raw text preceding the closing tag.
120     -         if (spanEnd > mLastOffset && mText.length() >= spanEnd) {
    121 +         int len = mText.length();
    122 +         if (spanEnd > mLastOffset && len >= spanEnd) {
121 123
                  mBuffer.append(ResXmlEncoders.escapeXmlChars(mText.substring(mLastOffset,
                  spanEnd)));
122     -         } else if (mText.length() >= mLastOffset && mText.length() < spanEnd) {
123     -             LOGGER.warning("Span (" + name + ") exceeds text length " +
                  mText.length());
    124 +         } else if (len >= mLastOffset && len < spanEnd) {
    125 +             LOGGER.warning("Span (" + name + ") exceeds text length " + len);
124 126
                  mBuffer.append(ResXmlEncoders.escapeXmlChars(mText.substring(mLastOffset)));
125 127           }
126 128           mLastOffset = spanEnd;
```

...a/brut/androlib/res/table/ResEntrySpec.java

```
                  @@ -16,17 +16,9 @@
16  16             */
17  17            package brut.androlib.res.table;
18  18
19      - import com.google.common.collect.Sets;
20      -
21  19            import java.util.Objects;
22      - import java.util.Set;
```

```
23  20
24  21    public class ResEntrySpec {
25      -     private static final Set<String> INVALID_ENTRY_NAMES = Sets.newHashSet(
26      -         "0_resource_name_obfuscated", // #3067
27      -         "(name removed)" // #2940
28      -     );
29      -
30  22        public static final String DUMMY_PREFIX = "APKTOOL_DUMMY_";
31  23        public static final String MISSING_PREFIX = "APKTOOL_MISSING_";
32  24        public static final String RENAMED_PREFIX = "APKTOOL_RENAMED_";
            @@ -40,13 +32,33 @@ public ResEntrySpec(ResTypeSpec typeSpec, ResId id,
            String name) {
40  32            assert typeSpec.getId() == id.getTypeId();
41  33            mTypeSpec = typeSpec;
42  34            mId = id;
43      -         // Some apps had their entry names collapsed to a single value in
44      -         // the key string pool. Rename to avoid duplicates.
45      -         if (name == null || name.isEmpty() ||
            INVALID_ENTRY_NAMES.contains(name)) {
46      -             mName = RENAMED_PREFIX + id;
47      -         } else {
48      -             mName = name;
    35  +         // Some apps had their entry names obfuscated or collapsed to
    36  +         // a single value in the key string pool.
    37  +         mName = isValidEntryName(name) ? name : RENAMED_PREFIX + id;
    38  +     }
    39  +
    40  +     private static boolean isValidEntryName(String name) {
    41  +         // Must not be empty.
    42  +         int len = name.length();
    43  +         if (len == 0) {
    44  +             return false;
    45  +         }
    46  +         // Must start with a valid Java identifier start character.
    47  +         if (!Character.isJavaIdentifierStart(name.charAt(0))) {
    48  +             return false;
    49  +         }
    50  +         // The rest must be valid Java identifier part characters.
    51  +         for (int i = 1; i < len; i++) {
    52  +             char ch = name.charAt(i);
```

```
      53 +             // Whitelisted special characters.
      54 +             if (ch == '.' || ch == '-') {
      55 +                 continue;
      56 +             }
      57 +             if (!Character.isJavaIdentifierPart(ch)) {
      58 +                 return false;
      59 +             }
49    60           }
      61 +         return true;
50    62       }
51    63
52    64       public ResTypeSpec getTypeSpec() {
```

.../androlib/res/table/value/ResAttribute.java

```
@@ -16,6 +16,7 @@
16    16    */
17    17   package brut.androlib.res.table.value;
18    18
      19 + import android.util.TypedValue;
19    20   import brut.androlib.exceptions.AndrolibException;
20    21   import brut.androlib.res.table.ResEntry;
21    22   import brut.androlib.res.table.ResId;
@@ -62,7 +63,7 @@ public class ResAttribute extends ResBag implements
ValuesXmlSerializable {
62    63       public static final ResAttribute DEFAULT = new ResAttribute(
63    64           null, TYPE_ANY, Integer.MIN_VALUE, Integer.MAX_VALUE,
L10N_NOT_REQUIRED);
64    65
65        - protected final int mType;
      66 + protected int mType;
66    67       protected final int mMin;
67    68       protected final int mMax;
68    69       protected final int mL10n;
@@ -149,6 +150,51 @@ public ResPrimitive getValue() {
149   150           }
150   151       }
151   152
      153 + public void addType(int type) {
```

```
154 +         if ((mType & TYPE_ANY) == TYPE_ANY) {
155 +             return;
156 +         }
157 +         switch (type) {
158 +             case TypedValue.TYPE_REFERENCE:
159 +             case TypedValue.TYPE_DYNAMIC_REFERENCE:
160 +             case TypedValue.TYPE_ATTRIBUTE:
161 +             case TypedValue.TYPE_DYNAMIC_ATTRIBUTE:
162 +                 mType |= TYPE_REFERENCE;
163 +                 return;
164 +             case TypedValue.TYPE_STRING:
165 +                 mType |= TYPE_STRING;
166 +                 return;
167 +             case TypedValue.TYPE_FLOAT:
168 +                 mType |= TYPE_FLOAT;
169 +                 return;
170 +             case TypedValue.TYPE_DIMENSION:
171 +                 mType |= TYPE_DIMEN;
172 +                 return;
173 +             case TypedValue.TYPE_FRACTION:
174 +                 mType |= TYPE_FRACTION;
175 +                 return;
176 +             case TypedValue.TYPE_INT_BOOLEAN:
177 +                 mType |= TYPE_BOOL;
178 +                 return;
179 +             default:
180 +                 if (type >= TypedValue.TYPE_FIRST_COLOR_INT && type <=
    TypedValue.TYPE_LAST_COLOR_INT) {
181 +                     mType |= TYPE_COLOR;
182 +                 } else if (type >= TypedValue.TYPE_FIRST_INT && type <=
    TypedValue.TYPE_LAST_INT) {
183 +                     mType |= TYPE_INT;
184 +                 }
185 +                 return;
186 +         }
187 +     }
188 +
189 +     public boolean hasSymbolsForValue(ResItem value) throws AndrolibException {
190 +         return getSymbolsForValue(value) != null;
191 +     }
```

```
      192 +
      193 +     protected Symbol[] getSymbolsForValue(ResItem value) throws
          AndrolibException {
      194 +         // Stub for attribute types with symbols.
      195 +         return null;
      196 +     }
      197 +
152   198         public String formatValue(ResItem value, boolean asTextNode) throws
          AndrolibException {
153   199             if (!(value instanceof ResXmlEncodable)) {
154   200                 return null;
```

...a/brut/androlib/res/table/value/ResBag.java

```
      @@ -16,6 +16,8 @@
16    16       */
17    17    package brut.androlib.res.table.value;
18    18
      19 + import brut.androlib.exceptions.AndrolibException;
      20 +
19    21    import java.util.logging.Logger;
20    22
21    23    public abstract class ResBag extends ResValue {
      @@ -61,4 +63,8 @@ public ResItem getValue() {
61    63                 return mValue;
62    64             }
63    65         }
      66 +
      67 +     public void resolveKeys() throws AndrolibException {
      68 +         // Stub for bags with resolvable keys.
      69 +     }
64    70    }
```

.../brut/androlib/res/table/value/ResEnum.java

```
      @@ -16,13 +16,18 @@
16    16       */
17    17    package brut.androlib.res.table.value;
18    18
```

```
      19 + import android.util.TypedValue;
19    20   import brut.androlib.Config;
20    21   import brut.androlib.exceptions.AndrolibException;
      22 + import brut.androlib.res.table.ResConfig;
21    23   import brut.androlib.res.table.ResEntry;
22    24   import brut.androlib.res.table.ResEntrySpec;
      25 + import brut.androlib.res.table.ResId;
      26 + import brut.androlib.res.table.ResPackage;
23    27   import org.xmlpull.v1.XmlSerializer;
24    28
25    29   import java.io.IOException;
      30 + import java.util.Arrays;
26    31   import java.util.HashMap;
27    32   import java.util.Map;
28    33   import java.util.Objects;
           @@ -32,77 +37,137 @@ public class ResEnum extends ResAttribute {
32    37       private static final Logger LOGGER =
     Logger.getLogger(ResEnum.class.getName());
33    38
34    39       private final Symbol[] mSymbols;
35       -    private final Map<Integer, String> mFormatCache;
      40 +    private Map<Integer, Symbol[]> mSymbolsCache;
      41 +    private Map<Integer, String> mFormatsCache;
36    42
37    43       public ResEnum(ResReference parent, int type, int min, int max, int l10n,
     Symbol[] symbols) {
38    44           super(parent, type, min, max, l10n);
39    45           mSymbols = symbols;
40       -        mFormatCache = new HashMap<>();
41    46       }
42    47
43    48       @Override
44       -    protected String formatValueToSymbols(ResItem value) throws
     AndrolibException {
45       -        if (!(value instanceof ResPrimitive)) {
      49 +    public void resolveKeys() throws AndrolibException {
      50 +        ResPackage pkg = mParent.getPackage();
      51 +        Config config = pkg.getTable().getConfig();
      52 +        boolean removeUnresolved = config.getDecodeResolve() ==
     Config.DecodeResolve.REMOVE;
```

```
                                                                    +
                                                                    +         for (Symbol symbol : mSymbols) {
                                                                    +             ResReference key = symbol.getKey();
                                                                    +             if (key.resolve() != null) {
                                                                    +                 continue;
                                                                    +             }
                                                                    +
                                                                    +             ResId entryId = key.getId();
                                                                    +
                                                                    +             // #2836 - Skip item if the resource cannot be identified.
                                                                    +             if (removeUnresolved || entryId.getPackageId() != pkg.getId()) {
                                                                    +                 LOGGER.warning(String.format(
                                                                    +                     "null enum reference: key=%s, value=%s", key,
                                                                      symbol.getValue()));
                                                                    +                 continue;
                                                                    +             }
                                                                    +
                                                                    +             pkg.addEntrySpec(entryId, ResEntrySpec.MISSING_PREFIX + entryId);
                                                                    +             pkg.addEntry(entryId, ResConfig.DEFAULT, new ResCustom("id"));
                                                                    +         }
                                                                    +     }
                                                                    +
                                                                    +     @Override
                                                                    +     protected Symbol[] getSymbolsForValue(ResItem value) throws
                                                                      AndrolibException {
                                                                    +         if (!isSymbolValueType(value)) {
46                                                                  77                return null;
47                                                                  78            }
48                                                                  79
49                                                                  80            int data = ((ResPrimitive) value).getData();
50                                                                  -         String formatted = mFormatCache.get(data);
51                                                                  -         if (formatted != null) {
52                                                                  -             return formatted;
                                                                    81        return getSymbols(data);
                                                                    82    }
                                                                    83
                                                                    84    private boolean isSymbolValueType(ResItem value) throws AndrolibException {
                                                                    85        if (!(value instanceof ResPrimitive)) {
                                                                    86            return false;
53                                                                  87            }
```

```
54  88
        89  +         int type = ((ResPrimitive) value).getType();
        90  +         return type == TypedValue.TYPE_INT_DEC || type ==
            TypedValue.TYPE_INT_HEX;
        91  +     }
        92  +
        93  +     private Symbol[] getSymbols(int data) throws AndrolibException {
        94  +         if (mSymbolsCache == null) {
        95  +             // Lazily establish a symbols cache for performance.
        96  +             mSymbolsCache = new HashMap<>();
        97  +         } else if (mSymbolsCache.containsKey(data)) {
        98  +             return mSymbolsCache.get(data);
        99  +         }
       100  +
       101  +         Symbol[] symbols = new Symbol[mSymbols.length];
       102  +         int symbolsCount = 0;
       103  +
55  104             for (Symbol symbol : mSymbols) {
56      -             if (symbol.getValue().getData() != data) {
57      -                 continue;
       105  +             if (symbol.getValue().getData() == data) {
       106  +                 symbols[symbolsCount++] = symbol;
58  107             }
       108  +         }
       109  +
       110  +         if (symbolsCount == 0) {
       111  +             symbols = null;
       112  +         } else if (symbolsCount < symbols.length) {
       113  +             symbols = Arrays.copyOf(symbols, symbolsCount);
       114  +         }
       115  +
       116  +         mSymbolsCache.put(data, symbols);
       117  +         return symbols;
       118  +     }
       119  +
       120  +     @Override
       121  +     protected String formatValueToSymbols(ResItem value) throws
            AndrolibException {
       122  +         if (!isSymbolValueType(value)) {
       123  +             return null;
```

```
        124 +          }
        125 +
        126 +          int data = ((ResPrimitive) value).getData();
        127 +          if (mFormatsCache == null) {
        128 +              // Lazily establish a formats cache for performance.
        129 +              mFormatsCache = new HashMap<>();
        130 +          } else if (mFormatsCache.containsKey(data)) {
        131 +              return mFormatsCache.get(data);
        132 +          }
        133 +
        134 +          Symbol[] symbols = getSymbols(data);
        135 +          String formatted = null;
        136 +
        137 +          if (symbols != null) {
        138 +              for (Symbol symbol : symbols) {
        139 +                  ResEntrySpec keySpec = symbol.getKey().resolve();
        140 +                  if (keySpec == null) {
        141 +                      continue;
        142 +                  }
 59     143
 60         -              ResReference key = symbol.getKey();
 61         -              ResEntrySpec keySpec = key.resolve();
 62         -              if (keySpec != null) {
 63     144                    formatted = keySpec.getName();
 64         -                  mFormatCache.put(data, formatted);
 65     145
 66     146                    // fill_parent is deprecated since API 8 but appears first.
 67     147                    // Keep looking for match_parent and use it instead if found.
 68     148                    if (data == -1 && formatted.equals("fill_parent")) {
 69     149                        continue;
 70     150                    }
        151 +                  break;
 71     152                }
 72         -              break;
 73     153            }
 74     154
        155 +          mFormatsCache.put(data, formatted);
 75     156            return formatted;
 76     157        }
 77     158
```

```
78  159         @Override
79  160         protected void serializeSymbolsToValuesXml(XmlSerializer serial, ResEntry
                entry)
80  161                 throws AndrolibException, IOException {
81      -         Config config = mParent.getPackage().getTable().getConfig();
82      -         boolean removeUnresolved = config.getDecodeResolve() ==
                  Config.DecodeResolve.REMOVE;
83      -
84  162             for (Symbol symbol : mSymbols) {
85      -               ResReference key = symbol.getKey();
86      -               ResEntrySpec keySpec = key.resolve();
87      -               ResPrimitive value = symbol.getValue();
88      -
89      -               String name;
90      -               if (keySpec != null) {
91      -                   name = keySpec.getName();
92      -               } else {
93      -                   // #2836 - Support skipping items if the resource cannot be
                        identified.
94      -                   if (removeUnresolved) {
95      -                       LOGGER.warning(String.format(
96      -                           "null enum reference: key=%s, value=%s", key, value));
97      -                       continue;
98      -                   }
99      -
100     -                   name = ResEntrySpec.MISSING_PREFIX + key.getId();
        163 +               ResEntrySpec keySpec = symbol.getKey().resolve();
        164 +               if (keySpec == null) {
        165 +                   continue;
101 166               }
102 167
103 168               serial.startTag(null, "enum");
104     -               serial.attribute(null, "name", name);
105     -               serial.attribute(null, "value", value.encodeAsResXmlAttrValue());
        169 +               serial.attribute(null, "name", keySpec.getName());
        170 +               serial.attribute(null, "value",
                        symbol.getValue().encodeAsResXmlAttrValue());
106 171               serial.endTag(null, "enum");
107 172           }
108 173       }
```

...brut/androlib/res/table/value/ResFlags.java

```
@@ -16,10 +16,14 @@
 */
package brut.androlib.res.table.value;

+ import android.util.TypedValue;
import brut.androlib.Config;
import brut.androlib.exceptions.AndrolibException;
+ import brut.androlib.res.table.ResConfig;
import brut.androlib.res.table.ResEntry;
import brut.androlib.res.table.ResEntrySpec;
+ import brut.androlib.res.table.ResId;
+ import brut.androlib.res.table.ResPackage;
import org.xmlpull.v1.XmlSerializer;

import java.io.IOException;
@@ -34,166 +38,197 @@ public class ResFlags extends ResAttribute {
    private static final Logger LOGGER =
Logger.getLogger(ResFlags.class.getName());

    private final Symbol[] mSymbols;
-   private final Symbol[] mZeroFlags;
-   private final Symbol[] mFlags;
-   private final Map<Integer, String> mFormatCache;
+   private Map<Integer, Symbol[]> mSymbolsCache;
+   private Map<Integer, String> mFormatsCache;
+   private Symbol[] mSortedSymbols;

    public ResFlags(ResReference parent, int type, int min, int max, int l10n,
Symbol[] symbols) {
        super(parent, type, min, max, l10n);
        mSymbols = symbols;
-       mFormatCache = new HashMap<>();
+   }
+
+   @Override
+   public void resolveKeys() throws AndrolibException {
+       ResPackage pkg = mParent.getPackage();
```

```
        53 +          Config config = pkg.getTable().getConfig();
        54 +          boolean removeUnresolved = config.getDecodeResolve() ==
             Config.DecodeResolve.REMOVE;
45      55
46         -          Symbol[] zeroFlags = new Symbol[symbols.length];
47         -          int zeroFlagsCount = 0;
48         -          Symbol[] flags = new Symbol[symbols.length];
49         -          int flagsCount = 0;
        56 +          for (Symbol symbol : mSymbols) {
        57 +              ResReference key = symbol.getKey();
        58 +              if (key.resolve() != null) {
        59 +                  continue;
        60 +              }
50      61
51         -          for (Symbol symbol : symbols) {
52         -              ResPrimitive value = symbol.getValue();
        62 +              ResId entryId = key.getId();
53      63
54         -              if (value.getData() == 0) {
55         -                  zeroFlags[zeroFlagsCount++] = symbol;
56         -              } else {
57         -                  flags[flagsCount++] = symbol;
        64 +              // #2836 - Skip item if the resource cannot be identified.
        65 +              if (removeUnresolved || entryId.getPackageId() != pkg.getId()) {
        66 +                  LOGGER.warning(String.format(
        67 +                      "null flag reference: key=%s, value=%s", key,
             symbol.getValue()));
        68 +                  continue;
58      69                }
59         -          }
60      70
61         -          mZeroFlags = zeroFlagsCount < zeroFlags.length
62         -              ? Arrays.copyOf(zeroFlags, zeroFlagsCount) : zeroFlags;
63         -          mFlags = flagsCount < flags.length ? Arrays.copyOf(flags, flagsCount) :
             flags;
64         -
65         -          // We establish a priority list for the flags. This can never be
             completely
66         -          // accurate to the source, but it's a best-guess approach.
67         -          Comparator<Symbol> byBitCount = Comparator.comparingInt(
```

```
68        -             (Symbol symbol) -> Integer.bitCount(symbol.getValue().getData()));
69        -         Comparator<Symbol> byRawValue = Comparator.comparingInt(
70        -             (Symbol symbol) -> symbol.getValue().getData());
71        -         Arrays.sort(mFlags, byBitCount.reversed().thenComparing(byRawValue));
      71  +             pkg.addEntrySpec(entryId, ResEntrySpec.MISSING_PREFIX + entryId);
      72  +             pkg.addEntry(entryId, ResConfig.DEFAULT, new ResCustom("id"));
      73  +         }
72    74          }
73    75
74    76          @Override
75        -     protected String formatValueToSymbols(ResItem value) throws
          AndrolibException {
76        -         if (!(value instanceof ResPrimitive)) {
      77  +     protected Symbol[] getSymbolsForValue(ResItem value) throws
          AndrolibException {
      78  +         if (!isSymbolValueType(value)) {
77    79                ...
78    80            }
79    81
80    82            int data = ((ResPrimitive) value).getData();
81        -         String formatted = mFormatCache.get(data);
82        -         if (formatted != null) {
83        -             return formatted;
      83  +         return getSymbols(data);
      84  +     }
      85  +
      86  +     private boolean isSymbolValueType(ResItem value) throws AndrolibException {
      87  +         if (!(value instanceof ResPrimitive)) {
      88  +             return false;
84    89            }
85    90
86        -         Symbol[] symbols;
87        -         int count = 0;
88        -         if (data == 0) {
89        -             symbols = mZeroFlags;
90        -             count = symbols.length;
      91  +         int type = ((ResPrimitive) value).getType();
      92  +         return type == TypedValue.TYPE_INT_DEC || type ==
          TypedValue.TYPE_INT_HEX;
      93  +     }
```

```
91  94
92      -             if (count == 0) {
93      -                 return null;
        95  +     private Symbol[] getSymbols(int data) throws AndrolibException {
        96  +         if (mSymbolsCache == null) {
        97  +             // Lazily establish a symbols cache for performance.
        98  +             mSymbolsCache = new HashMap<>();
        99  +         } else if (mSymbolsCache.containsKey(data)) {
       100  +             return mSymbolsCache.get(data);
       101  +         }
       102  +
       103  +         if (mSortedSymbols == null) {
       104  +             // Lazily establish a priority list for the flags. This can never
                        be
       105  +             // completely accurate to the source, but it's a best-effort
                        approach.
       106  +             mSortedSymbols = mSymbols.clone();
       107  +             Comparator<Symbol> byBitCount = Comparator.comparingInt(
       108  +                 (Symbol symbol) ->
                        Integer.bitCount(symbol.getValue().getData()));
       109  +             Comparator<Symbol> byRawValue = Comparator.comparingInt(
       110  +                 (Symbol symbol) -> symbol.getValue().getData());
       111  +             Arrays.sort(mSortedSymbols,
                        byBitCount.reversed().thenComparing(byRawValue));
       112  +         }
       113  +
       114  +         Symbol[] symbols = new Symbol[mSortedSymbols.length];
       115  +         int symbolsCount = 0;
       116  +
       117  +         if (data == 0) {
       118  +             for (Symbol symbol : mSortedSymbols) {
       119  +                 if (symbol.getValue().getData() == 0) {
       120  +                     symbols[symbolsCount++] = symbol;
       121  +                 }
94  122              }
95  123          } else {
96      -             symbols = new Symbol[mFlags.length];
97  124              int mask = 0;
98  125
99      -             for (Symbol symbol : mFlags) {
```

```
                126 +             for (Symbol symbol : mSortedSymbols) {
100             127                   int flag = symbol.getValue().getData();
101                 -
102             128                   if ((data & flag) != flag || (mask & flag) == flag) {
103             129                       continue;
104             130                   }
105             131
106                 -                 symbols[count++] = symbol;
                132 +                 symbols[symbolsCount++] = symbol;
107             133                   mask |= flag;
108             134
109             135                   if (mask == data) {
110             136                       break;
111             137                   }
112             138               }
113             139
114                 -             if (count == 0) {
115                 -                 LOGGER.warning("Invalid flags value: " + value);
116                 -                 return null;
117                 -             }
118                 -         }
                140 +             // Filter out redundant flags.
                141 +             if (symbolsCount > 2) {
                142 +                 Symbol[] filtered = new Symbol[symbolsCount];
                143 +                 int filteredCount = 0;
119             144
120                 -         // Render the flags as a format.
121                 -         Config config = mParent.getPackage().getTable().getConfig();
122                 -         boolean removeUnresolved = config.getDecodeResolve() ==
                    Config.DecodeResolve.REMOVE;
123                 -         StringBuilder sb = new StringBuilder();
124                 -         for (int i = 0; i < count; i++) {
125                 -             Symbol symbol = symbols[i];
                145 +                 for (int i = 0; i < symbolsCount; i++) {
                146 +                     Symbol symbol = symbols[i];
                147 +                     mask = 0;
126             148
127                 -             // Filter out redundant flags.
128                 -             if (count > 2) {
129                 -                 int mask = 0;
```

```
      149 +                // Combine the other flags.
      150 +                for (int j = 0; j < symbolsCount; j++) {
      151 +                    Symbol other = symbols[j];
130   152
131       -                // Combine the other flags.
132       -                for (int j = 0; j < count; j++) {
133       -                    Symbol other = symbols[j];
      153 +                    if (j != i) {
      154 +                        mask |= other.getValue().getData();
      155 +                    }
      156 +                }
134   157
135       -                    if (j != i) {
136       -                        mask |= other.getValue().getData();
      158 +                    // Skip if it doesn't add at least one unique bit.
      159 +                    if ((symbol.getValue().getData() & ~mask) == 0) {
      160 +                        continue;
137   161                  }
138       -                }
139   162
140       -                // Skip if it doesn't add at least one unique bit.
141       -                if ((symbol.getValue().getData() & ~mask) == 0) {
142       -                    continue;
      163 +                    filtered[filteredCount++] = symbol;
143   164                  }
144       -            }
145   165
146       -            // Append the flag to the format.
147       -            ResReference key = symbol.getKey();
148       -            ResEntrySpec keySpec = key.resolve();
149       -            if (sb.length() > 0) {
150       -                sb.append('|');
      166 +                symbols = filtered;
      167 +                symbolsCount = filteredCount;
151   168              }
152       -            if (keySpec != null) {
153       -                sb.append(keySpec.getName());
154       -            } else {
155       -                // #2836 - Support skipping items if the resource cannot be
          identified.
```

```
156     -                   if (removeUnresolved) {
157     -                       return null;
        169 +           }
        170 +
        171 +           if (symbolsCount == 0) {
        172 +               symbols = null;
        173 +           } else if (symbolsCount < symbols.length) {
        174 +               symbols = Arrays.copyOf(symbols, symbolsCount);
        175 +           }
        176 +
        177 +           mSymbolsCache.put(data, symbols);
        178 +           return symbols;
        179 +       }
        180 +
        181 +       @Override
        182 +       protected String formatValueToSymbols(ResItem value) throws
                    AndrolibException {
        183 +           if (!isSymbolValueType(value)) {
        184 +               return null;
        185 +           }
        186 +
        187 +           int data = ((ResPrimitive) value).getData();
        188 +           if (mFormatsCache == null) {
        189 +               // Lazily establish a formats cache for performance.
        190 +               mFormatsCache = new HashMap<>();
        191 +           } else if (mFormatsCache.containsKey(data)) {
        192 +               return mFormatsCache.get(data);
        193 +           }
        194 +
        195 +           Symbol[] symbols = getSymbols(data);
        196 +           String formatted = null;
        197 +
        198 +           if (symbols != null) {
        199 +               StringBuilder sb = new StringBuilder();
        200 +
        201 +               for (Symbol symbol : symbols) {
        202 +                   ResEntrySpec keySpec = symbol.getKey().resolve();
        203 +                   if (keySpec == null) {
        204 +                       continue;
158     205                     }
```

```
159  206
160       -                 sb.append(ResEntrySpec.MISSING_PREFIX + key.getId());
      207 +                 if (sb.length() > 0) {
      208 +                     sb.append('|');
      209 +                 }
      210 +                 sb.append(keySpec.getName());
161  211              }
      212 +
      213 +             formatted = sb.toString();
162  214          }
163  215
164       -         formatted = sb.toString();
165       -         mFormatCache.put(data, formatted);
      216 +         mFormatsCache.put(data, formatted);
166  217          return formatted;
167  218      }
168  219
169  220      @Override
170  221      protected void serializeSymbolsToValuesXml(XmlSerializer serial, ResEntry
           entry)
171  222              throws AndrolibException, IOException {
172       -         Config config = mParent.getPackage().getTable().getConfig();
173       -         boolean removeUnresolved = config.getDecodeResolve() ==
           Config.DecodeResolve.REMOVE;
174       -
175  223          for (Symbol symbol : mSymbols) {
176       -             ResReference key = symbol.getKey();
177       -             ResEntrySpec keySpec = key.resolve();
178       -             ResPrimitive value = symbol.getValue();
179       -
180       -             String name;
181       -             if (keySpec != null) {
182       -                 name = keySpec.getName();
183       -             } else {
184       -                 // #2836 - Support skipping items if the resource cannot be
           identified.
185       -                 if (removeUnresolved) {
186       -                     LOGGER.warning(String.format(
187       -                         "null flag reference: key=%s, value=%s", key, value));
188       -                     continue;
```

```
189     -                }
190     -
191     -                name = ResEntrySpec.MISSING_PREFIX + key.getId();
        224 +            ResEntrySpec keySpec = symbol.getKey().resolve();
        225 +            if (keySpec == null) {
        226 +                continue;
192     227              }
193     228
194     229              serial.startTag(null, "flag");
195     -                serial.attribute(null, "name", name);
196     -                serial.attribute(null, "value", value.encodeAsResXmlAttrValue());
        230 +            serial.attribute(null, "name", keySpec.getName());
        231 +            serial.attribute(null, "value",
                         symbol.getValue().encodeAsResXmlAttrValue());
197     232              serial.endTag(null, "flag");
198     233          }
199     234      }
```

.../brut/androlib/res/table/value/ResItem.java

```
        @@ -41,8 +41,6 @@ public abstract class ResItem extends ResValue {
41  41          STANDARD_TYPE_FORMATS.put("string", Sets.newHashSet("string"));
42  42      }
43  43
44  -   public abstract String getFormat();
45  -
46  44      public static ResItem parse(ResPackage pkg, int type, int data, String
            rawValue) {
47  45          switch (type) {
48  46              case TypedValue.TYPE_NULL:
        @@ -72,4 +70,6 @@ public static ResItem parse(ResPackage pkg, int type, int
        data, String rawValue)
72  70                  return null;
73  71          }
74  72      }
        73  +
        74  +   public abstract String getFormat();
75  75  }
```

...rut/androlib/res/table/value/ResPlural.java

```
@@ -56,8 +56,6 @@ public void serializeToValuesXml(XmlSerializer serial,
ResEntry entry)
56  56
57  57             for (RawItem item : mItems) {
58  58                 int key = item.getKey();
59      -              ResItem value = item.getValue();
60      -
61  59                 String quantity;
62  60                 switch (key) {
63  61                     case ATTR_OTHER:
@@ -83,6 +81,7 @@ public void serializeToValuesXml(XmlSerializer serial,
ResEntry entry)
83  81                         continue;
84  82                 }
85  83
    84  +              ResItem value = item.getValue();
86  85                 String body;
87  86                 if (value instanceof ResString) {
88  87                     body =
ResXmlEncoders.enumerateNonPositionalSubstitutionsIfRequired(
```

...brut/androlib/res/table/value/ResStyle.java

```
@@ -19,6 +19,7 @@
19  19  import brut.androlib.Config;
20  20  import brut.androlib.exceptions.AndrolibException;
21  21  import brut.androlib.exceptions.UndefinedResObjectException;
    22  + import brut.androlib.res.table.ResConfig;
22  23  import brut.androlib.res.table.ResEntry;
23  24  import brut.androlib.res.table.ResEntrySpec;
24  25  import brut.androlib.res.table.ResId;
@@ -78,12 +79,42 @@ public ResItem getValue() {
78  79              }
79  80          }
80  81
    82  +     @Override
```

```
+     public void resolveKeys() throws AndrolibException {
+         ResPackage pkg = mParent.getPackage();
+         Config config = pkg.getTable().getConfig();
+         boolean removeUnresolved = config.getDecodeResolve() ==
  Config.DecodeResolve.REMOVE;
+
+         for (Item item : mItems) {
+             ResReference key = item.getKey();
+             ResEntrySpec keySpec = key.resolve();
+             if (keySpec != null) {
+                 try {
+                     keySpec.getPackage().getDefaultEntry(keySpec.getId());
+                     continue;
+                 } catch (UndefinedResObjectException ignored) {
+                 }
+             }
+
+             ResId entryId = key.getId();
+
+             // #2836 - Skip item if the resource cannot be identified.
+             if (removeUnresolved || entryId.getPackageId() != pkg.getId()) {
+                 LOGGER.warning(String.format(
+                     "null style reference: key=%s, value=%s", key,
  item.getValue()));
+                 continue;
+             }
+
+             if (keySpec == null) {
+                 pkg.addEntrySpec(entryId, ResEntrySpec.MISSING_PREFIX +
  entryId);
+             }
+             pkg.addEntry(entryId, ResConfig.DEFAULT, ResAttribute.DEFAULT);
+         }
+     }
+
      @Override
      public void serializeToValuesXml(XmlSerializer serial, ResEntry entry)
              throws AndrolibException, IOException {
-         Config config = mParent.getPackage().getTable().getConfig();
-         boolean skipDuplicates = !config.isAnalysisMode();
```

```
86        -
87  118           serial.startTag(null, "style");
88  119           serial.attribute(null, "name", entry.getName());
89  120           if (mParent.resolve() != null) {
          @@ -92,52 +123,48 @@ public void serializeToValuesXml(XmlSerializer serial,
          ResEntry entry)
92  123               serial.attribute(null, "parent", "");
93  124           }
94  125
        126   +       Config config = mParent.getPackage().getTable().getConfig();
        127   +       boolean skipDuplicates = !config.isAnalysisMode();
95  128           Set<String> processedNames = new HashSet<>();
96  129           for (Item item : mItems) {
97        -             ResReference key = item.getKey();
98        -             ResEntrySpec keySpec = key.resolve();
99        -             ResItem value = item.getValue();
100       -
101       -             // #2836 - Support skipping items if the resource cannot be
          identified.
        130   +             ResEntrySpec keySpec = item.getKey().resolve();
102 131               if (keySpec == null) {
103       -                 LOGGER.warning(String.format("null style reference: key=%s,
          value=%s", key, value));
104 132                   continue;
105 133               }
106 134
107       -             String name = keySpec.getFullName(entry.getPackage(), true);
        135   +             String keyName = keySpec.getFullName(entry.getPackage(), true);
108 136
109 137               // #3400 - Skip duplicate items in styles.
110       -             if (skipDuplicates && processedNames.contains(name)) {
        138   +             if (skipDuplicates && processedNames.contains(keyName)) {
111 139                   continue;
112 140               }
113 141
114       -             String body = null;
        142   +             // We need the attribute entry's value to format the item's value.
        143   +             ResValue keyValue;
115 144               try {
```

```
116     -                // We need the attribute entry's value to format the item's value.
117     -                ResValue keyDefValue =
118     -                    keySpec.getPackage().getDefaultEntry(keySpec.getId()).getValue();
119     -
120     -                if (keyDefValue instanceof ResAttribute) {
121     -                    body = ((ResAttribute) keyDefValue).formatValue(value, true);
122     -                } else {
123     -                    LOGGER.warning("Unexpected style item key: " + keySpec);
124     -                }
        145 +                keyValue =
                             keySpec.getPackage().getDefaultEntry(keySpec.getId()).getValue();
125     146              } catch (UndefinedResObjectException ignored) {
        147 +                continue;
        148 +            }
        149 +
        150 +            ResItem value = item.getValue();
        151 +            String body = null;
        152 +            if (keyValue instanceof ResAttribute) {
        153 +                body = ((ResAttribute) keyValue).formatValue(value, true);
        154 +            } else {
        155 +                LOGGER.warning("Unexpected style item key: " + keySpec);
126     156              }
127     157              if (body == null) {
128     158                  // Fall back to default attribute.
129     159                  body = ResAttribute.DEFAULT.formatValue(value, true);
130     160              }
131     -            if (body == null) {
132     -                continue;
133     -            }
134     161
135     162              serial.startTag(null, "item");
136     -            serial.attribute(null, "name", name);
        163 +            serial.attribute(null, "name", keyName);
137     164              serial.text(body);
138     165              serial.endTag(null, "item");
139     166
140     -            processedNames.add(name);
```

```
      167 +             processedNames.add(keyName);
141   168             }
142   169
143   170             serial.endTag(null, "style");
```

...a/brut/androlib/res/xml/ResXmlEncoders.java

```
@@ -91,7 +91,8 @@ public static String encodeAsXmlValue(String str) {
91    91                return str;
92    92            }
93    93
94        -         StringBuilder sb = new StringBuilder(str.length() + 10);
      94  +         int len = str.length();
      95  +         StringBuilder sb = new StringBuilder(len + 10);
95    96
96    97            switch (str.charAt(0)) {
97    98                case '#':
@@ -105,7 +106,7 @@ public static String encodeAsXmlValue(String str) {
105   106           int startPos = 0;
106   107           boolean enclose = false;
107   108           boolean wasSpace = true;
108       -         for (int i = 0; i < str.length(); i++) {
      109 +         for (int i = 0; i < len; i++) {
109   110               char ch = str.charAt(i);
110   111               if (isInStyleTag) {
111   112                   if (ch == '>') {
@@ -140,7 +141,7 @@ public static String encodeAsXmlValue(String str) {
140   141                               break;
141   142                           }
142   143                           // Skip writing trailing \u0000 if we are at end of
      string.
143       -                   if ((sb.length() + 1) == str.length() && ch ==
      '\u0000') {
      144 +                   if ((sb.length() + 1) == len && ch == '\u0000') {
144   145                           continue;
145   146                       }
146   147                       sb.append(String.format("\\u%04x", (int) ch));
@@ -161,7 +162,8 @@ public static String encodeAsResXmlAttrValue(String str)
      {
```

```
161  162              return str;
162  163          }
163  164
164       -          StringBuilder sb = new StringBuilder(str.length() + 10);
     165  +          int len = str.length();
     166  +          StringBuilder sb = new StringBuilder(len + 10);
165  167
166  168          switch (str.charAt(0)) {
167  169              case '#':
     @@ -171,7 +173,7 @@ public static String encodeAsResXmlAttrValue(String str)
     {
171  173                  break;
172  174          }
173  175
174       -          for (int i = 0; i < str.length(); i++) {
     176  +          for (int i = 0; i < len; i++) {
175  177              char ch = str.charAt(i);
176  178              switch (ch) {
177  179                  case '\\':
     @@ -239,29 +241,27 @@ private static Pair<List<Integer>, List<Integer>>
     findSubstitutions(String str,
239  241          if (nonPosMax == -1) {
240  242              nonPosMax = Integer.MAX_VALUE;
241  243          }
242       -          int pos;
243       -          int pos2 = 0;
     244  +
244  245          List<Integer> nonPositional = new ArrayList<>();
245  246          List<Integer> positional = new ArrayList<>();
246       -
247  247          if (str == null) {
248  248              return Pair.of(nonPositional, positional);
249  249          }
250  250
251       -          int length = str.length();
252       -
     251  +          int len = str.length();
     252  +          int pos, pos2 = 0;
253  253          while ((pos = str.indexOf('%', pos2)) != -1) {
254  254              pos2 = pos + 1;
```

```
255       -             if (pos2 == length) {
      255 +             if (pos2 == len) {
256   256                   nonPositional.add(pos);
257   257                   break;
258   258               }
259   259               char ch = str.charAt(pos2++);
260   260               if (ch == '%') {
261   261                   continue;
262   262               }
263       -             if (ch >= '0' && ch <= '9' && pos2 < length) {
264       -                 while ((ch = str.charAt(pos2++)) >= '0' && ch <= '9' && pos2 < length);
      263 +             if (ch >= '0' && ch <= '9' && pos2 < len) {
      264 +                 while ((ch = str.charAt(pos2++)) >= '0' && ch <= '9' && pos2 < len);
265   265                   if (ch == '$') {
266   266                       positional.add(pos);
267   267                       continue;
```

...va/brut/androlib/BuildAndDecodeApkTest.java

```
@@ -438,7 +438,7 @@ public void drawableXhdpiTest() throws BrutException {
438   438
439   439       @Test
440   440       public void ninePatchImageColorTest() throws IOException {
441       -         String fileName = "res/drawable-xhdpi/9patch.9.png";
      441 +         String fileName = "res/drawable-xhdpi/ninepatch.9.png";
442   442
443   443           File control = new File(sTestOrigDir, fileName);
444   444           File test = new File(sTestNewDir, fileName);
```

...t/java/brut/androlib/DecodeResolveTest.java

```
@@ -58,47 +58,47 @@ public void decodeResolveRemoveTest() throws
BrutException {
58    58        }
59    59
60    60        @Test
61        -     public void decodeResolveDummyTest() throws BrutException {
```

```
62        -         sConfig.setDecodeResolve(Config.DecodeResolve.DUMMY);
          61  +     public void decodeResolveKeepTest() throws BrutException {
          62  +         sConfig.setDecodeResolve(Config.DecodeResolve.KEEP);
63        63
64        64            ExtFile testApk = new ExtFile(sTmpDir, TEST_APK);
65        -         ExtFile testDir = new ExtFile(testApk + ".out.dummies");
          65  +         ExtFile testDir = new ExtFile(testApk + ".out.leave");
66        66            new ApkDecoder(testApk, sConfig).decode(testDir);
67        67
68        68            assertTrue(new File(testDir, "res/values/strings.xml").isFile());
69        69
70        -         File attrXml = new File(testDir, "res/values/attrs.xml");
71        -         Document attrDocument = loadDocument(attrXml);
          70  +         Document attrDocument = loadDocument(new File(testDir,
                    "res/values/attrs.xml"));
72        71            assertEquals(4, attrDocument.getElementsByTagName("enum").getLength());
73        72
74        -         File colorXml = new File(testDir, "res/values/colors.xml");
75        -         Document colorDocument = loadDocument(colorXml);
          73  +         Document colorDocument = loadDocument(new File(testDir,
                    "res/values/colors.xml"));
76        74            assertEquals(8,
                    colorDocument.getElementsByTagName("color").getLength());
77        -         assertEquals(1,
                    colorDocument.getElementsByTagName("item").getLength());
          75  +         assertEquals(0,
                    colorDocument.getElementsByTagName("item").getLength());
78        76
79        -         File publicXml = new File(testDir, "res/values/public.xml");
80        -         Document publicDocument = loadDocument(publicXml);
          77  +         Document publicDocument = loadDocument(new File(testDir,
                    "res/values/public.xml"));
81        78            assertEquals(22,
                    publicDocument.getElementsByTagName("public").getLength());
82        79        }
83        80
84        81        @Test
85        -     public void decodeResolveKeepTest() throws BrutException {
86        -         sConfig.setDecodeResolve(Config.DecodeResolve.KEEP);
          82  +     public void decodeResolveDummyTest() throws BrutException {
```

```
                83 +         sConfig.setDecodeResolve(Config.DecodeResolve.DUMMY);
87              84
88              85           ExtFile testApk = new ExtFile(sTmpDir, TEST_APK);
89                 -         ExtFile testDir = new ExtFile(testApk + ".out.leave");
                86 +         ExtFile testDir = new ExtFile(testApk + ".out.dummies");
90              87           new ApkDecoder(testApk, sConfig).decode(testDir);
91              88
92              89           assertTrue(new File(testDir, "res/values/strings.xml").isFile());
93              90
94                 -         Document attrDocument = loadDocument(new File(testDir,
                   "res/values/attrs.xml"));
                91 +         File attrXml = new File(testDir, "res/values/attrs.xml");
                92 +         Document attrDocument = loadDocument(attrXml);
95              93           assertEquals(4, attrDocument.getElementsByTagName("enum").getLength());
96              94
97                 -         Document colorDocument = loadDocument(new File(testDir,
                   "res/values/colors.xml"));
                95 +         File colorXml = new File(testDir, "res/values/colors.xml");
                96 +         Document colorDocument = loadDocument(colorXml);
98              97           assertEquals(8,
                   colorDocument.getElementsByTagName("color").getLength());
99                 -         assertEquals(0,
                   colorDocument.getElementsByTagName("item").getLength());
                98 +         assertEquals(1,
                   colorDocument.getElementsByTagName("item").getLength());
100             99
101                -         Document publicDocument = loadDocument(new File(testDir,
                   "res/values/public.xml"));
102                -         assertEquals(21,
                   publicDocument.getElementsByTagName("public").getLength());
               100 +         File publicXml = new File(testDir, "res/values/public.xml");
               101 +         Document publicDocument = loadDocument(publicXml);
               102 +         assertEquals(23,
                   publicDocument.getElementsByTagName("public").getLength());
103             103      }
104             104   }
```

...ndrolib/ResourceDirectoryTraversalTest.java

```
                @@ -16,6 +16,7 @@
16              16       */
```

```
17  17  package brut.androlib;
18  18
    19  + import brut.androlib.res.table.ResEntrySpec;
19  20  import brut.common.BrutException;
20  21  import brut.directory.ExtFile;
21  22
    @@ -33,13 +34,13 @@ public static void beforeClass() throws Exception {
33  34      }
34  35
35  36      @Test
36      -     public void checkIfMaliciousRawFileIsDisassembledProperly() throws
        BrutException {
    37  +     public void checkIfMaliciousRawFileRenamed() throws BrutException {
37  38          sConfig.setForced(true);
38  39
39  40          ExtFile testApk = new ExtFile(sTmpDir, TEST_APK);
40  41          ExtFile testDir = new ExtFile(testApk + ".out");
41  42          new ApkDecoder(testApk, sConfig).decode(testDir);
42  43
43      -         assertTrue(new File(testDir, "res/raw/poc").exists());
    44  +         assertTrue(new File(testDir, "res/raw/" + ResEntrySpec.RENAMED_PREFIX +
        "0x7f040000").exists());
44  45      }
45  46  }
```

…/testapp/res/drawable-xhdpi/9patch.9.png → …stapp/res/drawable-xhdpi/ninepatch.9.png

File renamed without changes.

….j.util/src/main/java/brut/util/BrutIO.java

```
    @@ -103,9 +103,4 @@ public static String sanitizePath(File baseDir, String
    path) throws InvalidPathE
103 103
104 104          return basePath.relativize(resolvedPath).toString();
105 105      }
106     -
107     -     public static boolean detectPossibleDirectoryTraversal(String path) {
108     -         return path.contains("../") || path.contains("/..")
109     -                 || path.contains("..\\") || path.contains("\\..");
```

```
110       -     }
111  106      }
```

...rc/main/java/brut/xmlpull/MXSerializer.java

```
     @@ -266,13 +266,13 @@ private void rebuildIndentationBuf() {
266  266              }
267  267              int bufPos = 0;
268  268              if (writeLineSeparator) {
269       -              for (int i = 0; i < lineSeparator.length(); i++) {
     269  +              for (int i = 0, n = lineSeparator.length(); i < n; i++) {
270  270                    indentationBuf[bufPos++] = lineSeparator.charAt(i);
271  271                }
272  272            }
273  273            if (writeIndentation) {
274  274                for (int i = 0; i < maxIndentLevel; i++) {
275       -                for (int j = 0; j < indentationString.length(); j++) {
     275  +                for (int j = 0, n = indentationString.length(); j < n; j++) {
276  276                        indentationBuf[bufPos++] = indentationString.charAt(j);
277  277                    }
278  278                }
     @@ -894,8 +894,9 @@ private void writeAttributeValue(String value) throws
     IOException {
894  894                return;
895  895            }
896  896            // .[&, < and " escaped],
     897  +         int len = value.length();
897  898            int pos = 0;
898       -         for (int i = 0; i < value.length(); i++) {
     899  +         for (int i = 0; i < len; i++) {
899  900                char ch = value.charAt(i);
900  901                if (ch == '&') {
901  902                    if (i > pos) {
     @@ -930,8 +931,8 @@ private void writeAttributeValue(String value) throws
     IOException {
930  931                        pos = i + 1;
931  932                    } else {
932  933                        if (TRACE_ESCAPING) {
933       -                        System.err.println(getClass().getName() + " DEBUG
     ATTR value.len=" + value.length()
934       -                                + " " + printable(value));
```

```
+                             System.err.println(getClass().getName() + " DEBUG ATTR value.len=" + len + " "
+                                     + printable(value));
                         }
                         throw new IllegalStateException(
                                 "character " + printable(ch) + " (" + Integer.toString(ch) + ") is not allowed in output"
@@ -951,8 +952,9 @@ private void writeElementContent(String text) throws IOException {
             }
 
             // escape '<', '&', ']]>', <32 if necessary
+            int len = text.length();
             int pos = 0;
-            for (int i = 0; i < text.length(); i++) {
+            for (int i = 0; i < len; i++) {
                 // TODO: check if doing char[] text.getChars() would be faster than
                 // getCharAt(i) ...
                 char ch = text.charAt(i);
@@ -964,7 +966,7 @@ private void writeElementContent(String text) throws IOException {
                     }
                 } else {
                     if (ch == '&') {
-                        if (!(i < text.length() - 3 && text.charAt(i + 1) == 'l'
+                        if (!(i < len - 3 && text.charAt(i + 1) == 'l'
                                 && text.charAt(i + 2) == 't' && text.charAt(i + 3) == ';')) {
                             if (i > pos) {
                                 write(text.substring(pos, i));
@@ -990,8 +992,8 @@ private void writeElementContent(String text) throws IOException {
                             // fallthrough
                         } else {
                             if (TRACE_ESCAPING) {
-                                System.err.println(getClass().getName() + " DEBUG TEXT value.len=" + text.length()
-                                        + " " + printable(text));
```

```
        995 +                                System.err.println(getClass().getName() + " DEBUG
            TEXT value.len=" + len + " "
        996 +                                        + printable(text));
995     997                              }
996     998                              throw new IllegalStateException("character " +
            Integer.toString(ch)
997     999                                      + " is not allowed in output" + getLocation()
        ↕   @@ -1066,9 +1068,10 @@ private static String printable(String str) {
1066    1068             if (str == null) {
1067    1069                 return "null";
1068    1070             }
1069        -            StringBuffer retval = new StringBuffer(str.length() + 16);
        1071 +          int len = str.length();
        1072 +          StringBuffer retval = new StringBuffer(len + 16);
1070    1073             retval.append('\'');
1071        -            for (int i = 0; i < str.length(); i++) {
        1074 +          for (int i = 0; i < len; i++) {
1072    1075                 addPrintable(retval, str.charAt(i));
1073    1076             }
1074    1077             retval.append('\'');
        ↕
```

….yaml/src/main/java/brut/yaml/YamlLine.java

```
        ↕   @@ -39,7 +39,8 @@ public YamlLine(String line) {
39      39                   return;
40      40               }
41      41               // count indent - space only
42          -            for (int i = 0; i < line.length(); i++) {
            42 +         int len = line.length();
            43 +         for (int i = 0; i < len; i++) {
43      44                   if (line.charAt(i) == ' ') {
44      45                       indent++;
45      46                   } else {
        ↕
```

…n/java/brut/yaml/YamlStringEscapeUtils.java

```
        ↕   @@ -63,9 +63,8 @@ private static void escapeJavaStyleString(Writer writer,
            String str) throws IOEx
```

```
63  63          if (str == null) {
64  64              return;
65  65          }
66      -       int sz;
67      -       sz = str.length();
68      -       for (int i = 0; i < sz; i++) {
    66  +       int len = str.length();
    67  +       for (int i = 0; i < len; i++) {
69  68              char ch = str.charAt(i);
70  69              // "[^\t\n\r\u0020-\u007E\u0085\u00A0-\uD7FF\uE000-\uFFFD]"
71  70              // handle unicode
```

Comments 0

Please sign in to comment.