EXHIBIT 2

EXHIBIT C
Supplemental Infringement Contentions for the '702 Patent

NOTE: The infringement evidence cited below is exemplary and not exhaustive. The cited examples are taken from Android 2.2, 2.3, and Google's Android websites. Oracle's infringement contentions apply to all versions of Android having similar or nearly identical code or documentation, including past and expected future releases. Although Oracle's investigation is ongoing, the '702 patent is infringed by all versions of Android from Oct. 21, 2008 to the present, including Android 1.1, 1.5 ("Cupcake"), 1.6 ("Donut"), 2.0/2.1 ("Éclair"), 2.2 ("Froyo"), and 2.3 ("Gingerbread").

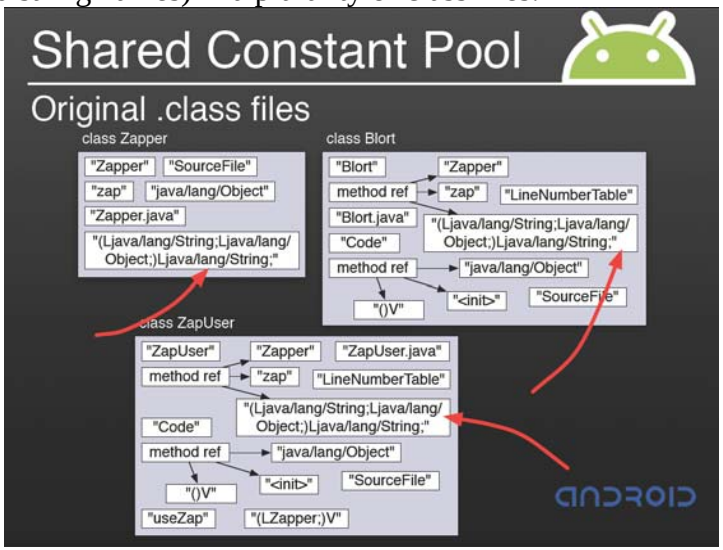
The cited source code examples are taken from http://android.git.kernel.org/. The citations are shortened and mirror the file paths shown in http://android.git.kernel.org/. For example, "dalvik\vm\native\InternalNative.c" maps to "[platform/dalvik.git] / vm / native / InternalNative.c" (accessible at http://android.git.kernel.org/?p=platform/dalvik.git;a=blob;f=vm/native/InternalNative.c). Google has apparently made modifications to certain source code files and directories since Oracle's Preliminary Infringement Contentions were served on December 2, 2010. As such, file paths may in some cases refer to earlier versions of Android than what is immediately available at http://android.git.kernel.org/.

It appears that the Android git source code repository (accessible through http://android.git.kernel.org/) was created on or around Oct. 21, 2008. As such, the list of infringing Android versions may be expanded based on what Oracle learns about earlier Android versions.

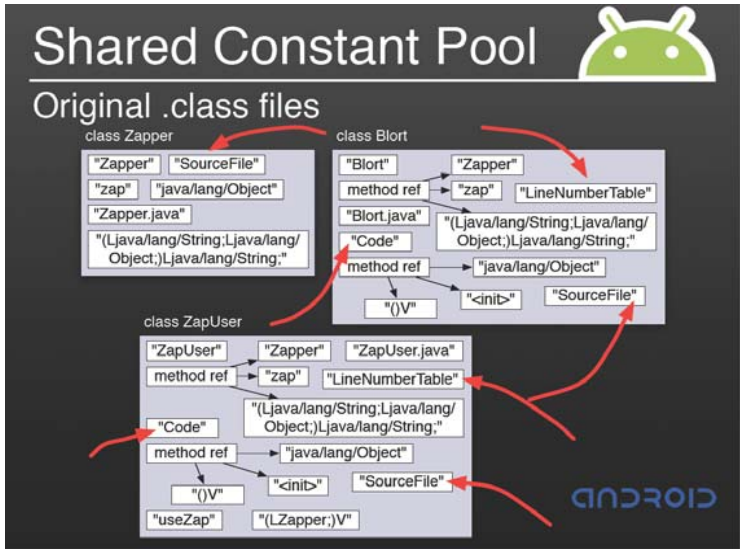
The asserted claims include apparatus, method, and computer-readable medium claims. Anyone who makes, uses, offers to sell, sells, or imports the computers running the Android SDK within or into the United States directly infringes the apparatus claims. Similarly, anyone who engages in the above conduct with respect to storage devices containing the Android SDK directly infringes the computer-readable medium claims. Anyone who uses the Android SDK directly infringes the method claims. Thus Google and its downstream licensees, including device manufacturers and application developers, directly infringe. Google induces and contributes to infringement of all asserted claims by distributing the Android SDK with the intention that it will be executed by developers. The Android code cited below necessarily infringes because developers must run the Android dx tool to build Android applications, and generate Android bytecode and .dex files, and run the Dalvik virtual machine to test them. The Android SDK is a tool used purely to build and test Android programs. It is neither a staple article nor capable of substantial non-infringing use. Google supplies the Android SDK in and from the United States.

When infringement evidence first presented with respect to one claim is referred to with respect to another, the evidence is applicable because it is not limited to a particular form of infringement.

The '702 Patent	Infringed By
1. A method of pre-processing class files comprising:	The Android dx tool involves a method of pre-processing .class files into a Dalvik executable format (.dex) file. **"dx** The dx tool lets you generate Android bytecode from .class files. The tool converts target files and/or directories to Dalvik executable format (.dex) files, so that they can run in the Android environment." Android Developer Tools available at http://developer.android.com/guide/developing/tools/othertools.html The method of pre-processing class files into a .dex file that can be interpreted by the Dalvik Virtual Machine (Dalvik VM) is explained in the Dalvik VM video presentation and related presentation from Google I/O 2008, dated 5/29/2008. *See* Google I/O 2008 Video entitled "*Google I/O 2008 - Dalvik Virtual Machine Internals*," presented by Dan Bornstein, http://developer.android.com/videos/index.html#v=ptjedOZEXPM ("Dalvik Video"), at time 5:45–10:45. *See also* Google I/O 2008 Presentation Slides, entitled, "*Dalvik Virtual Machine Internals, Google I/O 2008,*" presented by Dan Bornstein ("Dalvik Presentation") at slides 11-22, available at http://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf?attredirects=0. In the Android source code, *see generally*: "Classes for translating Java classfiles into Dalvik classes. PACKAGES USED: • com.android.dx.cf.code

The '702 Patent	Infringed By
	• com.android.dx.cf.direct • com.android.dx.cf.iface • com.android.dx.dex.code • com.android.dx.dex.file • com.android.dx.rop.code • com.android.dx.rop.cst • com.android.dx.util" dalvik\dx\src\com\android\dx\dex\cf\package.html.
determining plurality of duplicated elements in a plurality of class files;	The Android dx tool determines a plurality of duplicated elements in a plurality of class files, as explained in the Dalvik Video at time 7:50-8:45 and Dalvik Presentation, slides 18-19. The Dalvik Presentation shows the determination of a plurality of duplicated elements (e.g., class signatures and string names) in a plurality of class files:  (Dalvik Presentation, slide 18) (Shows identification of common class signatures in the class files)

The '702 Patent	Infringed By



(Dalvik Presentation, slide 19)
(Shows identification of common string names in the class files)

In the Android source code, *see also generally*:

> "Interfaces and implementation of things related to the constant pool.
> PACKAGES USED:
> * com.android.dx.rop.type
> * com.android.dx.util"

dalvik/dx/src/com/android/dx/rop/cst/package.html.

See also DexFile.java:

```
440
 441     /**
 442      * Gets the {@link IndexedItem} corresponding to the given constant,
 443      * if it is a constant that has such a correspondence, or return
```

The ’702 Patent	Infringed By

```
 444      * {@code null} if it isn't such a constant. This will throw
 445      * an exception if the given constant <i>should</i> have been found
 446      * but wasn't.
 447      *
 448      * @param cst {@code non-null;} the constant to look up
 449      * @return {@code null-ok;} its corresponding item, if it has a
corresponding
 450      * item, or {@code null} if it's not that sort of constant
 451      */
 452     /*package*/ IndexedItem findItemOrNull(Constant cst) {
 453         IndexedItem item;
 454
 455         if (cst instanceof CstString) {
 456             return stringIds.get(cst);
 457         } else if (cst instanceof CstType) {
 458             return typeIds.get(cst);
 459         } else if (cst instanceof CstBaseMethodRef) {
 460             return methodIds.get(cst);
 461         } else if (cst instanceof CstFieldRef) {
 462             return fieldIds.get(cst);
 463         } else {
 464             return null;
 465         }
 466     }
 467
 468     /**
 469      * Returns the contents of this instance as a {@code .dex} file,
 470      * in a {@link ByteArrayAnnotatedOutput} instance.
 471      *
 472      * @param annotate whether or not to keep annotations
 473      * @param verbose if annotating, whether to be verbose
 474      * @return {@code non-null;} a {@code .dex} file for this instance
 475      */
 476     private ByteArrayAnnotatedOutput toDex0(boolean annotate,
 477             boolean verbose) {
 478         /*
 479          * The following is ordered so that the prepare() calls which
 480          * add items happen before the calls to the sections that get
 481          * added to.
 482          */
 483
 484         classDefs.prepare();
 485         classData.prepare();
 486         wordData.prepare();
```

The '702 Patent	Infringed By

```
 487         byteData.prepare();
 488         methodIds.prepare();
 489         fieldIds.prepare();
 490         protoIds.prepare();
 491         typeLists.prepare();
 492         typeIds.prepare();
 493         stringIds.prepare();
 494         stringData.prepare();
 495         header.prepare();
 496
 497         // Place the sections within the file.
 498
 499         int count = sections.length;
 500         int offset = 0;
 501
 502         for (int i = 0; i < count; i++) {
 503             Section one = sections[i];
 504             int placedAt = one.setFileOffset(offset);
 505             if (placedAt < offset) {
 506                 throw new RuntimeException("bogus placement for section " +
i);
 507             }
 508
 509             try {
 510                 if (one == map) {
 511                     /*
 512                      * Inform the map of all the sections, and add it
 513                      * to the file. This can only be done after all
 514                      * the other items have been sorted and placed.
 515                      */
 516                     MapItem.addMap(sections, map);
 517                     map.prepare();
 518                 }
 519
 520                 if (one instanceof MixedItemSection) {
 521                     /*
 522                      * Place the items of a MixedItemSection that just
 523                      * got placed.
 524                      */
 525                     ((MixedItemSection) one).placeItems();
 526                 }
 527
 528                 offset = placedAt + one.writeSize();
 529             } catch (RuntimeException ex) {
```

The ’702 Patent	Infringed By

```
530                 throw ExceptionWithContext.withContext(ex,
531                         "...while writing section " + i);
532             }
533         }
534
535         // Write out all the sections.
536
537         fileSize = offset;
538         byte[] barr = new byte[fileSize];
539         ByteArrayAnnotatedOutput out = new ByteArrayAnnotatedOutput(barr);
540
541         if (annotate) {
542             out.enableAnnotations(dumpWidth, verbose);
543         }
544
545         for (int i = 0; i < count; i++) {
546             try {
547                 Section one = sections[i];
548                 int zeroCount = one.getFileOffset() - out.getCursor();
549                 if (zeroCount < 0) {
550                     throw new ExceptionWithContext("excess write of " +
551                             (-zeroCount));
552                 }
553                 out.writeZeroes(one.getFileOffset() - out.getCursor());
554                 one.writeTo(out);
555             } catch (RuntimeException ex) {
556                 ExceptionWithContext ec;
557                 if (ex instanceof ExceptionWithContext) {
558                     ec = (ExceptionWithContext) ex;
559                 } else {
560                     ec = new ExceptionWithContext(ex);
561                 }
562                 ec.addContext("...while writing section " + i);
563                 throw ec;
564             }
565         }
566
567         if (out.getCursor() != fileSize) {
568             throw new RuntimeException("foreshortened write");
569         }
570
571         // Perform final bookkeeping.
572
573         calcSignature(barr);
```

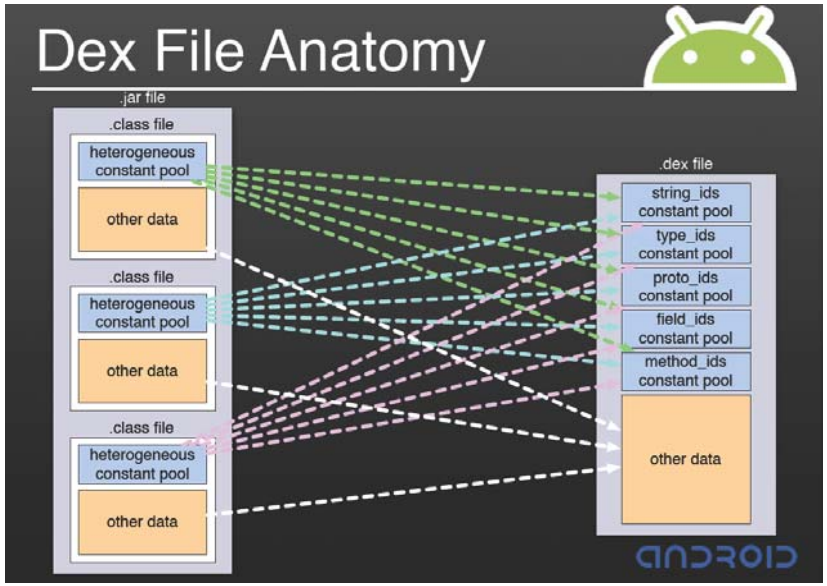
The ’702 Patent	Infringed By

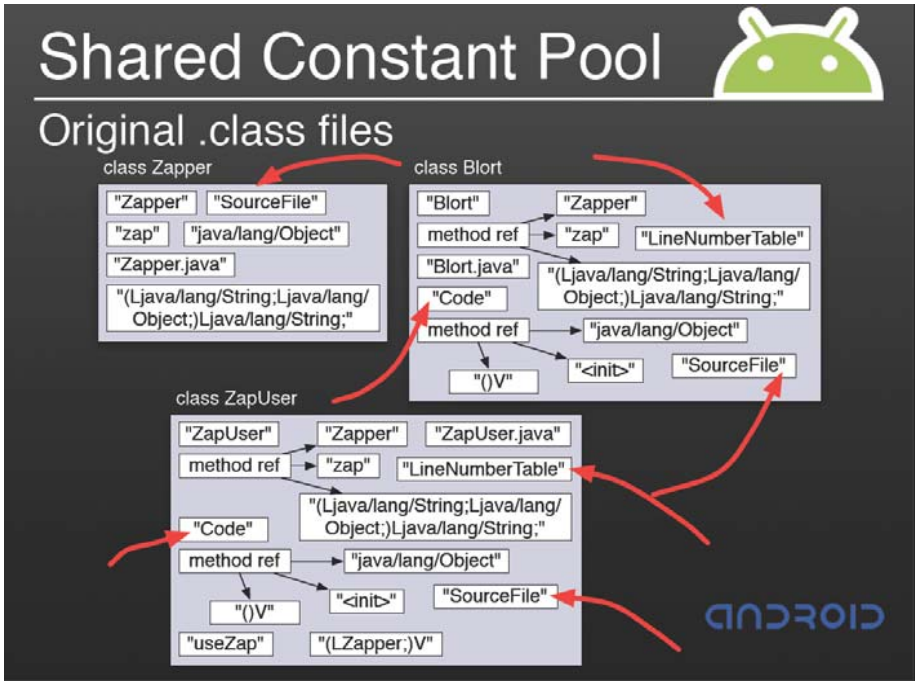
```
574         calcChecksum(barr);
575
576         if (annotate) {
577             wordData.writeIndexAnnotation(out, ItemType.TYPE_CODE_ITEM,
578                     "\nmethod code index:\n\n");
579             getStatistics().writeAnnotation(out);
580             out.finishAnnotating();
581         }
582
583         return out;
584     }
585
586     /**
587      * Generates and returns statistics for all the items in the file.
588      *
589      * @return {@code non-null;} the statistics
590      */
591     public Statistics getStatistics() {
592         Statistics stats = new Statistics();
593
594         for (Section s : sections) {
595             stats.addAll(s);
596         }
597
598         return stats;
599     }
600
601     /**
602      * Calculates the signature for the {@code .dex} file in the
603      * given array, and modify the array to contain it.
604      *
605      * @param bytes {@code non-null;} the bytes of the file
606      */
607     private static void calcSignature(byte[] bytes) {
608         MessageDigest md;
609
610         try {
611             md = MessageDigest.getInstance("SHA-1");
612         } catch (NoSuchAlgorithmException ex) {
613             throw new RuntimeException(ex);
614         }
615
616         md.update(bytes, 32, bytes.length - 32);
617
```

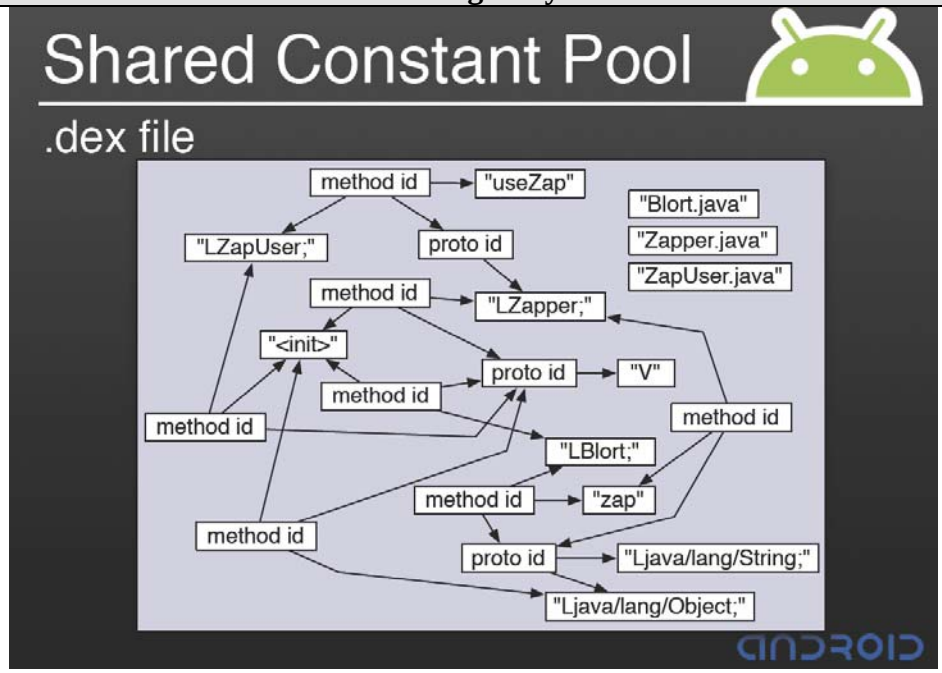
The '702 Patent	Infringed By

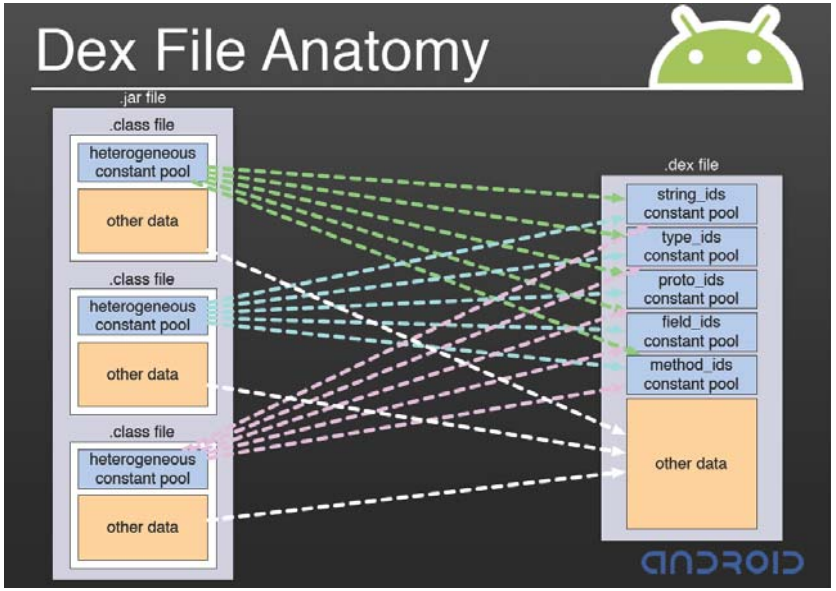
	618 try {
	619 int amt = md.digest(bytes, 12, 20);
	620 if (amt != 20) {
	621 throw new RuntimeException("unexpected digest write: " + amt +
	622 " bytes");
	623 }
	624 } catch (DigestException ex) {
	625 throw new RuntimeException(ex);
	626 }
	627 }
	628
	629 /**
	630 * Calculates the checksum for the {@code .dex} file in the
	631 * given array, and modify the array to contain it.
	632 *
	633 * @param bytes {@code non-null;} the bytes of the file
	634 */
	635 private static void calcChecksum(byte[] bytes) {
	636 Adler32 a32 = new Adler32();
	637
	638 a32.update(bytes, 12, bytes.length - 12);
	639
	640 int sum = (int) a32.getValue();
	641
	642 bytes[8] = (byte) sum;
	643 bytes[9] = (byte) (sum >> 8);
	644 bytes[10] = (byte) (sum >> 16);
	645 bytes[11] = (byte) (sum >> 24);
	646 }
	647 }
	
	dalvik/dx/src/com/android/dx/dex/file/DexFile.java.
	
	See also:
	dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java
	dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java
	dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java

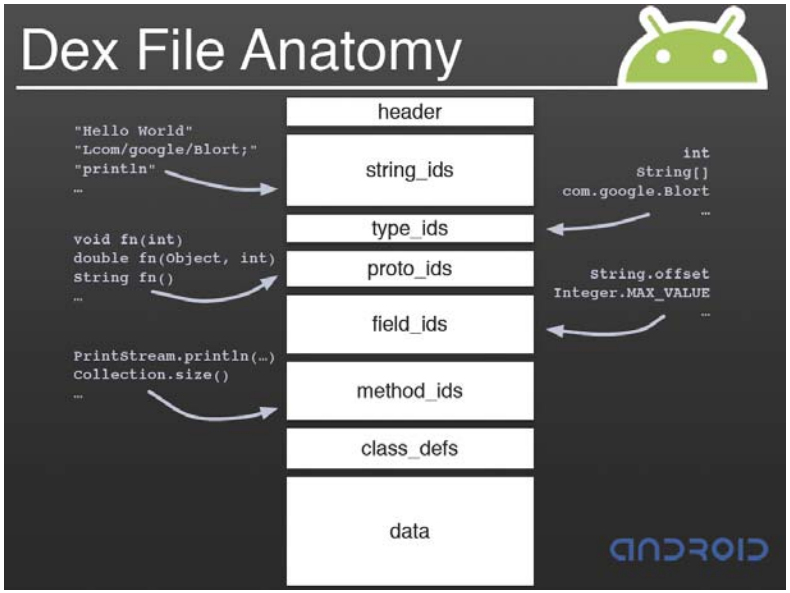
forming a shared table comprising said plurality of duplicated	The Android dx tool forms a shared table of the duplicated elements from the plurality of class files. This process is explained in the Dalvik Video at time 7:20–9:25 and Dalvik

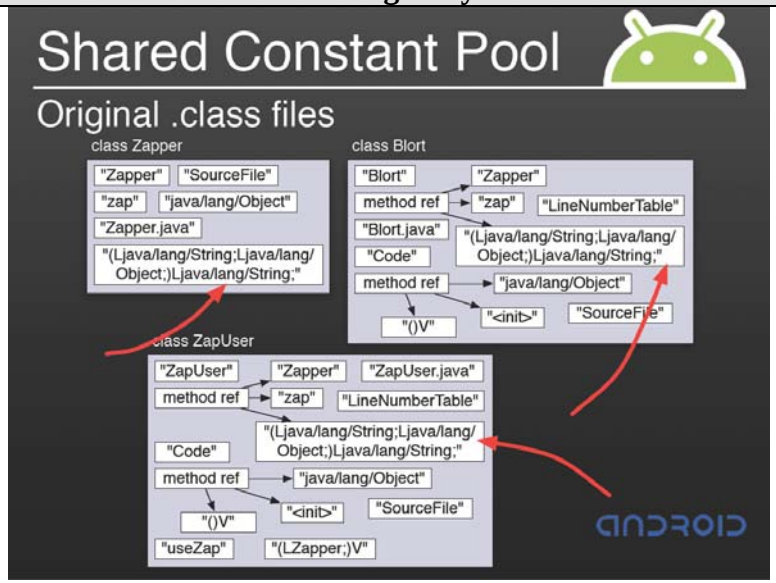
The '702 Patent	Infringed By
elements;	Presentation, slides 15-20, where the recited shared table includes, e.g., one or more of the "string_ids constant pool," "type_ids constant pool," "proto_ids constant pool," "field_ids constant pool," and "method_ids constant pool." The Dalvik Presentation shows the elements of the class files combining into a shared constant pool (shared tables) in the .dex file.  (Dalvik Presentation, slide 15) In the illustration above, each of "string_ids," "type_ids" and "method_ids" are examples of the shared tables (or, equivalently, a collective shared table). In addition, the discussion of the "Shared Constant Pool" in the Dalvik Video explains that the duplicated elements in the class files are consolidated into the shared constant pool (shared table) of the .dex file. *See* Dalvik Presentation, slides 15-21. For example, slide 19 of the Dalvik Presentation shows the separate class files having

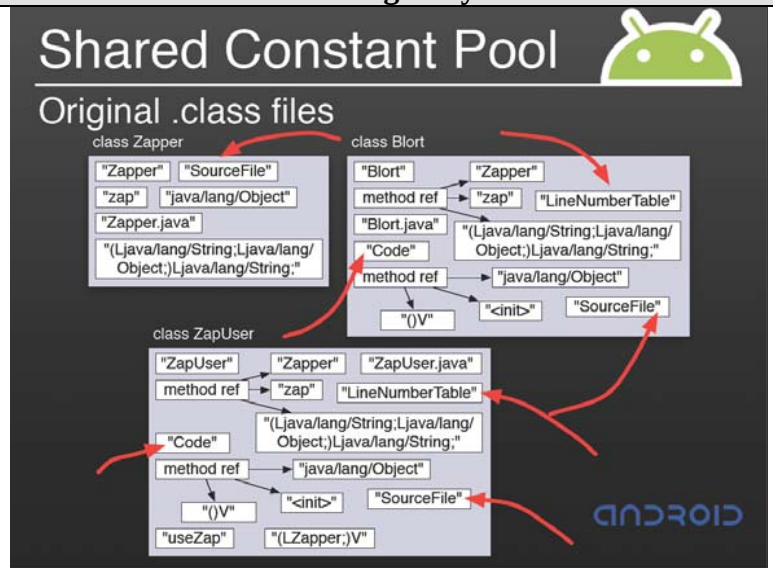
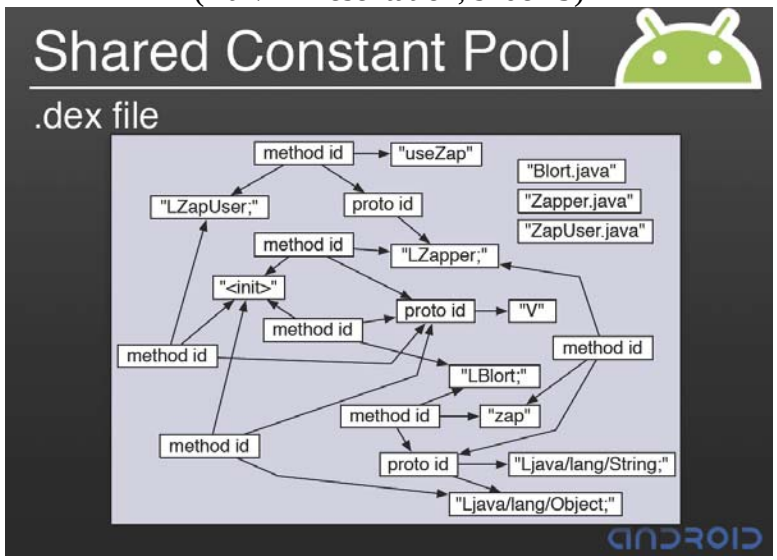
The '702 Patent	Infringed By
	duplicated elements.  (Dalvik Presentation, slide 19) Next, slide 20 of the Dalvik Presentation shows a representation of the class files after being processed into a single .dex file, with the duplicate elements removed; the elements are then stored in a shared constant pool (shared table):

The '702 Patent	Infringed By
	 (Dalvik Presentation, slide 20) In the Android source code, *see also generally*: "Interfaces and implementation of things related to the constant pool. PACKAGES USED: * com.android.dx.rop.type * com.android.dx.util" dalvik/dx/src/com/android/dx/rop/cst/package.html. *See also:* dalvik/dx/src/com/android/dx/dex/file/DexFile.java dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java

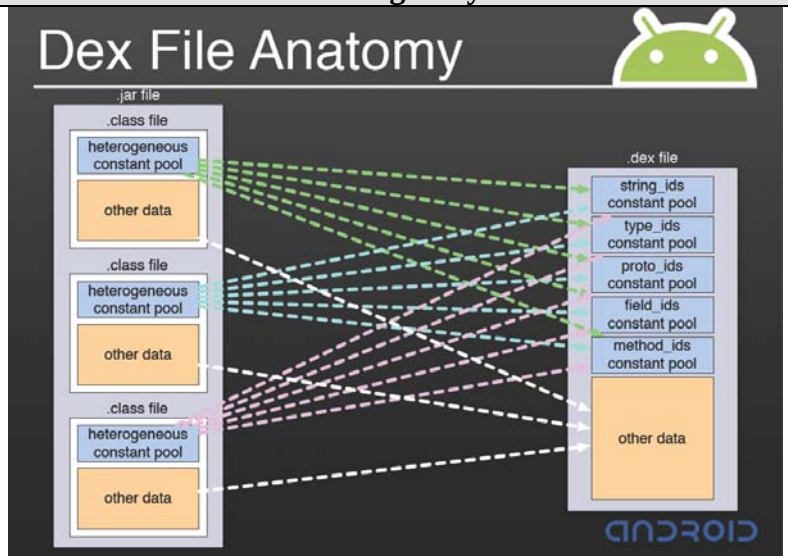
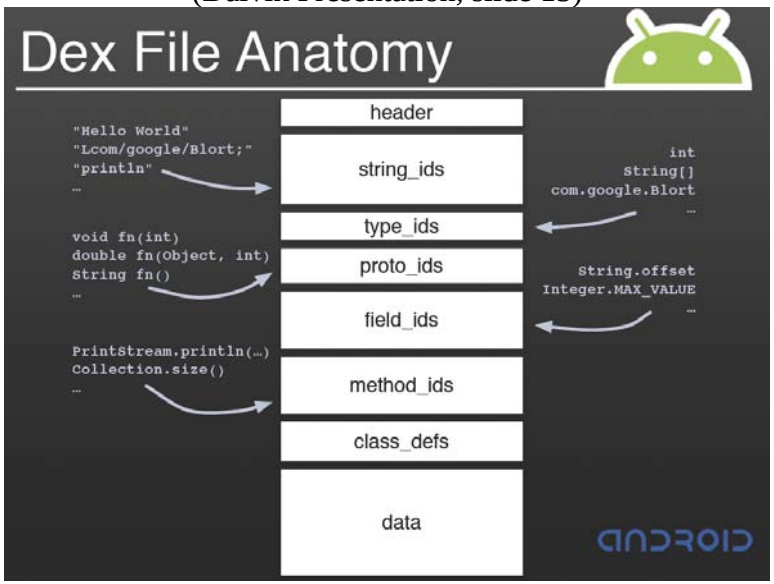
The '702 Patent	Infringed By
	dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java
removing said duplicated elements from said plurality of class files to obtain a plurality of reduced class files; and	The Android dx tool removes the duplicated elements from the plurality of class files (e.g., as part of the process of forming the .dex file) and obtains a plurality of reduced class files (the reduced class files including a subset of the code and data contained in the class files). This process, and contents of the reduced class file, is clearly explained and illustrated in the Dalvik Video at time 7:20–9:25 and Dalvik Presentation, slides 15-20. The Dalvik Presentation shows the class files combining into a shared constant pool (shared table) in the .dex file, whereby duplicated elements are removed from the class files when using a subset of the code and data contained in the class files, i.e., the reduced class files, to form the .dex file.  (Dalvik Presentation, slide 15)

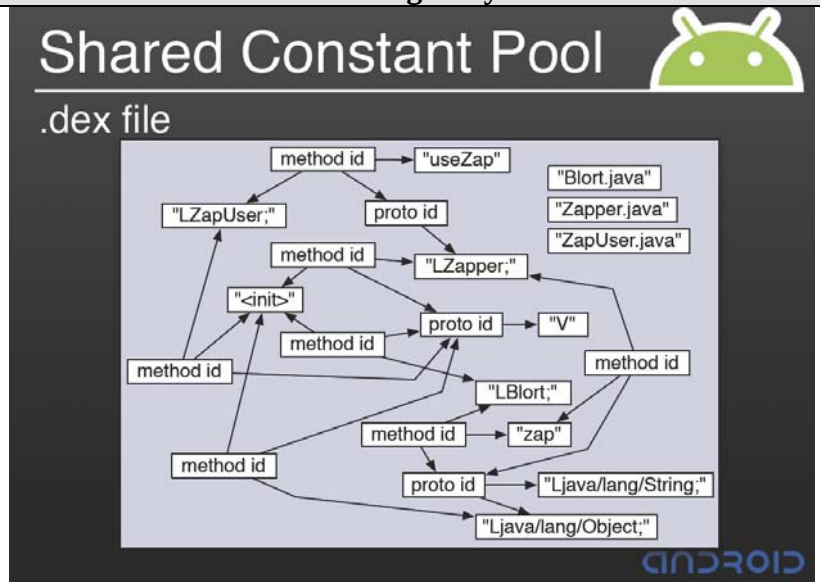
The '702 Patent	Infringed By
	The original class files are combined into a single .dex file, which includes a plurality of reduced class files (i.e., a subset of code and data of the class files, with duplicates removed). This is also illustrated in slide 11 of the Dalvik presentation, which shows the anatomy of a .dex file:  (Dalvik Presentation, slide 11) Next, slides 18-20 of the Dalvik Presentation show the removal of the duplicated elements of the plurality of class files such that the resulting .dex file contains only one copy of each element in its shared constant pool (shared table).

The '702 Patent	Infringed By
	 (Dalvik Presentation, slide 18)

The '702 Patent	Infringed By
	 (Dalvik Presentation, slide 19)  (Dalvik Presentation, slide 20)

The '702 Patent	Infringed By
	In the Android source code, *see also generally*: "Interfaces and implementation of things related to the constant pool. PACKAGES USED: * com.android.dx.rop.type * com.android.dx.util" dalvik/dx/src/com/android/dx/rop/cst/package.html. *See also:* dalvik/dx/src/com/android/dx/dex/file/DexFile.java dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java
forming a multi-class file comprising said plurality of reduced class files and said shared table.	As explained above, the Android dx tool forms a multi-class file—the .dex file—comprising the reduced class files and a shared constant pool (shared table) such that duplicate elements have been removed. This process is explained in the Dalvik Video at time 7:20–9:25 and Dalvik Presentation, slides 11 and 15-20. The reduced class files include a subset of the code and data of the original class files, e.g., "class_defs" and "data" illustrated in slide 11 and the "other data" illustrated in slide 15, and the recited shared table includes, e.g., one or more of the "string_ids constant pool," "type_ids constant pool," "proto_ids constant pool," "field_ids constant pool," and "method_ids constant pool." The Dalvik Presentation shows the original class files being combined into a .dex file (multi-class file) comprising the plurality of reduced class files and the shared constant pool (shared table):

The ’702 Patent	Infringed By
	 (Dalvik Presentation, slide 15)  (Dalvik Presentation, slide 11)

The ’702 Patent	Infringed By
	 (Dalvik Presentation, slide 20) In the Android source code, *see generally*: “Classes for translating Java classfiles into Dalvik classes. PACKAGES USED: • com.android.dx.cf.code • com.android.dx.cf.direct • com.android.dx.cf.iface • com.android.dx.dex.code • com.android.dx.dex.file • com.android.dx.rop.code • com.android.dx.rop.cst • com.android.dx.util” dalvik\dx\src\com\android\dx\dex\cf\package.html.

The '702 Patent	Infringed By
	See also: /** * Representation of an entire {@code .dex} (Dalvik EXecutable) * file, which itself consists of a set of Dalvik classes. */ public final class DexFile { /** {@code non-null;} word data section */ private final MixedItemSection wordData; dalvik\dx\src\com\android\dx\dex\file\DexFile.java. *See also:* dalvik/dx/src/com/android/dx/dex/file/DexFile.java dalvik/dx/src/com/android/dx/dex/file/TypeIdsSection.java dalvik/dx/src/com/android/dx/dex/file/TypeIdItem.java dalvik/dx/src/com/android/dx/cf/cst/ConstantPoolParser.java

The '702 Patent	Infringed By
5. The method of claim 1, wherein said step of determining a plurality of duplicated elements comprises:	*See* Claim 1, *supra*.
determining one or more constants shared between two or more class files.	The Android dx tool determines constants shared between two or more class files. This process is explained in the Dalvik Video at time 7:20-9:25 and Dalvik Presentation, slides 11-20. The Dalvik Presentation shows the elements of the class files identified for combining into a shared constant pool (shared tables) in the .dex file.