EXHIBIT 27

Exhibit N – U.S. Patent No. 5,481,721

Motorola directly and/or indirectly infringes at least claims 1 and 5 of the ’721 patent, either literally or through the doctrine of equivalents. Motorola’s infringing products include mobile devices such as smartphones and tablet computers, including but not limited to the: Atrix, Bravo, Cliq, Cliq XT, Cliq 2, Charm, Defy, Devour, BackFlip, Devour, Droid, Droid 2, Droid 2 Global, Droid X, Droid Pro, Flipout, Flipside, i1, Xoom, (collectively, “the ’721 Accused Products”).[1]

For the purposes of this analysis, Plaintiffs provide a description of the Android Binder framework, specifically, how the Android OS passes messages between objects in different processes. For claims 1 and 5 Plaintiffs further provide a specific example of the Android Window System Manager passing information about touch screen events to an object in another process. The Window System Manager example described herein is intended to be exemplary only, and not to limit the scope of Plaintiffs’ infringement allegations. In any instance in which a call is made through the Binder framework and the remote object determines that additional information is needed to execute the call, the Binder framework operates as described in this claim chart, with the exception of the identities of the calling and receiving objects, and of the methods called. Accordingly, any such calls constitute infringement. The AIDL files listed in **Exh. N-1** to this claim chart are representative of the different calls made through the Binder framework.

In addition to Motorola's direct infringement of the claims of the ’721 patent through its development, testing, manufacture and use of its devices, Motorola also indirectly infringes claims 1 and 5 of the patent. Manufacturers, retailers, distributors, end-users and others in the distribution channel of the ’721 Accused Products directly infringe these claims by using, selling, offering for sale, and/or importing these devices into the United States. Motorola contributes to and induces the infringement of asserted claims 1 and 5 through its promotion and provision of intentional marketing, sale and/or technical support of the ’721 Accused Products and associated specialized components in the United States, and through the intentional design, marketing, manufacture, sale, and/or technical support of the ’721 Accused Products abroad to induce direct infringement in the United States. Motorola supplies ’721 Accused Products and actively encourages the use, sale, offer for sale, and importation of the same in the United States through the promotion and provision of marketing literature and user guides, which induces and results in direct infringement. *See, e.g.*, Motorola Droid X User Guide (WI-Apple0034078-34145). Upon information and belief, Motorola has known or should have known that these actions would cause direct infringement of the ’721 patent and did so with specific intent to encourage direct infringement. Additionally, the ’721 Accused Products have no substantial non-infringing uses.

[1] Motorola has announced additional smartphones including XRT and Titanium which may also infringe the ’721 Patent. Apple reserves the right to supplement this analysis and this list of accused products as discovery into these newly announced products progresses.

These infringement contentions are preliminary and based only on publicly available information as to the ’721 Accused Products. Motorola has not yet provided discovery as to its accused products and in addition, Plaintiffs’ investigation of Motorola’s infringement is ongoing. Based on discovery and Plaintiffs’ continued investigations, Plaintiffs reserve the right to amend these contentions to identify additional bases for infringement and additional accused products, including products that Motorola may introduce in the future. Accordingly, Plaintiffs reserve its right to amend these contentions as discovery and its investigation proceeds. Also, these disclosures are made based on information ascertained to date, and Plaintiffs expressly reserve the right to modify or amend the disclosures contained herein based on the Court’s claim constructions or to reflect additional information that becomes available to Plaintiffs.

U.S. Patent No. 5,481,721	**Infringement Contentions**
1. A method for sending an object oriented programming language based message having dynamic binding from a first object in a first process to a second object in a second process, said method comprising the steps of:	The ’721 Accused Products implement a method for sending an object oriented programming language based message having dynamic binding from a first object in a first process to a second object in a second process. Each of the ’721 Accused Products implements the functionalities claimed in the ’721 patent and described herein in essentially the same manner. Applications on the ’721 Accused Products are written in the Java programming language, which is object oriented and supports object oriented messages. *See* **Exh. N-2** [Android Developers-What is Android?], **Exh. N-3** [Java Tutorial], **Exh. N-4** [Android Developers-Designing for Performance]. To pass messages between processes (*i.e.*, interprocess communication (IPC)), the ’721 Accused Products use a Binder framework. *See* **Exh. N-5** [Android Anatomy and Physiology Presentation] at pp. 10-20, 101, 102, 107, 108. Using the Binder framework, object oriented programming language based messages can then be sent, by way of a proxy object, from the first process to the second process and replied to. *Id.* at pp. 11-19. *See also* **Exh. N-6** [Android Developers-Android Interface Definition Language] (“AIDL (Android Interface Definition Language) is similar to other IDLs you might have worked with. It allows you to define the programming interface that both the client and service agree upon in order to communicate with each other using interprocess communication (IPC). On Android, one process cannot normally access the memory of another process. So to talk, they need to decompose their objects into primitives that the

U.S. Patent No. 5,481,721	Infringement Contentions
	operating system can understand, and marshall the objects across that boundary for you. The code to do that marshalling is tedious to write, so Android handles it for you with AIDL.”). *See* **Exh. N-7** [Android Developers-IBinder] (“The key IBinder API is transact() matched by Binder.onTransact(). These methods allow you to send a call to an IBinder object and receive a call coming in to a Binder object, respectively. This transaction API is synchronous, such that a call to transact() does not return until the target has returned from Binder.onTransact(); this is the expected behavior when calling an object that exists in the local process, and the underlying inter-process communication (IPC) mechanism ensures that these same semantics apply when going across processes.”). Messages passed from the first process to the second process are dynamically bound at runtime. Namely, calls made to and through the proxy objects are not bound to a particular method in the concrete proxy subclass until runtime. For instance, a data value called “cookie” provides information to the Binder functionality within the kernel regarding which object in the second process should receive the call made upon the proxy object. This value is set during runtime. *See* **Exh. N-16** [binder.h], **Exh. N-17** [binder.c]. • The ’721 Accused Products implement the claimed method through their use of the Binder framework to send object oriented programming language based messages between objects in different processes.[2] In the Android framework, application programs access Android services, such as graphics, UI event processing, audio, telephony, wireless communications, and the like, by way of “services” within the Android Architecture. *See* **Exh. N-5** [Android Anatomy and Physiology Presentation] at 61-75. Within the Android framework, these Android services run in different processes than user applications. *See id.* at 9. An Android application accesses these service, by way of the Binder framework. *See id.* at 9-20. • By way of the representative example, Android’s Window Manager Service

[2] The object oriented programming language based messages take the form of C++ or Java based messages.

U.S. Patent No. 5,481,721	Infringement Contentions
	receives, for example, events generated at the touch screen. It communicates these events to applications provided by the ’721 Accused Products. Because the applications are in different processes from the Window Manager Service, the Window Manager Service (*i.e.*, the first object in the first process) uses the Binder framework to communicate the events to objects in the applications (*i.e.*, a second process). *See* **Exh. N-5** [Android Anatomy and Physiology Presentation] at 9-20, 64 and **Exh. N-7** [Android Developers-IBinder]. At runtime, when an event is received at the Window Manager Service that is to be transmitted to another process (*i.e.*, a message), the Binder framework, at runtime, dynamically binds the message to the methods in the second object. *See* **Exh. N-7** [Android Developers-IBinder], **8** [WindowManagerService.java] (discloses the use of IWindow, which uses the Binder framework to communicate to other processes).
transmitting, using a first processing means, said object oriented programming language based message to a first proxy in said first process;	The ’721 Accused Products perform the step of transmitting, using a first processing means, an object oriented programming language based message to a first proxy in a first process. More specifically, the Android Interface Definition Language (AIDL) defines an interface between the first process and the second process. *See* **Exh. N-6** [Android Developers-Android Interface Definition Language] (“AIDL (Android Interface Definition Language) is similar to other IDLs you might have worked with. It allows you to define the programming interface that both the client and service agree upon in order to communicate with each other using interprocess communication (IPC). On Android, one process cannot normally access the memory of another process. So to talk, they need to decompose their objects into primitives that the operating system can understand, and marshall the objects across that boundary for you. The code to do that marshalling is tedious to write, so Android handles it for you with AIDL.”). The interface includes a proxy object at the first process for receiving the messages from the first process to the second process. *See* **Exh. N-7** [Android Developers-IBinder] (“Attempt to retrieve a local implementation of an interface for this Binder object. If null is returned, you will need to instantiate a proxy class to marshall calls through the

U.S. Patent No. 5,481,721	Infringement Contentions
	transact() method.”). Such proxy classes include classes having names of the form XXXX[3].Stub.Proxy, and may be manually written or auto-generated by the AIDL compiler from appropriate interface classes. Instances of these proxy classes are located in a first process, *e.g.*, the process from which the object oriented programming language based message is sent, and messages are sent to the proxy object using a first processing means, such as a processor. *See, e.g.*, **Exh. N-8** [WindowManagerService.java], **Exh. N-9** [IWindow.java], **Exh. N-10** [IWindow.aidl] for specific examples of proxy objects being called. • By way of the representative example, the process with Android’s Window Manager Service contains objects of the class IWindow.Stub.Proxy, which is generated by the AIDL compiler from the interface IWindow. *See* **Exh. N-8** [WindowManagerService.java], **Exh. N-9** [IWindow.java], **Exh. N-10** [IWindow.aidl]. These objects serve as proxies for objects within Android applications that handle events, such as user interface events dispatched from the Window Manager Service. *Id.*. The Android Window Manager Service transmits object oriented programming language based messages to these proxies by invoking their methods. *See* **Exh. N-6** [Android Developers-Android Interface Definition Language]. For example, to send a touchscreen user interface event to an object in an application that can respond to it, the Window Manager Service calls the dispatchPointer() method of the appropriate IWindow.Stub.Proxy object. *See* **Exh. N-8** [WindowManagerService.java]. The IWindow.Stub.Proxy object exists in a first process, that is, the process of the Window Manager Service that calls its methods. *See* **Exh. N-8** [WindowManagerService.java], **Exh. N-9** [IWindow.java], **Exh. N-10** [IWindow.aidl].
using said first proxy and said first processing means, encoding said object	The ’721 Accused Products perform the step of, using a first proxy, encoding an object oriented programming language based message into an operating system based message

[3] For the purposes of this chart, "XXXX" refers to any combination of one or more alphanumeric characters.

U.S. Patent No. 5,481,721	Infringement Contentions
oriented programming language based message into an operating system based message at run time;	at runtime. After the first process passes the message to the proxy at the first process, the proxy writes certain data about the method called and the arguments supplied into a Parcel data structure. *See* **Exh. N-7** [Android Developers-IBinder] ("The data sent through transact() is a Parcel, a generic buffer of data that also maintains some meta-data about its contents. The meta data is used to manage IBinder object references in the buffer, so that those references can be maintained as the buffer moves across processes. This mechanism ensures that when an IBinder is written into a Parcel and sent to another process, if that other process sends a reference to that same IBinder back to the original process, then the original process will receive the same IBinder object back. These semantics allow IBinder/Binder objects to be used as a unique identity (to serve as a token or for other purposes) that can be managed across processes."). The proxy object then calls the transact() method of an object of the class BinderProxy; this method invokes a series of other methods, including the transact() methods of objects of the classes BpBinder and IPCThreadState and the writeTransactionData() method of an IPCThreadState object. *See* **Exh. N-12** [Binder.java]; *see* also **Exh. N-13** [android_util_Binder.cpp], **Exh. N-14** [BpBinder.cpp], **Exh. N-15** [IPCThreadState.cpp]. This method writes the data from the Parcel object, along with other data such as data relating to the target object, the message size, etc., into a binder_transaction_data structure. *See, e.g.*, **Exh. N-15** [IPCThreadState.cpp] (writeTransactionData() method). The binder_transaction_data structure is an Android operating system message. *See, e.g.*, **Exh. N-16** [binder.h], **Exh. N-17** [binder.c] (referencing the binder_transaction_data structure). This series of calls occurs in the first process (*e.g.*, the process of the calling object), and utilizes the first processing means (*e.g.*, a processor). • By way of the representative example, when the Android Window Manager Service informs an application about a touch screen event, it sends the dispatchPointer() message to a proxy object of the class IWindow.Stub.Proxy, along with arguments describing the touch screen event, such as a multitouch event, that has occurred. *See* **Exh. N-8** [WindowManagerService.java], **Exh. N-**

U.S. Patent No. 5,481,721	Infringement Contentions
	9 [IWindow.java] (method dispatchPointer of the proxy). IWindow.Stub.Proxy's implementation of the dispatchPointer() method encodes information about this message into a Parcel object, and then invokes the transact() method on an object of the class BinderProxy. *See* **Exh. N-12** [Binder.java] (having class BinderProxy with method transact()); *see* also **Exh. N-13** [android_util_Binder.cpp], **Exh. N-14** [BpBinder.cpp], **Exh. N-15** [IPCThreadState.cpp]. Several additional transact() methods are called and a binder_transaction_data structure is create at runtime. *See, e.g.*, **Exh. N-15** [IPCThreadState.cpp] (writeTransactionData() method).
transmitting said operating system based message to said second process in said second processing means at run time;	The '721 Accused Products perform the step of transmitting operating system based messages from one process to a second process at runtime. After a binder_transaction_data message is created by an IPCThreadState object's writeTransactionData() method, this message is written into a C++ Parcel object called mOut, and then into the write_buffer member of a data structure of the type binder_write_read. *See, e.g.*, **Exh. N-15** [IPCThreadState.cpp], **Exh. 18** [Parcel.cpp], **Exh. N-19** [parcel.h]. This binder_write_read structure is then itself passed from the first process into Android's Linux kernel via, for example, an ioctl() system call. *Id.* The Android kernel contains an implementation of ioctl() specific to the Binder Driver which copies the data in the write_buffer from the address space of the first process into the Android Linux kernel. *See* **Exh. N-17** [binder.c]. This data is then incorporated into a kernel structure of the type binder_transaction, which is placed on a queue associated with the target (second) process at runtime for access by the second process. *Id.* An object in the second process of class IPCThreadState creates a new binder_write_read structure and then uses an ioctl() system call to retrieve the operating system based binder_transaction_data message from the Android Linux kernel. *See* **Exh. N-15** [IPCThreadState.cpp]. For example, the Binder Driver within the Android kernel retrieves the binder_transaction structure that the kernel placed on its work queue. *See* **Exh. N-17** [binder.c]. The Binder Driver then uses the data within this

U.S. Patent No. 5,481,721	Infringement Contentions
	binder_transaction structure to reconstruct the binder_transaction_data message that was constructed in the first process. *See* **Exh. N-17** [binder.c]. This message is then copied to the user space of the second process. *Id.* • By way of the representative example, after the message to a proxy object of the class IWindow.Stub.Proxy is received and written into the binder_transaction_data structure, the message is placed on a queue associated with the second process for access by ViewRoot.W, used by, and in, the ViewRoot object (the second object). *See* **Exh. N-17** [binder.c].
decoding, using a second process, said operating system based message into a language based message;	The ’721 Accused Products perform the step of decoding, in the second process, a received operating system message into a programming language based message. The binder_transaction_data message is received from the Binder Driver by an object of class IPCThreadState that has been instantiated in the second process. *See* **Exh. N-17** [binder.c]. This object extracts from the message a code that indicates which method on the target object should be called, and a data buffer in the form of a Parcel object that stores any arguments associated with that method call. *Id.* The code and corresponding Parcel are passed, using the execTransact() method of the IPCThreadState object, to the transact() method of an object of the class JavaBBinder, which in turn passes the data to the onTransact() method of that object. *See* **Exh. N-13** [android_util_Binder.cpp], **Exh. N-15** [IPCThreadState.cpp], **Exh. N-23** [Binder.cpp]. The JavaBBinder object in turn uses the execTransact() method of the Java Binder class to pass the method code and Parcel up to the onTransact() method of a stub class that is associated with the target object. *See* **Exh. N-13** [android_util_Binder.cpp]. For example, an Android object that can be called remotely has an associated stub object that communicates with a JavaBBinder object to complete the process of decoding binder_transaction_data messages received from another process into Java messages to be sent to the actual target object. These stub objects include classes generated by the AIDL compiler from the interface defined for the particular target object, and include objects instantiated from classes of the form XXXX.Stub. *See* **Exh. N-6** [Android

U.S. Patent No. 5,481,721	Infringement Contentions
	Developers-Android Interface Definition Language]. • By way of the representative example, when the Android Window Manager Service has called, for example, the method dispatchPointer() on an IWindow.Stub.Proxy object, the execTransact() method of the Java Binder class will call the onTransact() method of an object of class IWindow.Stub, which will use the method code to determine which method of the application's receiver object should be invoked and generate an object oriented Java message in the form of a call to that method. *See, e.g.*, **Exh. N-9** [IWindow.java], **Exh. N-13** [android_util_Binder.cpp]. In this case, the decoding occurs in the second process in which the application that is receiving user interface events dispatched by the Window Manager Service is running.
transmitting, using said second processing means, said object oriented programming language based message to said second object in said second process;	The '721 Accused Products perform the step of transmitting the object oriented programming language based message to the appropriate object in the second process. The onTransact() method of the appropriate stub object transmits, using a processor, the object oriented message to the second target object by calling one of its methods. *See, e.g.*, **Exh. N-9** [IWindow.java]. • By way of the representative example, once the message transmitted from the IWindow.Stub.Proxy object of the Android Window Manager Service has been decoded and passed to the onTransact() method of the IWindow.Stub object of the application provided by the '721 Accused Products, the stub object transmits the message to the intended object in the application (*i.e.*, the object with, for example, the dispatchPointer () method."). *See, e.g.*, **Exh. N-9** [IWindow.java].
executing said object oriented programming language based message by said second object in said second process.	The '721 Accused Products perform the step of executing the object oriented programming language message received by the object in the second process. Once the methods of the target Java objects are invoked by the corresponding stub

U.S. Patent No. 5,481,721	Infringement Contentions
	objects, they will be executed. *See, e.g.*, **Exh. N-9** [IWindow.java]. • By way of the representative example, once the message from the Android Window Manager Service is received by the intended object at the application, the dispatchPointer () method of the object, for example, will be executed. *See, e.g.*, **Exh. N-9** [IWindow.java].
5. The method of claim 1 wherein the step of executing said object oriented programming language based message further includes the steps of: said second object determining, using said second processing means, whether additional information is needed to execute said object oriented programming language based message;	The ’721 Accused Products include second objects that perform the step of determining, using a second processing means (*i.e.*, a processor), whether additional information is needed to execute the object oriented programming language based message. Each of the ’721 Accused Products implements the functionalities claimed in the ’721 patent and described herein in essentially the same manner. Applications provided with the ’721 Accused Products include certain objects that include the capability to determine if additional information is needed. *See, e.g.*, **Exh. N-20** [ViewRoot.java]. Accordingly, when a first object in a first process transmits an object oriented programming language based message to a second object in a second process, the second object, provided it has the capability, will determine if additional information is needed beyond that transmitted in the message. • By way of the representative example, an object of the class ViewRoot has the capability to determine whether a message of the dispatchPointer() method from the Android Window Manager Service (*i.e.*, the first object in the first process) requires additional information before the method can be executed with a local object. *See, e.g.*, **Exh. N-20** [ViewRoot.java].
said second object generating, using said second processing means, an object oriented programming language based query if it is determined that additional information is needed;	The ’721 Accused Products perform the step of generating, using a second processing means, an object oriented programming language based query if it is determined that additional information is needed. The ’721 Accused Products include objects that can be queried remotely, *e.g.*, that can be invoked by objects in other processes via the Binder mechanism when additional

U.S. Patent No. 5,481,721	Infringement Contentions
	information is needed. *See, e.g.*, **Exh. N-20** [ViewRoot.java]. • By way of the representative example, if the object of the class ViewRoot determines that additional information is required for the dispatchPointer() method, the getPendingPointerMove () method is invoked to generate an object oriented programming language based query. *See, e.g.*, **Exh. N-20** [ViewRoot.java].
encoding, using said second processing means, said object oriented programming language based query into an operating system based query at run time if it is determined that additional information is needed;	The ’721 Accused Products perform the step of encoding, using a second processing means, an object oriented programming based query into an operating system based query at runtime if it is determined that additional information is needed. As described in reference to claim 1, after the process passes the message to the proxy in the process, the proxy writes certain data about the method called and the arguments supplied into a Parcel data structure. *See* **Exh. N-7** [Android Developers-IBinder]. (“The data sent through transact() is a Parcel, a generic buffer of data that also maintains some meta-data about its contents. The meta data is used to manage IBinder object references in the buffer, so that those references can be maintained as the buffer moves across processes. This mechanism ensures that when an IBinder is written into a Parcel and sent to another process, if that other process sends a reference to that same IBinder back to the original process, then the original process will receive the same IBinder object back. These semantics allow IBinder/Binder objects to be used as a unique identity (to serve as a token or for other purposes) that can be managed across processes.”). The proxy object then calls the transact() method of an object of the class BinderProxy; this method invokes a series of other methods, including the transact() methods of objects of the classes BpBinder and IPCThreadState and the writeTransactionData() method of an IPCThreadState object. *See* **Exh. N-12** [Binder.java]; *see* also **Exh. N-13** [android_util_Binder.cpp], **Exh. N-14** [BpBinder.cpp], **Exh. N-15** [IPCThreadState.cpp]. This method writes the data from the Parcel object, along with other data such as data relating to the target object, the message size, etc., into a binder_transaction_data structure. *See, e.g.*, **Exh. N-15** [IPCThreadState.cpp]. The binder transaction data

U.S. Patent No. 5,481,721	Infringement Contentions
	structure is an Android operating system message. *See*, *e.g.*, **Exh. N-16** [binder.h], **Exh. N-17** [binder.c]. This series of calls occurs in the second process (*e.g.*, the process of the calling object), and utilizes the second processing means (*e.g.*, a processor). • By way of the representative example, if an Android Application sends the getPendingPointerMove() query, the IWindowSession.Stub.Proxy object writes certain data about the query into a Parcel object, and then invokes the transact() method of an object of the class BinderProxy. *See* **Exh. N-7** [Android Developers-IBinder]. *See* **Exh. N-21** [IWindowSession.java]. *See also* **Exh. N-13** [android_util_Binder.cpp], **Exh. N-14** [BpBinder.cpp], **Exh. N-15** [IPCThreadState.cpp]. This method writes the data from the Parcel object, along with other data such as data relating to the target object, the message size, etc., into a binder_transaction_data structure. *See, e.g.*, **Exh. N-15** [IPCThreadState.cpp]. The binder_transaction_data structure is an Android operating system message. *See, e.g.*, **Exh. N-16** [binder.h], **Exh. N-17** [binder.c].
transmitting said operating system based query to said first process at run time, using said second processing means if it is determined that additional information is needed;	The ’721 Accused Products perform the step of transmitting operating system based queries to a first process at runtime using a second processing means (*i.e.*, a processor), if it is determined that additional information is needed. As described in reference to claim 1, after a binder_transaction_data message is created by an IPCThreadState object’s writeTransactionData() method, this message is written into a C++ Parcel object called mOut, and then into the write_buffer member of a data structure of the type binder_write_read. *See*, *e.g.*, **Exh. N-15** [IPCThreadState.cpp], **Exh. 18** [Parcel.cpp], **Exh. N-19** [parcel.h]. This binder_write_read structure is then itself passed from the second process into Android’s Linux kernel via, for example, an ioctl() system call. *See* **Exh. N-15** [IPCThreadState.cpp] (ioctl() call) The Android kernel contains an implementation of ioctl() specific to the Binder Driver which copies the data in the write_buffer from the address space of the second process into the Android Linux kernel. *See* **Exh. N-17** [binder.c] (binder_octl()). This data is then incorporated into a kernel structure of the type binder transaction, which is placed on a queue

U.S. Patent No. 5,481,721	Infringement Contentions
	associated with the target (first) process at runtime for access by the first process. *Id.* An object in the first process of class IPCThreadState creates a new binder_write_read structure and then uses an ioctl() system call to retrieve the operating system based binder_transaction_data message from the Android Linux kernel. *See* **Exh. N-15** [IPCThreadState.cpp]. For example, the Binder Driver within the Android kernel retrieves the binder_transaction structure that the kernel placed on its work queue. *See* **Exh. N-17** [binder.c]. The Binder Driver then uses the data within this binder_transaction structure to reconstruct the binder_transaction_data message that was constructed in the second process. *See* **Exh. N-17** [binder.c] (binder_thread_read()). This message is then copied to the user space of the first process. *Id.* • By way of the representative example, the message transmitted by the IWindowSession.Stub.Proxy class is placed in the queue of a target object associated with the first process. *See* **Exh. N-15** [IPCThreadState.cpp], **Exh. 18** [Parcel.cpp], **Exh. N-19** [parcel.h].
decoding, using said first processing means, said operating system based query into an object oriented programming language based query at run time if it is determined that additional information is needed;	The ’721 Accused Products perform the step of decoding, using a first processing means, an operating system based query into an object oriented programming language based query at runtime if it is determined that additional information is needed. As described in reference to claim 1, the binder_transaction_data message is received from the Binder Driver by an object of class IPCThreadState that has been instantiated in the first process. *See* **Exh. N-17** [binder.c]. This object extracts from the message a code that indicates which method on the target object should be called, and a data buffer in the form of a Parcel object that stores any arguments associated with that method call. *Id.* The code and corresponding Parcel are passed, using the execTransact() method of the IPCThreadState object, to the transact() method of an object of the class JavaBBinder, which in turn passes the data to the onTransact() method of that object. *See* **Exh. N-13** [android_util_Binder.cpp], **Exh. N-15** [IPCThreadState.cpp], **Exh. N-23** [Binder.cpp]. The JavaBBinder object in turn uses the execTransact() method of the Java Binder class

U.S. Patent No. 5,481,721	Infringement Contentions
	to pass the method code and Parcel up to the onTransact() method of a stub class that is associated with the target object. *See* **Exh. N-13** [android_util_Binder.cpp]. For example, an Android object that can be called remotely has an associated stub object that communicates with a JavaBBinder object to complete the process of decoding binder_transaction_data messages received from another process into Java messages to be sent to the actual target object. These stub objects include classes generated by the AIDL compiler from the interface defined for the particular target object, and include objects instantiated from classes of the form XXXX.Stub. *See* **Exh. N-6** [Android Developers-Android Interface Definition Language]. • By way of example, if it is determined that additional information is needed and the second object has issued the query getPendingPointerMove() on a Session member of the class WindowManagerService, the execTransact() method of the Java Binder class will call the onTransact() method of an object of class IWindowSession.Stub, which will use the method code to determine which method of the Session member should be called, and generate an object oriented Java query in the form of a call to that method. *See* **Exh. N-8** [WindowManagerService.java], **Exh. N-13** [android_util_Binder.cpp], **Exh. N-14** [BpBinder.cpp], **Exh. N-15** [IPCThreadState.cpp], **Exh. N-20** [ViewRoot.java], **Exh. N-21** [IWindowSession.java], **Exh. N-23** [Binder.cpp]. In this case, the decoding occurs in the first process in which the WindowManagerService and Session member exist.
transmitting, using said first processing means, said object oriented programming language based query to said first object if it is determined that additional information is needed.	The ’721 Accused Products perform the step of transmitting, using a first processing means, the object oriented programming language based query to a first object if it is determined that additional information is needed. The onTransact() method of the appropriate stub object transmits the object oriented message to the target object by calling one of its methods. *See, e.g.*, **Exh. N-8** [WindowManagerService.java], **Exh. N-9** [IWindow.java], **Exh. N-13** [android_util_Binder.cpp], **Exh. N-14** [BpBinder.cpp], **Exh. N-15** [IPCThreadState.cpp], **Exh. N-20** [ViewRoot.java], **Exh. N-21** [IWindowSession.java], **Exh. N-23**

U.S. Patent No. 5,481,721	Infringement Contentions
	[Binder.cpp]. • By way of the representative example, an object extending class IWindowSession.Stub invokes the appropriate method in the target Session member of the WindowManagerService object (*e.g.*, in the example discussed above, the getPendingPointerMove() method). *See, e.g.*, **Exh. N-8** [WindowManagerService.java], **Exh. N-13** [android_util_Binder.cpp], **Exh. N-14** [BpBinder.cpp], **Exh. N-15** [IPCThreadState.cpp], **Exh. N-20** [ViewRoot.java], **Exh. N-21** [IWindowSession.java], **Exh. N-23** [Binder.cpp]. The query, in the form of a method invocation, is transmitted to the first object using the first processing means (*i.e.*, a processor).