

# AIMTools

## Android

# QuickStart Guide



# Contents

## Introduction

- Key Features

- Technical Overview

## Getting Started

- A Quick Demo – Running Your First AIMTools Sample

## Architecture

- AIMTools Architecture: Overview

  - Libraries Overview

  - API

  - Pegasus Function

  - API (Operations)

  - API (PIC\_PARM)

- AIMTools Libraries: Dispatcher

  - Opcodes

## Source Code & Sample Applications

- Source Code: A Quick Demo Revisited

- Sample Applications: Timing Tests

- Other Sample Applications

- Incorporating AIMTools Into Your App

## Additional Information

**AIM**Tools

# Introduction

# Key Features

- JPEG 2000 and Lossless JPEG viewing
  - 8 bit through 16 bit grayscale decompression support for Lossy and Lossless JPEG and JPEG 2000
  - Window leveling
  - Patent-pending quality and speed enhancements provide for a more responsive and better quality viewing application
- JPEG viewing
  - Derived from a highly optimized code base used worldwide by some of the largest medical imaging companies
  - Support for RGB and CMYK JPEG viewing
- Tiff viewing
- High-speed image resize from RAW decompressed image data

# Technical Overview

- C language API provides total control over image processing functions.
- API accessible in C/C++ or Java via the Java Native Interface(JNI).
- Thread-safety allows multi-threaded client applications.
- Image data input/output provided by buffers allows for maximum flexibility.
- Available on Google Android.

**AIM**Tools

# Getting Started

# A Quick Demo

## Running Your First AIMTools Sample

- If you have not already done so, download the Software Development Kit (SDK) from [here](#), create a directory, and unzip the SDK into the directory.
- For an overview of the Development Kit contents, please see the readme-android.txt file located at the top level of the kit.
- The sample applications are Apps built with Ant using the Android NDK and SDK. The code for the quickstartsample can be found in the samples/quickstartsample/quickstartd folder.
- Build and run the quickstartsample App. The App decompresses a specified jpeg file into a bitmap file. It reads input files and writes output files from/to the App's document directory. Once the App is installed in the emulator or your device, you can transfer jpeg files to the document folder on the device with the ADB (Android Debug Bridge).

# A Quick Demo

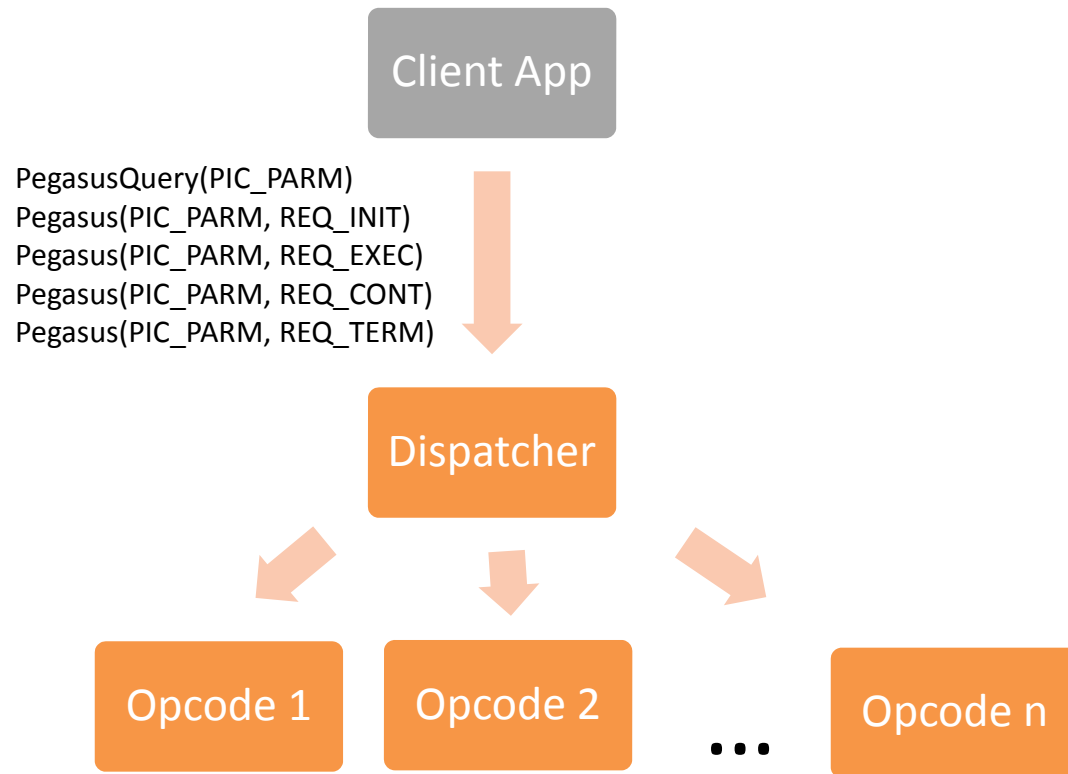
## Running Your First AIMTools Sample

- The location of the document directory is displayed in the initial screen of the App and can be toggled between a directory in the internal storage and external storage (if available).
- Select a file to decompress and if desired, change the output file name on the next screen. Then press the “Save” button to decompress the file.
- This will decompress the input file to the specified output file. You have just run your first AIMTools sample application!
- The output file can be retrieved from the document directory.
- We’ll revisit the quickstartsample at the end!

AIMTools

# Architecture

# AIMTools Architecture: Overview



Note: The items depicted on this diagram are described on the following pages.

# AIMTools Architecture: Libraries Overview

- If you are familiar with PICTools, you will note that AIMTools uses the same Architecture. PICTools is an imaging toolkit targeted to desktop and server platforms running Operating Systems such as Windows, Linux, Solaris, AIX, and OS X. (For more information of PICTools, please look [here](#).)
- AIMTools is implemented in a set of libraries:
  - Dispatcher
    - exposes a common API to clients that is used across all opcodes
    - prepares the opcode libraries for execution
    - provides common functions across all opcodes
  - Opcodes
    - AIMTools functions are grouped into cohesive, modular libraries called opcodes
    - each opcode implements a specific technology. Examples of AIMTools opcodes include:
      - sequential JPEG decompression opcode
      - Lossless JPEG decompression opcode
      - JPEG 2000 decompression opcode
      - Zoom opcode
    - modular opcode architecture means that client applications only need to deploy the opcodes containing the functionality required by the application

# AIMTools Architecture: API

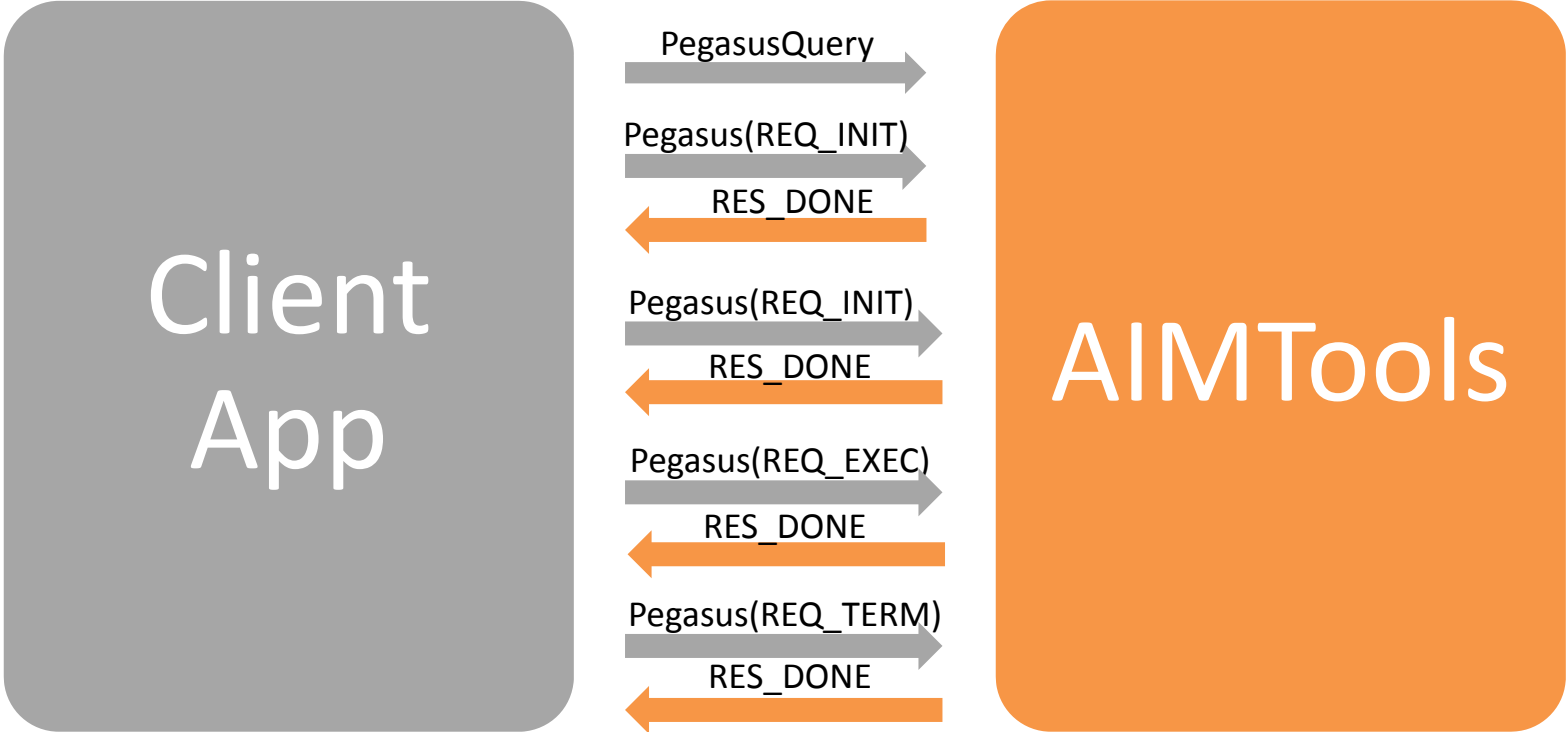
- The AIMTools API is embodied in a set of functions, operations performed by those functions, and data structures passed to/from the functions.
- Complete details on the API can be found in the *AIMTools Programmer's Reference*.
- The most important and most used functions are:
  - PegasusQuery(PIC\_PARM)
    - examines an image to determine image properties such as image format, size, and dimensions
    - the PIC\_PARM structure (discussed below) allows the client to specify the image properties to be returned and then contains the requested properties upon return from the PegasusQuery function
  - Pegasus(PIC\_PARM, Request)
    - the single entry point into the image manipulation functions provided by AIMTools
    - the PIC\_PARM structure specifies the input parameters for the operation and contains the results of executing the operation
    - the Request specifies the operation to be performed

# AIMTools Architecture: Pegasus Function

- The Operations that can be requested by calling the Pegasus function are:
  - REQ\_INIT: begins initialization of the operation. Any required internal memory allocation or other opcode initialization takes place.
  - REQ\_EXEC: begins execution of the operation.
  - REQ\_CONT: used in certain modes of operation and causes resumption of the execution phase in case it was paused.
  - REQ\_TERM: terminates the operation. May be used to terminate an operation at any time, but normally used at the conclusion of REQ\_EXEC processing.
- A response is returned from calls to the Pegasus function. Numerous responses are possible, as discussed in the *AIMTools Programmer's Reference*. Some possible responses are:
  - RES\_DONE: indicates the operation was completed without error.
  - RES\_ERR: an error occurred. Examine the Status field in the PIC\_PARM structure for the specific error code.

# AIMTools Architecture: API (Operations)

## Basic Sequence of Operations



# AIMTools Architecture: API (PIC\_PARM)

- The main data structure used to pass data to and from AIMTools is the PIC\_PARM structure.
  - contains all input parameters used by the client to control opcode operations
    - contains a section of generic parameters, common to all opcodes
    - contains a section of opcode specific parameters
  - contains all information returned from the opcode
  - contains input and output buffers for passing image data
    - input buffer is known as the get queue
    - output buffer is known as the put queue

# AIMTools Libraries: Dispatcher

- On most systems, the dispatcher is available in the form of either a static or dynamic library.
  - The dynamic libraries are found in the SDK in the bin directory
  - The static libraries are found in the SDK in the lib directory
  - The file name for the dispatcher dynamic library is libpicd20.so
  - The file name for the dispatcher static library is libpicd20.a
  - Dispatcher libraries are built for both ARMV5 and ARMV7A. The linker will pull in the code needed for all platforms being built, but the installer will include only the code required for the target on which your App is being installed.

# AIMTools Libraries: Dispatcher

- A debug version of the dispatcher is available which allows generation of a trace/debug log.
  - This is helpful for problem detection and identification.
- The debug dispatcher file name is the same as the non-debug version, but with the addition of the letter 'd' prior to the file extension (libpicd20d.a or libpicd20d.so).
- Details on how to use the debug dispatcher to generate a debug log can be found in the *PICTools and AIMTools Programmer's Guide*.
- The debug log will be written to the directory specified in the pdebug.ini configuration file, which should be placed in a directory on the LD\_LIBRARY\_PATH or in /mnt/sdcard/pegasus (if your app can access that directory).

# AIMTools Libraries: Opcodes

- Opcodes are available on Android as:
  - dynamic libraries.
  - A separate library exists for each opcode, named with a unique number.
  - A complete list of opcodes and opcode numbers can be found in the Overview section of the *AIMTools Programmer's Reference*.
  - The opcode library files are in the SDK bin directory.
  - The file names for the opcode library files are `picd??20.so`, where '??' represents the opcode number.
  - The opcode libraries are built for both ARMV5 and ARMV7A. The linker will pull in the code needed for all platforms being built, but the installer will include only the code required for the target on which your App is being installed.

# AIMTools Libraries: Opcodes

- The installer will unpack opcode dynamic libraries (\*.so files) into your app's native library directory by the installer. In general, this will be the lib directory within your application's directory, often similar to
  - /data/data/com.example.myapp/lib
- The native library directory can be determine programmatically from the nativeLibraryDir member of your application's ApplicationInfo:
  - MyApp.this.getApplicationInfo().nativeLibraryDir
- The dispatcher locates opcode libraries in one of two ways:
  - The app calls PegasusLoad () to load the library before invoking Pegasus().
  - The app adds the directory containing the libraries to the LD\_LIBRARY\_PATH environment variable, which the dispatcher will search.

**AIM**Tools

# Source Code & Sample Applications

# Source Code: quickstartsample Revisited

- The source code for the quickstartsample and all other sample programs is located in the SDK in the samples directory.
- All files and directories for building with Ant and the Android NDK and SDK are located with the source code.
- Details on the quickstartsample, along with further AIMTools programming information, can be found in the *PICTools and AIMTools Programmer's Guide*.
- Interesting include files, located in the SDK include directory are:
  - pic.h: defines the PIC\_PARM structure and all other API declarations
  - errors.h: defines the error codes which may be returned in PIC\_PARM.Status

# Sample Applications: Timing Tests

- Sample applications are include that can provide data on the amount of time required to perform AIMTools operation.
- However, running in Evaluation Mode can skew results.
  - Until AIMTools is licensed, sample applications run in Evaluation Mode.
  - In Evaluation Mode a delay is introduced before processing and a pop-up dialog may be present.
- Some sample applications provide a means of excluding the delay in timing tests:
  - An option is provided for running an operation multiple times and specifying the iteration to begin timing. Only the first iteration includes the delay, so by starting timing on the 2<sup>nd</sup> iteration, the delay is not included in timing calculations.
  - Many samples include the `-x[n1[:n2]]` option for running multiple iterations, where n1 specifies the number of iterations to run the operation, and n2 specifies the iteration to begin timing. Average timing over runs n2 through n1 is displayed.

# Other Sample Applications

- Sample Apps are provided for all AIMTools operations present in the SDK.
- In order to run any sample App, it must first be built using Ant and the Android NDK and SDK.
- The sample Apps provide a text entry field in which the name of the test app and the test parameters are entered. (If you are familiar with the PICTools test programs on other platforms, this command is exactly the same as the command line test program command).
- Each sample can be executed without any parameters to display a usage message which briefly describes all available parameters.
- The parameters correspond to the options provided by each AIMTools operation. Details on each operation and the associated options can be found in the *AIMTools Programmer's Reference*, located in the SDK doc directory.

# Incorporating AIMTools Into Your App

- Your project must refer to the AIMTools include files.
  - In the SDK, these files are in the include folder.
  - In your jni/Android.mk file, add the include folder directory to the LOCAL\_C\_INCLUDES definition.
- Your project must link with the AIMTools libraries.
  - In the SDK, these files are in the bin and/or lib folders.
  - In your build environment (shell), add the library directory to the NDK\_MODULE\_PATH environment variable, which must also be exported.
  - In your jni/Android.mk file, import all static and dynamic libraries you will use. For example:
    - \$(call import-module, picds)
  - In your jni/Android.mk file, add all static libraries to your LOCAL\_STATIC\_LIBRARIES definition. For example:
    - LOCAL\_STATIC\_LIBRARIES += picds
  - In your jni/Application.mk file, include all dynamic libraries in your APP\_MODULES definition. For example:
    - APP\_MODULES = myapp picd64

# Incorporating AIMTools Into Your App

- Using the F2DPLUS (libpicd8520.so) opcode in your app:
  - F2DPLUS can decompress TIFF files with embedded JPEG images. In order to use this feature, you must link your app with both the F2DPLUS opcode and the SEP2DPLUS (libpicd8520.so) opcode.
  - Since F2DPLUS may invoke the SEP2DPLUS opcode, it links dynamically with the dispatcher library; consequently, you must link your app with the shared object version of dispatcher library (libpicd20.so) and include it in the LOCAL\_SHARED\_LIBRARIES definition in your jni/Android.mk file:
    - LOCAL\_SHARED\_LIBRARIES += picd
  - For an example of an App that uses F2DPLUS, see the f2dtestd sample project.

# Additional Information

- Please see the following documents in the SDK doc directory:
  - *PICTools and AIMTools Programmer's Guide*
  - *AIMTools Programmer's Reference*
- We welcome and encourage your questions and comments at anytime during your AIMTools use, from evaluation through deployment of your application.  
Contact us at [support@accusoft.com](mailto:support@accusoft.com).