PICTools

QuickStart Guide

Contents

Introduction
- Key Features
- Technical Overview

Getting Started
- A Quick Demo – Running Your First PICTools Sample

Architecture
- PICTools Architecture: Overview
  - Libraries Overview
  - API
  - Pegasus Function
  - API (Operations)
  - API (PIC_PARM)
- PICTools Libraries: Dispatcher
  - Opcodes

Source Code & Sample Applications
- Source Code: A Quick Demo Revisited
- Additional Illustrative Samples
- Sample Applications: Timing Tests
- Other Sample Applications

Additional Information

PICTools

Introduction

Key Features

- Fastest image compression and decompression available.
- Support for a wide variety of file formats and compression technologies.
- Patented advanced JPEG editing within compressed JPEG space which avoids quality loss due to decompression/recompression associated with other JPEG editing techniques.
- Image editing and automatic enhancement, including auto-red eye correction, auto-contrast correction, and much more.
- Document image processing including image cleanup, auto-binarization, and much more.

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.
- Both static and dynamic libraries available.
- Available on multiple platforms including Win32/64, Linux32/64, Solaris Sparc32/64, Solaris Intel32/64, AIX32/64, Apple OS X 32/64.

PICTools

Getting Started

A Quick Demo

Running Your First PICTools Sample

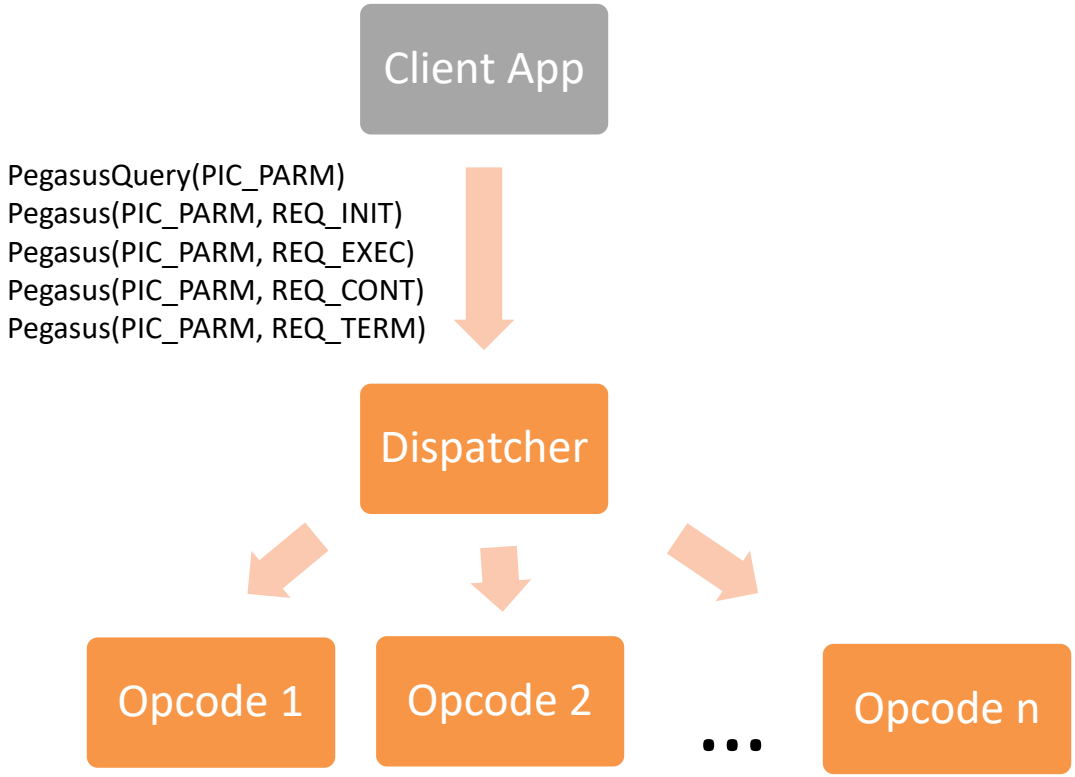
- If you have not already done so, download the appropriate SDK from here, create a directory and unzip the Software Development Kit (SDK) into the directory.
- For an overview of the Development Kit contents, please see the readme.txt file located at the top level of the kit.
- The sample applications are command-line executables, so in a terminal session, navigate to the bin directory.
- On Unix-based operating systems, you may need to set the SSMPATH environment variable to the bin directory path.
- Execute the quickstartsample without any parameters. This will display the usage message.
- Now, execute quickstartsample specifying an input jpeg file and an output bitmap file name. This will decompress the input file to the specified output file. You have just run your first PICTools sample application!

➢ We'll revisit the quickstartsample at the end!

PICTools

Architecture

PICTools Architecture: Overview



PICTools Architecture: Libraries Overview

- PICTools is implemented in a set of libraries:
  - Dispatcher
    - exposes a common API to clients that is used across all opcodes
    - loads the opcode libraries for execution
    - provides common functions across all opcodes
  - Opcodes
    - PICTools functions are grouped into cohesive, modular libraries called opcodes
    - each opcode implements a specific technology, e.g.
      - sequential JPEG compression opcode
      - binarize opcode
      - JPEG 2000 expand opcode
      - document cleanup opcode
    - modular opcode architecture means that client applications only need to deploy the opcodes containing the functionality required by the application

PICTools Architecture: API

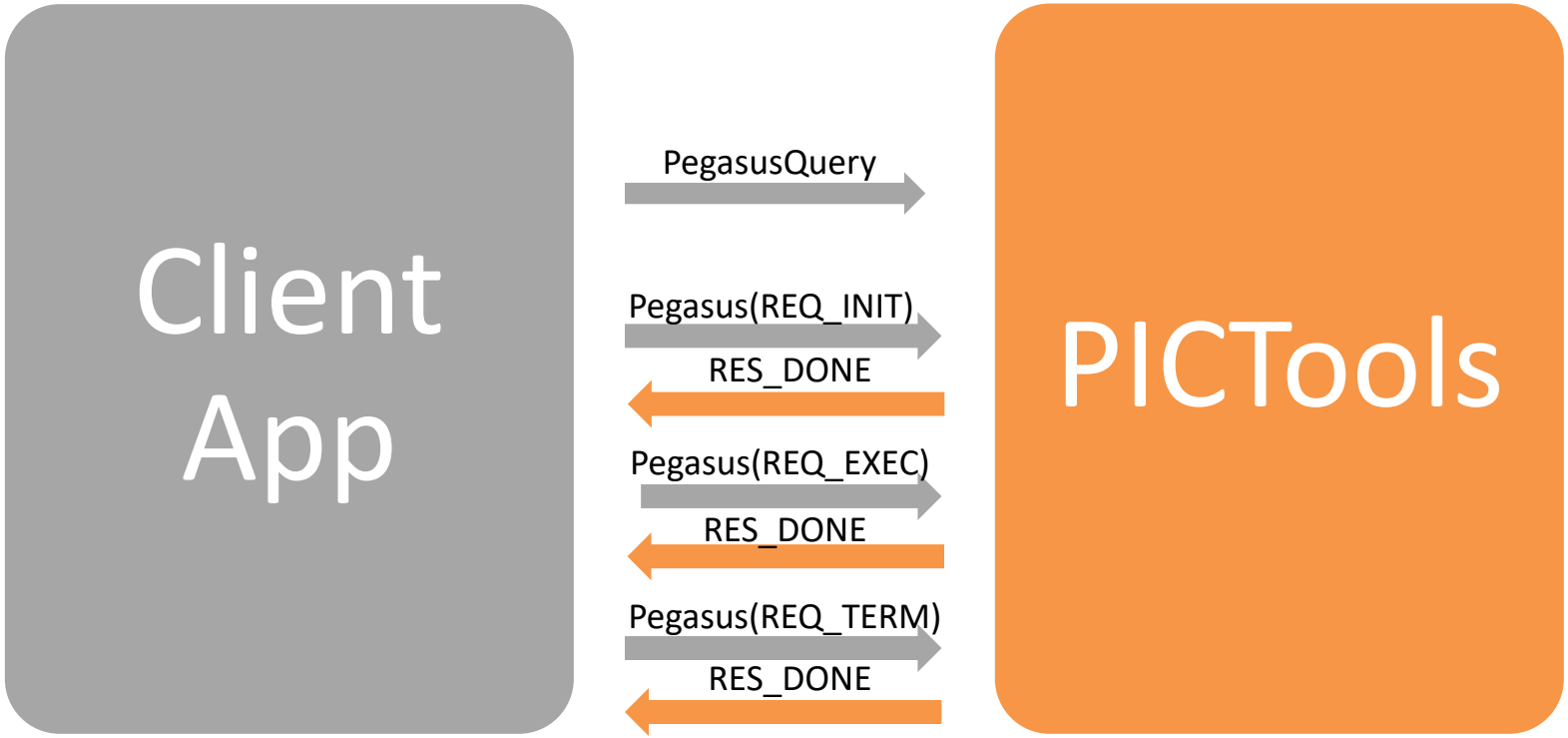
- The PICTools API is embodied in a set of functions, operations performed by those functions, and data structures passed to/from the functions.
- Complete details on the API can be found in the *PICTools Programmer's Reference.*
- The most important and most used functions are:
  - PegasusQuery(PIC_PARM)
    - 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 PICTools
    - 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

PICTools 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 *PICTools 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.

PICTools Architecture: API (Operations)

Basic Sequence of Operations



PICTools Architecture: API (PIC_PARM)

- The main data structure used to pass data to and from PICTools 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

PICTools 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 names for the libraries are as follows:

	Dynamic Library	Static Library
Win32	picn20.dll	picn20m.dll
Win64	picx20.dll	picx20m.dll
Linux32	libpicl20.so	libpicl20.a
Linux64	libpiclx20.so	libpiclx20.a
Solaris-Sparc32	libpicu20.so	libpicu20.a
Solaris-Sparc64	libpicux20.so	libpicux20.a
Solaris-Intel32	libpics20.so	libpics20.a
Solaris-Intel64	libpicsx20.so	libpicsx20.a
Aix32	libpica20.a	n/a
Aix64	libpicax20.a	n/a
OS X 32	libpicmu20.dylib	n/a
OS X 64	libpicmux20.dylib	n/a

PICTools 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 (for example, picn20d.dll).
- Details on how to use the debug dispatcher to generate a debug log can be found in Chapter 11 of the *PICTools and AIMTools Programmer’s Guide*.

PICTools Libraries: Opcodes

- Opcodes are available as:
  - dynamic libraries.
  - SSM files: these are compressed versions of the opcodes that are smaller, load faster, and can be embedded as resources in a client application (see the *PICTools Programmer's Reference* section on PegasusLoadFromRes for details on embedding SSM files).
  - Both formats are not available on all platforms.
  - A separate file 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 *PICTools Programmer's Reference.*
  - The opcode files are in the SDK bin directory.

PICTools Libraries: Opcodes

- The file names for the opcode files are as follows ('??' in the names below represents the opcode number):

	Dynamic Library	SSM
Win32	picn??20.dll	picn??20.ssm
Win64	picx??20.dll	picx??20.ssm
Linux32	n/a	picn??20.ssm
Linux64	n/a	picx??20.ssm
Solaris-Sparc32	picu??20.so	n/a
Solaris-Sparc64	picux??20.so	n/a
Solaris-Intel32	n/a	picn??20.ssm
Solaris-Intel64	n/a	picx??20.ssm
Aix32	pica??20.so	n/a
Aix64	picax??20.so	n/a
OS X 32	picm??20.piclib	picn??20.ssm
OS X 64	n/a	picx??20.ssm

PICTools

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.
- A Visual Studio solution and Linux makefiles are located with the source code.
- Details on the quickstartsample, along with further PICTools programming information, can be found in the *PICTools 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

Additional Illustrative Samples (aka quickstarts)

- Within the samples directory, you will find the quickstarts sub-directory.
- Contained here are various illustrative sample applicatons which can be expanded upon.
- The source code for these samples is split into two sections:
  - One provides the user-interface and image file handling
  - The other provides the interface to PICTools.
- These samples all make use of the “Helper” functions:
  - These functions are defined in helper.h and implemented in helper.c
  - The functions provide low-level functionality which is applicable to many applications
  - They may be used to simplify application code.

Additional Illustrative Samples

- The “Helper” functions provide:
  - The interface structure used between the PICTools interface function and the higher level functions.
  - Functions for initializing the PicParm structure and the queues used for passing image data.
  - Parsing/Writing of image headers.
  - Execution of the PegasusQuery operation
  - Other utility functions.
- Each illustrative sample can be executed without any parameters to display a usage message.

Sample Applications: Timing Tests

- Most sample applications can provide data on amount of time required to perform PICTools operation.
- However, running in Evaluation Mode can skew results.
  - Until PICTools 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.
- Many 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 2nd 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

- Command-line sample applications are provided for all PICTools operations present in the SDK.
- The executables for all sample applications are in the SDK bin directory.
- 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 PICTools operation. Details on each operation and the associated options can be found in the *PICTools Programmer's Reference*, located in the SDK doc directory.
- On Windows, a GUI sample application is available: Apolloxxx.exe (where xxx = Document, Medical, or Photo).
  - Demonstrates various compression/decompression technologies

Additional Information

- Please see the following documents in the SDK doc directory:
  - *PICTools Programmer's Guide*
  - *Pictools and AIMTools Programmer's Reference*
  - On non-Windows platforms, *Deploying PICTools applications on Linux, Solars, AIX, and OS X*
- We welcome and encourage your questions and comments at anytime during your PICTools use, from evaluation through deployment of your application.
  Contact us at support@accusoft.com.