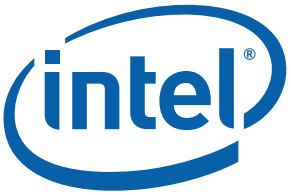

Intel® Integrated Performance Primitives for the Windows* OS on the IA-32 Architecture

User's Guide

November 2007

Document Number: 318254-001US

World Wide Web: http://developer.intel.com

Version	Version Information	Date
-001	Original issue of Intel® Integrated Performance Primitives (Intel® IPP) for Windows, IA-32 Architecture User's Guide.	September 2007
-002	V.5.3 Product update	November 2007

Contents

Tables

Examples

Overview 1

Intel® Integrated Performance Primitives (Intel® IPP) is a software library which provides a broad range of functionality including general signal, image, speech, graphics, data compression, cryptography, text strings and audio processing, vector manipulation and matrix math, as well as more sophisticated primitives for construction of audio, video and speech codecs such as MP3 (MPEG-1 Audio, Layer 3), MPEG-4, H.264, H.263, JPEG, JPEG2000, GSM-AMR* and G.723, plus computer vision.

By supporting a variety of data types and layouts for each function and minimizing the number of data structures used the Intel IPP library delivers a rich set of options for developers to choose from while designing and optimizing an application. A variety of data types and layouts are supported for each function. Intel IPP software minimizes data structures to give the developer the greatest flexibility for building optimized applications, higher level software components, and library functions.

Technical Support

Intel provides a support web site, which contains a rich repository of self help information, including getting started tips, known product issues, product errata, license information, user forums, and more. Visit the Intel® IPP support website at http://www.intel.com/software/products/support/ipp.

About This Document

The Intel® IPP User's Guide presents information the user may need to know how to make the best of Intel® IPP software.

Intel® IPP software is delivered in 4 separate packages: for Intel® Pentium® Processor Family (the IA-32 architecture), Intel processors with the the Intel® 64 architecture, Intel® Itanium Processor Family (the IA-64 architecture), and IXP4XX Product Line.

This document focuses on the *usage* information needed to call Intel IPP routines from user's applications running on the Windows* OS for the IA-32 architecture. Windows usage of Intel IPP has its particular features described in this guide along with those that do not depend upon a particular OS and architecture.

After installation, you may find this document in the `\doc` directory.

Purpose

This *Intel® IPP User's Guide* is intended to assist in mastering the usage of the Intel IPP. In particular, it

- Helps you start using the library by describing the steps you need to perform after the installation of the product,
- Shows you how to configure the library and your development environment to use the library,
- Acquaints you with the library structure,
- Explains in detail how to select the linking model that suits you best, link your application to the library and provides simple usage scenarios,
- Describes various details of how to code, compile, and run your application with Intel IPP.

Audience

The guide is intended for Windows programmers whose software development experience may vary from beginner to advanced.

Document Organization

The document contains the following chapters and appendices:

Chapter 1	Overview describes the document's purpose and organization and explains notational conventions.
Chapter 2	Getting Started with Intel® IPP describes necessary steps and gives basic information needed to start using Intel IPP after its installation.
Chapter 3	Configuring Your Development Environment explains how to configure Intel IPP and your development environment for the use with the library.

Chapter 4	Intel® IPP Structure discusses the structure of the Intel IPP directory after installation as well as the library versions.
Chapter 5	Linking Your Application with Intel® IPP compares linking models, helps selecting linking model for particular purpose, describes the general link line syntax to be used for linking with Intel IPP libraries; discusses how to build custom dynamic libraries.
Chapter 6	Managing Performance and Memory discusses Intel IPP threading, aligning memory, thresholding denormal data before Finite Impulse Response (FIR) filtering, reusing buffers, using FFT for algorithmic optimization (where appropriate); gives information how to accomplish Intel IPP functions performance tests by using Intel® IPP Performance Test Tool.
Chapter 7	Language-Specific Usage Options discusses some special aspects of using Intel® IPP with different programming languages and the Windows* OS development environments.
Appendix A	Performance Test Tool Command Line Options gives brief descriptions of possible performance test tool command line options.

The document also includes an Index.

Notational Conventions

The document employs the following font conventions and symbols:

Table 1-1 Notational conventions

Italic	*Italic* is used for emphasis and also indicates document names in body text, for example: see *Intel IPP Reference Manual*
`Monospace lowercase`	Indicates filenames, directory names and pathnames, for example: `\tools\runtime\installer`
`Monospace lowercase mixed with uppercase`	Indicates commands and command-line options, for example: `ippsFFTFwd_CToC_32fc( xleft, Xleft, ctxN2, buffer )`
`UPPERCASE MONOSPACE`	Indicates system variables, for example, `$PATH`

Table 1-1 Notational conventions

Monospace italic	Indicates a parameter in discussions: routine parameters, for example, *lda*; makefile parameters, for example, *functions_list*; etc. When enclosed in angle brackets, indicates a placeholder for an identifier, an expression, a string, a symbol, or a value, for example, *<ipp directory>*.
[items]	Square brackets indicate that the items enclosed in brackets are optional.
{ item | item }	Braces indicate that only one of the items listed between braces should be selected. A vertical bar (|) separates the items

Getting Started with Intel® IPP

2

This chapter helps you start using the Intel® Integrated Performance Primitives (Intel® IPP) by giving basic information you need to know and describing the necessary steps you need to perform after the installation of the product.

Checking Your Installation

Once you complete the installation of the Intel IPP, it is useful to perform a basic verification task that confirms proper installation and configuration of the library.

1. Check that the directory you chose for installation has been created. The default installation directory is `C:\Program Files\Intel\IPP\5.3\ia32`.
2. Check that the file `ippenv.bat` is placed in the `tools\env` directory: you can use this file to set the environment variables `PATH`, `LIB`, and `INCLUDE` in the user shell.
3. Check that the dispatching DLLs and the processor-specific DLLs are on the path.
4. Run `ippiDemo.exe` (or `ippsDemo.exe`) from `C:\Program Files\Intel\IPP\5.3\ia32\demo`.
   If you receive error message “This application has failed to start because `ippcore.dll` was not found” or "No DLL were found in the waterfall procedure", this means that the Windows OS is unable to determine the location of the Intel IPP dynamic libraries. There are four solutions to this problem:
   - Ensure that the `IPP` library folder is in the path: before using the Intel IPP dynamic libraries, add `C:\Program Files\Intel\IPP\5.3\ia32` to the PATH environment variable as described in Setting Environment Variables.
   - Use the Intel IPP Runtime Installer (RTI) located in `IPP\5.3\ia32\tools\runtime` to automatically install the dynamic libraries to the `\system32` directory.
   - Manually copy the contents of `IPP\5.3\ia32\bin` to the `\system32` folder.

- Copy contents of `IPP\5.3\ia32\bin` to the application folder.

Note that you need to delete all Intel IPP DLLs from previous releases from the `C:\winnt\system and C:\winnt\system32` directories. Verify that paths to older library versions are not listed in the PATH environment variable.

Obtaining Version Information

To obtain information about the active library version including the version number and the licensing information, do one of the following:

- run the tool `iplid.exe` from the `\tools\support` folder, or
- call the `ippGetLibVersion` function. See the *Support Functions* chapter in the *Intel IPP Reference Manuals (v.3)* for the function description and calling syntax.

Building Your Application

Follow the procedures described below to build a Windows* OS application.

Setting Environment Variables

The batch file `ippenv.bat` in the `tools/env` directory sets Intel® IPP `LIB`, `INCLUDE`, and `PATH` environment variables for a command prompt session.

To set the environment variables outside of a single command prompt session, complete the following steps in Windows XP*:

1. Right-click the **My Computer** icon located on your desktop or from the Windows Explorer and select **Properties** (or click **Start > Control Panel** and select **System**),
2. Select the **Advanced** tab,
3. Select the **Environment Variables** button,
4. Use the interface to set the environment variables for only the current user (top dialog box) or for anyone who uses the system (bottom dialog box),
5. Select the variable you wish to modify and click the **Edit** button,
6. Add the path to related Intel IPP files to the existing list, for example:

   Select `LIB` and then type in the directory for the Intel IPP stub libraries (default is `C:\Program Files\Intel\IPP\5.3\ia32\stublib`),

   Select `INCLUDE` and then type in the directory for the Intel IPP header files (default is `C:\Program Files\Intel\IPP\5.3\ia32\include`),

   Select `PATH` and then type in the directory for the Intel IPP binaries (default is `C:\Program Files\Intel\IPP\5.3\ia32\bin`).

7. Click **OK** in the **Edit User Variable** dialog box,
8. Click **OK** in the **Environment Variables** dialog box,
9. Click **OK** in the **Systems Properties** dialog box.

For information on how to set up environment variables for threading, refer to section Intel IPP Threading and OpenMP* Support in Chapter 6.

Including Header Files

Intel IPP functions and types are defined in several header files organized by function group and are located in the `\include` directory. For example, the `ipps.h` file contains declarations for all signal processing functions.

The file `ipp.h` includes all Intel IPP header files. For forward compatibility, include only `ipp.h` in your program.

Calling IPP Functions

Due to the DLL dispatcher and merged static library mechanisms described in Linking Your Application with Intel® IPP, calling Intel IPP functions is as simple as calling any other C function.

To call Intel IPP function:

1. Include the `ipp.h` header file
2. Set up the function parameters
3. Call the function

The multiple versions of optimized code for each function are concealed under a single entry point. Refer to the *Intel IPP reference manuals* for details of function descriptions and required parameters.

Before You Begin Using Intel IPP

Before you get started using the Intel IPP, sorting out a few important basic concepts will help you get off to a good start.

The table below summarizes some important things to consider before you start using Intel IPP.

Table 2-1 What you need to know before you get started

Question you need to have answer to.	Identify the Intel IPP function domain that questions to answer belong to. **Reason**: If you know the function domain you intend to use, it will narrow the search in the Reference Manuals for specific routines you need. Besides, you may easily find a sample you would like to run from http://www.intel.com/software/products/ipp/samples.htm. Refer to Table 5-8 to understand what function domains are and to Table 5-9 to understand what kind of cross-domain dependency is introduced.
Linking model	Decide what linking model is appropriate for linking. **Reason:** If you use a linking model that is most appropriate for you, you will get the best linking results. For information on the benefits of each linking model, linking command syntax and examples, link libraries as well as on other linking topics like how to create a custom dynamic library, see Linking Your Application with Intel® IPP

Configuring Your Development Environment 3

This chapter provides information on configuring the Visual C++* environment for using Intel® IPP with Intel® C++ Compiler 10.0 and Microsoft* C++ Compiler. In each case, the location (path) to the Intel IPP is added to the project settings.

Configuring Microsoft* Visual C++*.NET 2003 or Microsoft Visual C++*2005 Environment

To configure Microsoft* Visual C++*.NET 2003 or Microsoft* Visual C++* 2005 environment to link with Intel IPP, follow the sequence of steps below:

1. Select **View > Solution Explorer** (and make sure this window is active),
2. Select **Tools > Options > Projects > VC++ Directories**,
3. In the drop down menu titled **Show directories for:**, select **Include Files**, and then type in the directory for the Intel IPP include files (for example, default is: `C:\Program Files\Intel\IPP\5.3\ia32\include`),
4. In the drop down menu titled **Show directories for:**, select **Library Files**, and then type in the directory for the Intel IPP library files (for example, default is: `C:\Program Files\Intel\IPP\5.3\ia32\stublib` or `C:\Program\Files\Intel\IPP\5.3\ia32\lib`),
5. In the drop down menu titled **Show directories for:**, select **Executable Files** and then type in the directory for the Intel IPP executable files (for example, default is: `C:\Program Files\Intel\IPP\5.3\ia32\bin`),
6. On the main toolbar, select **Project > Properties > Linker > Input** and in the **Additional Dependencies** line add the libraries you wish to link to (for example, `ipps.lib` or `ippsmerged.lib`). For more information on choosing the best linkage model for your Intel IPP application, please refer to Linking Your Application with Intel® IPP.

Using Intel® IPP with Intel® C++ Compiler

Using Intel IPP with Intel C++ Compiler is similar as using Intel IPP with Microsoft* C++ Compiler.

In Microsoft Visual C++* .NET environment, instead of providing settings at **Tools > Options > Projects > VC++ Directories**, choose to provide settings at **Tools > Options > Intel® C++** by the following steps:

1. Select **View > Solution Explorer** (and make sure this window is active),
2. Select **Tools > Options > Projects > Tools > Options > Intel® C++ > Intel® C++ XX**,
3. In the drop down menu **Show directories for:**, select **Include Files** and then type in the directory for the Intel IPP include files (for example, default is: `C:\Program Files\Intel\IPP\5.3\ia32\include`),
4. In the **Include** path, type in the directory for the Intel IPP library files (for example, default is: `C:\Program Files\Intel\IPP\5.3\ia32\stublib` or `C:\Program Files\Intel\IPP\5.3\ia32\lib`),
5. In the **library** path, type in the directory for the Intel IPP executable files (for example, default: `C:\Program Files\Intel\IPP\5.3\ia32\bin`),
6. On the main toolbar, select **Project > Properties > Linker > Input** and in the **Additional Dependencies** line add the libraries you wish to link to (for example, `ipps.lib` or `ippsmerged.lib`).

Intel® IPP Structure 4

This chapter discusses the structure of the Intel® Integrated Performance Primitives (Intel® IPP) after installation as well as the library types supplied.

High-level Directory Structure

Table 4-1 shows a high-level directory structure of Intel IPP after installation.

Table 4-1 High-level directory structure

Directory	File types
<ipp directory>	Main directory (by default: `C:\Program Files\Intel\IPP\5.3\ia32`)
<ipp directory>`\ippEULA.rtf`	End User License Agreement for Intel IPP
<ipp directory>`\bin`	Intel IPP DLLs associated with the IA-32 architecture
<ipp directory>`\demo`	Executable programs that demonstrate various image and signal processing functionalities
<ipp directory>`\doc`	All Intel IPP documentation files
<ipp directory>`\include`	Intel IPP Header Files
<ipp directory>`\lib`	All Intel IPP static libraries associated with the IA-32 architecture
<ipp directory>`\stublib`	All Intel IPP stub libraries associated with the IA-32 architecture. Used for linking DLLs.
<ipp directory>`\tools`	Intel IPP Performance Tests and linkage tools

Supplied Libraries

The Intel® IPP is organized by the following types of library files:

Table 4-2 Intel IPP structure by library types

Library types	Description	Folder location	Example
Import	Contain information about one or more dynamic-link libraries (DLLs), but do not contain the DLL's executable code. Used for loading DLLs	`ia32\stublib`	`ipps.lib`
Dynamic	Include both processor dispatchers and function implementations	`ia32\bin`	`ipps-5.3.dll`, `ippst7-5.3.dll`
Static merged	Contain function implementations for all supported processor types	`ia32\lib`	`ippsmerged.lib`
Static e-merged	Contain processor dispatchers for all functions	`ia32\lib`	`ippsemerged.lib`

Using Intel® IPP Dynamic Link Libraries (DLLs)

The Intel® IPP comes with "stub" library files that load the Intel® IPP DLLs and link to the correct entry points. In order to use the DLLs, link to the files `ipp*.lib` in the `\ia32\stublib` directory (see Table 4-1). You will either need to set your `lib` environment variable using the `ippenv.bat` file or refer to these files using their full path. Including these libraries is all you need to do to dynamically link to the DLL for the appropriate processor.

The DLLs `ipp*.dll` are "dispatcher" DLLs. These DLLs are in the `bin` directory. At run time, they will detect the processor and load the correct processor-specific DLLs. This allows your code to call Intel® IPP functions without worrying about which processor the code will execute on - the appropriate version is automatically used. These processor-specific DLLs are named `*px*.dll`, `*a6*.dll`, `*t7*.dll`, `*w7*.dll`, `*v8*.dll`, and `*p8*.dll` (see Table 5-1). For example, in the `\ia32\bin` directory, `ippiv8-5.3.dll` reflects the imaging processing libraries optimized for Intel Core 2 Duo processors.

The only actions needed to use the Intel® IPP DLLs, once the "stub" static libraries are linked, is to ensure that the dispatching DLLs and the processor-specific DLLs are on the path.

The first and the simplest example of calling IPP function could be the following code in file `t1.cpp`:

Example 4-1 Code calling example

```
#include <stdio.h>
#include <ipp.h>

int main() {
      const IppLibraryVersion* libver = ippGetLibVersion();
      printf("%s %s\n", libver->Name, libver->Version);
      return 0;
}

cmdlinetest>cl /nologo /Fet1.exe -I m:/ipp/windows/include/ t1.cpp
m:\ipp_prod_struct\windows\ia32\stublib\ippcore.lib
t1.cpp

cmdlinetest>t1.exe
ippcore-5.3.dll 5.3 build 81
```

For linking details, see Linking Your Application with Intel® IPP.

Using Intel® IPP Static Libraries

The Intel® IPP comes with "merged" static library files that contain every processor version of each function. These files reside in the `\ia32\lib` directory (see Table 4-1). Just as with the DLL dispatcher, the appropriate version of a function is executed when the function is called. This mechanism is not as convenient as the DLL mechanism, but can result in a smaller total code size in spite of the big size of the static libraries.

To use these static libraries, link to the files `ipp*merged.lib` in the `\lib` directory. You will either need to set your lib environment variable using the `ippenv.bat` file or refer to these files using their full path.

For linking details, see Linking Your Application with Intel® IPP.

Contents of the Documentation Directory

Table 4-3 shows the contents of the `doc` subdirectory in the Intel IPP installation directory.

Table 4-3 Contents of the doc directory

File name	Description
`doc_index.htm`	Index of the Intel IPP documentation
`ReleaseNotes.htm`	General overview of Intel IPP and information about this release
`README.txt`	Initial User Information
`ippsman.pdf`	Intel IPP Reference Manual, contains detailed descriptions of Intel IPP functions and interfaces for signal processing domain
`ippiman.pdf`	Intel IPP Reference Manual, contains detailed descriptions of Intel IPP functions and interfaces for image processing domain
`ippmman.pdf`	Intel IPP Reference Manual, contains detailed descriptions of Intel IPP functions and interfaces for small matrices
`ippcpman.pdf`	Intel IPP Reference Manual, contains detailed descriptions of Intel IPP functions and interfaces for cryptography
`userguide_win_ia32.pdf`	Intel® Integrated Performance Primitives (Intel® IPP) User's Guide, this document

Linking Your Application with Intel® IPP

5

This chapter discusses dispatching of the Intel® IPP software to specific processors using various models for linking the Intel® IPP to an application, considers the differences between the linking models regarding development and target environments, installation specifications, run-time conditions, and other application requirements and helps the user select the linking model that suits him best, shows linking procedure for each linking model, and gives linking examples.

Dispatching

Dispatching refers to detection of your central processing unit (CPU) and selecting the Intel IPP binary that corresponds to the hardware that you are using.

As most Intel IPP functions are optimized for a specific processor, a single Intel IPP function, for example `ippsCopy_8u()`, may have many versions, each one optimized to run on a specific Intel® processor. The Pentium 4 processor-specific function name for this function in merged static library is `w7_ippsCopy_8u()`.

Table 5-1 shows processor codes used for each platform:

Table 5-1 Identification Codes Associated with Processor-Specific Libraries

Abbreviation	Meaning
px	C-optimized for all IA-32 architecture based processors. Can be run on Intel processors starting with Intel® Pentium® Pro processors
a6	Optimized for Pentium® III processors
w7	Optimized for Pentium® 4 processors
t7	Optimized for Pentium® 4 processors with Streaming SIMD Extensions 3 (SSE3)
v8	Optimized for 32-bit applications on Intel® Core™2 and Intel® Xeon® 5100 processors

Table 5-1 Identification Codes Associated with Processor-Specific Libraries

Abbreviation	Meaning
p8	New Optimizations for 32-bit applications on 45nm Intel® Core™2 Duo (Penryn) family processors

Detecting Processor Type

Use the function `ippGetCpuType` declared in the `ippcore.h` file to detect the processor type used in your computer system. It returns an appropriate `IppCpuType` variable value. All of the enumerated values are given in the `ippdefs.h` header file. For example, the return value `ippCpuPII` means that your system uses Intel® Pentium® II processor.

Table 5-2 shows possible values of the function `ippGetCpuType` and their meaning.

Table 5-2 Detecting processor type. Returned values and their meaning

Returned Variable Value	Processor Type
`ippCpuPP`	Intel® Pentium® processor
`ippCpuPMX`	Pentium® processor with MMX(TM) technology
`ippCpuPPR`	Pentium® Pro processor
`ippCpuPII`	Pentium® II processor
`ippCpuPIII`	Pentium® III processor and Pentium® III Xeon® processor
`ippCpuP4`	Pentium® 4 processor and Intel® Xeon® processor
`ippCpuP4HT`	Pentium® 4 processor with Hyper–Threading Technology
`ippCpuP4HT2`	Pentium 4 Processor with Streaming SIMD Extensions 3
`ippCpuCentrino`	Intel® Centrino™ mobile Technology
`ippCpuCoreSolo`	Intel® Core™ Solo processor
`ippCpuCoreDuo`	Intel® Core™ Duo processor
`ippCpuITP = 0x10`	Intel® Itanium® processor
`ippCpuITP2`	Intel® Itanium® 2 processor
`ippCpuEM64T = 0x20`	Intel® 64 Instruction Set Architecture (ISA)
`ippCpuC2D`	Intel® Core™ 2 Duo Processor
`ippCpuC2Q`	Intel® Core™ 2 Quad processor
`ippCpuSSE = 0x40`	Processor supports Streaming SIMD Extensions instruction set
`ippCpuSSE2`	Processor supports Streaming SIMD Extensions 2 instruction set

Table 5-2 Detecting processor type. Returned values and their meaning (continued)

Returned Variable Value	Processor Type
`ippCpuSSE3`	Processor supports Streaming SIMD Extensions 3 instruction set
`ippCpuSSSE3`	Processor with Supplemental Streaming SIMD Extensions 3 instruction set
`ippCpuSSE41`	Processor supports Streaming SIMD Extensions 4.1 instruction set
`ippCpuSSE42`	Processor supports Streaming SIMD Extensions 4.2 instruction set
`ippCpuX8664 = 0x60`	Processor supports 64 bit extension
`ippCpuUnknown`	Unknown Processor

Selecting Between Linking Models

You can dispatch Intel IPP to your machine

- dynamically using the run-time dynamic link libraries (DLLs)
- statically using e-merged and merged static libraries
- statically without automatic dispatching using merged static libraries
- dynamically building your own, custom, DLL.

Answering the following questions will help you select the linkage model which best suites you:

- Are there limitations on how large the application executable can be? Are there limitations on how large the application installation package can be?
- Is the Intel IPP-based application a device driver or similar “ring 0” software that executes in Kernel mode at least some of the time?
- Will various users install the application on a range of processor types, or is the application explicitly supported only on a single type of processor? Is the application part of an embedded computer where only one type of processor is used?
- What resources are available for maintaining and updating customized Intel IPP components? What level of effort is acceptable for incorporating new processor optimizations into the application?
- How often will the application be updated? Will application components be distributed independently or always packaged together?

Dynamic Linking

The dynamic linking model is the simplest and the most commonly used linkage model taking full advantage of the dynamic dispatching mechanism in Windows* OS dynamic link libraries (DLLs) (see Intel® IPP Structure for details). The following table summarizes the dynamic linking features to help you understand the trade-offs of this linking model.

Table 5-3 Summary of dynamic linking features

Benefits	Considerations
• Automatic run-time dispatch of processor-specific optimizations, • Enabling updates with new processor optimizations without recompile/relink, • Reduction of disk space requirements for applications with multiple Intel IPP-based executables, • Enabling more efficient shared use of memory at run-time for multiple Intel IPP-based applications, • Simple redistribution of Intel IPP run-time libraries via RTI package.	• Installation package size: application + ~27-MB RTI package for Intel IPP 5.3. • Application executable requires access to Intel IPP run-time dynamic link libraries (DLLs) to run, • Not appropriate for kernel-mode/device-driver/ring-0 code, • Not appropriate for web applets/plug-ins that require very small download, • There is a one-time performance penalty when the Intel IPP DLLs are first loaded.

To dynamically link with Intel IPP:

1. Add `ipp.h` which will include the include files of all IPP domains,
2. Use the normal IPP function names when calling IPP functions,
3. Link corresponding domain import libraries. For example, if you use the `ippsCopy_8u` function, link against `ipps.lib`,
4. Make sure that run-time libraries, for example `ipps.dll`, are on the executable search path at run time. Run the `ippenv.bat` from directory `\tools\env` to ensure this application built with Intel IPP dynamic libraries will load the appropriate processor-specific DLL.

Static Linking (with Dispatching)

Some applications use only a few Intel® IPP functions and require a small memory footprint. Using the static link libraries via the *e-merged* libraries offers both the benefits of a small footprint and optimization on multiple processors. The e-merged libraries (such as `ippsemerged.lib)` provide an entry point for the non-decorated (with normal names) IPP functions, and the jump table to each processor-specific implementation. When linked with

your application, the function then calls corresponding functions in the merged libraries in accordance with the CPU setting detected by functions in `ippcorel.lib`. The emerged libraries do not contain any implementation code.

The e-merged libraries require initialization before any non-decorated functions can be called. One may choose the function `ippStaticInit()` that initializes the library to use the best optimization available, or the function `ippStaticInitCPU()` that lets you specify the CPU. In any case, one of the "Init" functions must be called before you call any other IPP functions. Without this, (most often issues) performance of your application will not be the best because a "px" version of the IPP functions will be called. To illustrate the performance difference, you can use the following example in the `t2.cpp` file:

Example 5-1 Performance difference with and without calling StaticInit

```
#include <stdio.h>
#include <ipp.h>

int main() {
      const int N = 20000, loops = 100;
      Ipp32f src[N], dst[N];
      unsigned int seed = 12345678, i;
      Ipp64s t1,t2;
      /// no StaticInit call, means PX code, not optimized
      ippsRandUniform_Direct_32f(src,N,0.0,1.0,&seed);
      t1=ippGetCpuClocks();
      for(i=0; i<loops; i++)
            ippsSqrt_32f(src,dst,N);
      t2=ippGetCpuClocks();
      printf("without StaticInit: %.1f clocks/element\n",
            (float)(t2-t1)/loops/N);
      ippStaticInit();
      t1=ippGetCpuClocks();
      for(i=0; i<loops; i++)
            ippsSqrt_32f(src,dst,N);
      t2=ippGetCpuClocks();
      printf("with StaticInit: %.1f clocks/element\n",
            (float)(t2-t1)/loops/N);
      return 0;
}

cmdlinetest>cl /nologo /Fet2.exe -I m:/ipp/windows/include/ t2.cpp /link
/LIBPATH:m:\ipp_prod_struct\windows\ia32\lib\ ippcorel.lib
ippsemerged.lib ippsmerged.lib
t2.cpp

cmdlinetest>t2
without StaticInit: 61.3 clocks/element
with StaticInit: 4.5 clocks/element
```

Table 5-4 summarizes pros and cons of the static linking via the e-merged libraries you should consider before using this particular linking model.

Table 5-4 Summary of features of the static linking (with dispatching)

Benefits	Considerations
• Dispatches processor-specific optimizations during run-time • Creates a self-contained application executable • Generates a smaller footprint than the full set of Intel IPP DLLs	• Intel IPP code is duplicated for multiple Intel IPP-based applications because of static linking, • An additional function call for dispatcher initialization is needed (once) during program initialization

Follow these steps to use static linkage with dispatching:

1. Include `ipps.h` in your code,
2. Before calling any Intel IPP functions, initialize the static dispatcher using either the function `ippStaticInit()` or `ippStaticInitCPU()` declared in `ippcore.h`,
3. Use the normal IPP function names to call IPP functions,
4. Link corresponding e-merged libraries followed by merged libraries, and then `ippcorel.lib`. For example, if the function `ippsCopy_8u()` is used, the linked libraries are `ippsemerged.lib`, `ippsmerged.lib`, and `ippcorel.lib`.

Static Linking (without Dispatching)

This linkage model is most appropriate when a self-contained application is needed, only one processor type is supported, and there are tight constraints on the executable size. One common use is for embedded applications where the application is bundled with only one type of processor.

The Table below summarizes basic features of this model of linking.

Table 5-5 Summary of features of the static linking (without dispatching)

Benefits	Considerations
• Small executable size with support for only one processor type • An executable suitable for kernel-mode/device-driver/ring-0 use • An executable suitable for Web applet or plug-in requiring very small file download and support for only one processor type • Self-contained application executable that does not require Intel IPP run-time DLLs to run • Smallest footprint for application package • Smallest installation package	• The executable is optimized for only one processor type, • Updates to processor-specific optimizations require rebuild and/or relink.

To use merged libraries directly (use `ippsmerged.lib` as an example):

1. include the function prototypes as follows:

   ```
   #define IPPAPI(type,name,arg) extern type __STDCALL w7_##name arg;
   #define IPPCALL(name) w7_##name
   ```

2. include `ipp.h` in your code
3. wrap each Intel® IPP call in a macro as follows:

   ```
   IPPCALL(ippsCopy_8u)(…)
   ```

4. link against `ippsmerged.lib`.

See also *Static linking to Intel® IPP Functions for One Processor* in `C:\Program Files\Intel\IPP\5.3\ia32\tools\staticlib\readme.htm`.

The example below `t3.cpp` demonstrates how to link with a merged library without dispatching. The `GetCpuClocks` function does not have the cpu-specific implementation and does not have a prefix. Because of the redefinitions, functions in `ippcore`, if you use them, should be declared separately.

Example 5-2 Linking with a merged library without dispatching

```
#include <stdio.h>

#define IPPAPI(type,name,arg) extern type __STDCALL w7_##name arg;
#define IPPCALL(name) w7_##name

#include <ipp.h>

extern "C" Ipp64u __STDCALL ippGetCpuClocks();

int main() {
      const int N = 900;
      Ipp32f x[N], mean, stdev;
      unsigned int seed = ippGetCpuClocks();
      IPPCALL(ippsRandUniform_Direct_32f)(x,N,0.0,1.0,&seed);
      IPPCALL(ippsMeanStdDev_32f)(x,N,&mean,&stdev,ippAlgHintFast);
      printf("random signal 0..1 with average %.3f, stddev %.3f\n",
            mean,stdev);
      return 0;
}

cmdlinetest>cl /nologo /Fet3.exe -I m:/ipp/windows/include/ t3.cpp /link
/LIBPATH:m:\ipp_prod_struct\windows\ia32\lib\ ippcorel.lib
ippsmerged.lib
t3.cpp

cmdlinetest>t3
random signal 0..1 with average 0.500, stddev 0.280
```

Custom Dynamic Linking

Some applications have a few internal modules and Intel® IPP code needs to be shared by these modules only. In this case you can use customized dynamic linking, where the customized dynamic link library (DLL) contains only the portion of Intel IPP functions that the application uses. The following table summarizes features of the custom dynamic linking.

Table 5-6 Summary of custom dynamic linking features

Benefits	Considerations
• Run-time dispatching of processor-specific optimizations, • Reduced hard-drive footprint compared with a full set of Intel IPP DLLs, • Smallest installation package to accommodate use of some of the same Intel IPP functions by multiple applications.	• Application executable requires access to Intel IPP run-time DLLs to run, • Developer resources are needed to create and maintain the Custom DLL, • Integration of new processor-specific optimizations requires rebuilding the Custom DLL, • Not appropriate for kernel-mode/device-driver/ring-0 code.

Building and Testing Custom DLL

1. Create an export file with the IPP function names that will be called in your application. For example:

```
/// t4export.def
EXPORTS
     ippGetCpuClocks
     ippsRandUniform_Direct_32f
     ippsMean_32f
```

2. Create a C-file with the DLL initialization function called `DllMain`. Call the `ippStaticInit` function inside `DllMain` in the case of `DLL_PROCESS_ATTACH` to initialize the CPU-specific code dispatching mechanism. For example:

```
/// t4dllmain.c
#define WIN32_LEAN_AND_MEAN
#include <windows.h>

#include <ippcore.h>

BOOL WINAPI DllMain(HINSTANCE hinstDLL,DWORD fdwReason,
     LPVOID lpvReserved) {
     switch (fdwReason) {
        case DLL_PROCESS_ATTACH: ippStaticInit();
     default:
```

```
        break;
    }
    return TRUE;
}
```

3. Create a custom DLL with the functions you are going to call. Call CL compiler and link with the parameters to build DLL with the export functions given in the def file (/def:<file>) and to import the functions from the IPP static libraries. For example:

```
>cl /Im:/ipp/windows/include t4dllmain.c /link /dll /def:t4export.def
/LIBPATH:m:/ipp/windows/lib/win32/ ippcorel.lib ipp semerged.lib
ippsmerged.lib /nologo /OUT:t4dllmain.dll
4dllmain.c
```

Creating library `t4dllmain.lib` and object `t4dllmain.exp`.

4. Test the created DLL library with any test application that calls the functions you include into the DLL. You can create your own header file or you still can include `ipp.h`. The following test application can serve as an example:

```
/// t4.cpp
#include <stdio.h>
#include <ipp.h>

int main() {
    const int N = 16000;
    Ipp32f x[N], mean, stdev;
    unsigned int seed = ippGetCpuClocks();
    ippsRandUniform_Direct_32f(x,N,0.0,1.0,&seed);
    ippsMean_32f(x,N, &mean, ippAlgHintFast);
    printf("random signal 0..1 with average %.3f\n", mean);
    return 0;
}
```

Build the application and run it

```
>cl /Im:\ipp\windows\include t4.cpp t4dllmain.lib /nologo
t4.cpp

>t4
random signal 0..1 with average 0.501
```

There is another way to create a custom DLL that allows more customization. You can define CPU specific code for your targeted processor. For example "w7" code is for Intel® Pentium® 4 processor. Please refer to the latest updated `customdll` samples from Intel IPP samples directory: `\ipp-samples\advanced-usage\linkage\customdll` at http://www.intel.com/software/products/ipp/samples.htm.

Summary of Intel IPP Linkage Model Comparison

The following table gives quick comparison of the IPP linkage models.

Table 5-7 Intel IPP linkage model summary comparison

Feature	Dynamic Linkage	Static Linkage with Dispatching	Static Linkage without Dispatching	Custom Dynamic Linkage
Processor Updates	Automatic	Recompile & redistribute	Release new processor-specific application	Recompile & redistribute
Optimization	All processors	All processors	One processor	All processors
Build	Link to stub static libraries	Link to static libraries and static dispatchers	Link to merged libraries	Build separate DLL
Calling	Regular names	Regular names	Processor-specific names	Regular names
Total Binary Size	Large	Small	Smallest	Small
Executable Size	Smallest	Small	Small	Smallest
Kernel Mode	No	Yes	Yes	No

Selecting Intel® IPP Libraries Needed by Your Application

Follow the below instructions to determine libraries to link to.

Dynamic Linkage

To use the dynamic linking libraries, you need to link to `ipp*.lib` files in the `\stublib` directory. Intel IPP domain-specific functions are included in this directory. For example, `ippj.lib` is the JPEG library and `ippi.lib` is the image processing library. You need to link to all corresponding domain libraries used in your applications.

For example, your application uses three Intel IPP functions `ippicopy_8uC1R`, `ippiCanny_16s8u_C1R` and `ippmMul_mc_32f`. These three functions belong to the image processing domain, computer vision domain and matrix domain respectively. In order to include these functions into your application, you need to link to the following three Intel IPP libraries:

`ippi.lib`

`ippcv.lib`

`ippm.lib`

Static Linkage with Dispatching

To use the static linking libraries, you need to link to `ipp*emerged.lib`, `ipp*merged.lib` and `ippcorel.lib` located in the `\lib` directory. Intel IPP domain-specific functions are included in this directory. For example, `ippjemerged.lib` and `ippjmerged.lib` are the JPEG static libraries. (see Building Your Application for details). Both `ipp*emerged.lib` and `ipp*merged.lib` libraries of each domain you use need to be linked into your application.

For example, your application uses three Intel IPP functions `ippicopy_8uC1R`, `ippiCanny_16s8u_C1R` and `ippmMul_mc_32f`. These three functions belong to the image processing domain, computer vision domain and matrix domain respectively. You need to link the following IPP libraries into your application:

`ippiemerged.lib and ippimerged.lib`

`ippcvemerged.lib and ippcvmerged.lib`

`ippmemerged.lib and ippmmerged.lib`

`ippcorel.lib`

The libraries used for each linking model are listed in the table below.

Table 5-8 Libraries used for each linking model

Domain Description	Header Files	Dynamic Linking	Static Linking with Dispatching & Custom Dynamic Linking	Static Linking without Dispatching
Audio Coding	ippac.h	ippac.lib	ippacemerged.lib	ippacmerged.lib
Color Conversion	ippcc.h	ippcc.lib	ippccemerged.lib	ippccmerged.lib
String Processing	ippch.h	ippch.lib	ippchemerged.lib	ippchmerged.lib
Common Functions	ippcore.h	ippcore.lib	ippcorel.lib	ippcorel.lib
Cryptography	ippcp.h	ippcp.lib	ippcpemerged.lib	ippcpmerged.lib
Computer Vision	ippcv.h	ippcv.lib	ippcvemerged.lib	ippcvmerged.lib
Data Compression	ippdc.h	ippdc.lib	ippdcemerged.lib	ippdcmerged.lib
Image Processing	ippi.h	ippi.lib	ippiemerged.lib	ippimerged.lib
JPEG Primitives	ippj.h	ippj.lib	ippjemerged.lib	ippjmerged.lib
Realistic Rendering	ippr.h	ippr.lib	ippremerged.lib	ipprmerged.lib
Small Matrix	ippm.h	ippm.lib	ippmemerged.lib	ippmmerged.lib
Signal Processing	ipps.h	ipps.lib	ippsemerged.lib	ippsmerged.lib
Speech Coding	ippsc.h	ippsc.lib	ippscemerged.lib	ippscmerged.lib
Speech Recognition	ippsr.h	ippsr.lib	ippsremerged.lib	ippsrmerged.lib
Video Coding	ippvc.h	ippvc.lib	ippvcemerged.lib	ippvcmerged.lib
Vector Math	ippvm.h	ippvm.lib	ippvmemerged.lib	ippvmmerged.lib

Library Dependencies by Domain (Static Linking)

The only way to know the exact dependencies for the Intel IPP functions called from your application is to link to the libraries. For example, if your application uses JPEG functions then you must link to `ippj.lib`. After you link, if there are unresolved symbols to functions with the prefix `ippi`, this means you also need to link to `ippi.lib`. The `ippcore.lib` is required for all domains:

Table 5-9 Library dependencies by domain

Domain	Library	Allowed dependencies
Audio Coding	`ippac`	`ippdc, ipps`
Color Conversion	`ippcc`	`ippi, ipps`
String Processing	`ippch`	`ipps`
Cryptography	`ippcp`	`none`
Computer Vision	`ippcv`	`ippi, ipps`
Data Compression	`ippdc`	`ipps`
Image Processing	`ippi`	`ipps`
JPEG Primitives	`ippj`	`ippi, ipps`
Matrix Math	`ippm`	`ipps`
Realistic Rendering	`ippr`	`ippi, ipps`
Signal Processing	`ipps`	`none`
Speech Coding	`ippsc`	`ipps`
Speech Recognition	`ippsr`	`ipps`
Video Coding	`ippvc`	`ippi, ipps`
Vector Math	`ippvm`	`none`

Refer to *Intel IPP Reference Manuals* to find which domain your function belongs to.

Linking Examples

For more examples, please refer to IPP code samples at
http://www.intel.com/software/products/ipp/samples.htm

Managing Performance and Memory

6

The chapter describes ways you can follow to get the most out of Intel® Integrated Performance Primitives (Intel® IPP) software: aligning memory, thresholding denormal data before Finite Impulse Response (FIR) filtering, reusing buffers, using FFT for algorithmic optimization (where appropriate), supporting multi-threading and OpenMP*. Finally, it gives information how to accomplish Intel IPP functions performance tests by using Intel® IPP Performance Test Tool and gives some examples of using Performance Tool Command Lines.

Memory Alignment

The performance of Intel® IPP, when operating on aligned or misaligned data, can be significantly different. Access to memory is faster if pointers to the data are aligned, and Intel IPP functions perform better if they process data with aligned pointers.

Use the following Intel® IPP functions for pointer alignment, memory allocation and deallocation:

`void* ippAlignPtr( void* ptr, int alignBytes )`

Aligns a pointer, can align to 2/4/8/16/…

`void* ippMalloc( int length )`

32-byte aligned memory allocation. Can only free memory with `ippFree`.

`void ippFree( void* ptr )`

Free memory allocated by `ippMalloc`. Can only free memory allocated by `ippMalloc`.

`Ipp<datatype>* ippsMalloc_<datatype>( int len )`

32-byte aligned memory allocation for signal elements of different data types. Can only free memory with `ippsFree`.

`void ippsFree( void* ptr )`

Free memory allocated by `ippsMalloc`.

`Ipp<datatype>* ippiMalloc_<mod>(int widthPixels, int heightPixels, int* pStepBytes)`

32-byte aligned memory allocation for images where every line of the image is padded with zeros. Can only free memory with `ippiFree`.

`void ippiFree( void* ptr )`

Frees memory allocated by `ippiMalloc`.

Below is the example of calling the `ippiMalloc` function.

Example 6-1 Calling the ippiMalloc function

```
#include "stdafx.h"
#include "ipp.h"
#include "tools.h"

int main(int argc, char *argv[])
{
      IppiSize size = {320, 240};

      int stride;
      Ipp8u* pSrc = ippiMalloc_8u_C3(size.width, size.height, &stride);
      ippiImageJaehne_8u_C3R(pSrc, stride, size);
      ipView_8u_C3R(pSrc, stride, size, "Source image", 0);

      int dstStride;
      Ipp8u* pDst = ippiMalloc_8u_C3(size.width, size.height, &dstStride);
      ippiCopy_8u_C3R(pSrc, stride, pDst, dstStride, size);
      ipView_8u_C3R(pDst, dstStride, size, "Destination image 1", 0);

      IppiSize ROISize = { size.width/2, size.height/2 };
      ippiCopy_8u_C3R(pSrc, stride, pDst, dstStride, ROISize);
      ipView_8u_C3R(pDst, dstStride, ROISize, "Destination image, small", 0);

      IppiPoint srcOffset = { size.width/4, size.height/4 };
      ippiCopy_8u_C3R(pSrc + srcOffset.x*3 + srcOffset.y*stride, stride,
            pDst, dstStride, ROISize);
      ipView_8u_C3R(pDst, dstStride, ROISize, "Destination image, small &
shifted", 1);

      return 0;
}
```

There are no restrictions to the amount of memory that can be allocated except as defined by the user's operating system and system hardware.

NOTE. Intel IPP memory functions differ from the standard `malloc` and `free` functions in that they align the memory to a 32-byte boundary for optimal performance on Intel architecture.

NOTE. Intel IPP `ippFree`, `ippsFree` and `ippiFree` functions cannot be used to free memory allocated by standard functions like `malloc` or `calloc`; nor can the memory allocated by the Intel IPP `malloc` functions be freed by the standard function free

Thresholding Data

Denormal numbers are the border values in the floating point format and special case values for the processor. Operations on denormal data make processing slow, even if corresponding interrupts are disabled. Denormal data occur, for example, in filtering by Infinite Impulse Response (IIR) and Finite Impulse Response (FIR) filters of the signal captured in fixed-point format and converted to the floating point format. To avoid the slowdown effect in denormal data processing, the Intel® IPP threshold functions can be applied to the input signal before filtering. For example:

```
if (denormal_data)
ippsThreshold_LT_32f_I( src, len, 1e-6f );
ippsFIR_32f( src, dst, len, st );
```

The `1e-6` value is the threshold level. The input data below that level are set to zero. Because the Intel IPP threshold function is very fast, the execution of two functions is faster than execution of one, if denormal numbers meet in the source data. Of course, if the denormal data occur while using the filtering procedure, the threshold functions do not help.

In this case, for Intel processors beginning with the Intel® Pentium® 4 processor and including the Itanium® processor and Itanium® 2 processor, it is possible to set a special computation mode - Flush to Zero. You can use the function `ippsSetFlushToZero` for this purpose. Note that this setting takes effect only when computing is done with the Streaming SIMD Extensions (SSE) and Streaming SIMD Extensions 2 (SSE2) instructions.

The following table illustrates how denormal data may affect performance and the effect of thresholding denormal data. As you can see, thresholding takes only three clocks more. On the other hand, denormal data can cause the application performance to drop 250 times.

Table 6-1 Performance resulting from thresholding denormal data

Data/Method	Normal	Denormal	Denormal + Threshold
CPU cycles per element	46	11467	49

Reusing Buffers

Some Intel® IPP functions require internal memory for various optimization strategies. At the same time you should be well aware that memory allocation inside of the function may have a negative impact on performance in some situations, for example, in the case of cache misses. To avoid or minimize memory allocation and keep your data in warm cache, some functions, for example, FFT and DFT transform, can use or reuse memory given as a parameter to the function.

If you have to call an FFT function many times, the reuse of an external buffer results in better performance. A simple example of this kind of processing is perform filtering using FFT transform or computing FFT as two FFT in two separate threads:

```
ippsFFTInitAlloc_C_32fc(  &ctxN2, order-1, IPP_FFT_DIV_INV_BY_N,
ippAlgHintAccurate );

ippsFFTGetBufSize_C_32fc( ctxN2, &sz );
buffer = sz > 0 ? ippsMalloc_8u( sz ) : 0;

int phase = 0;
/// prepare source data for two FFTs

ippsSampleDown_32fc( x, fftlen, xleft, &fftlen2, 2, &phase );
phase = 1;
ippsSampleDown_32fc( x, fftlen, xrght, &fftlen2, 2, &phase );

/// two FFTs may be calculated separately in two threads
ippsFFTFwd_CToC_32fc( xleft, Xleft, ctxN2, buffer );
ippsFFTFwd_CToC_32fc( xrght, Xrght, ctxN2, buffer );
```

The external buffer is not necessary. If the pointer to the buffer is 0, the function allocates memory inside

Using FFT

Using Fast Fourier Transform (FFT) is a universal method to increase performance of data processing, especially in the field of digital signal processing where filtering is essential.

The convolution theorem states that filtering of two signals in the spatial domain can be computed as point-wise multiplication in the frequency domain. The data transform to and from the frequency domain is usually fulfilled using the Fourier transform. You can apply the Finite Impulse Response (FIR) filter to the input signal using Intel® IPP FFT functions, which are very fast on Intel® processors. You can also increase the data array length to the next greater power of two by padding the array with zeroes and then applying the FFT forward transform function to the input signal and the FIR filter coefficients. Fourier coefficients obtained in this way are multiplied point-wise and the result can easily be transformed back to the spatial domain. The performance gain achieved by using FFT could be very significant.

If the applied filter is the same for several processing iterations then the FFT transform of the filter coefficients need to be done only once. The twiddle tables and the bit reverse tables are created in the initialization function for the forward and for inverse transforms at the same time. The main operations in this kind of filtering are presented below:

```
ippsFFTInitAlloc_R_32f( &pFFTSpec, fftord, IPP_FFT_DIV_INV_BY_N,
ippAlgHintNone );
/// perform forward FFT to put source data x to freq-domain

ippsFFTFwd_RToPack_32f( xx, XX, pFFTSpec, 0 );
/// point-wise multiplication in freq-domain is convolution

ippsMulPack_32f_I( HH, XX, fftlen );
/// perform inverse FFT to get result in time-domain

ippsFFTInv_PackToR_32f( XX, yy, pFFTSpec, 0 );
/// free FFT tables

ippsFFTFree_R_32f( pFFTSpec );
```

Another way to significantly improve performance is by using FFT and multiplication for processing large size data. Note that the zeros in the example above could be pointers to the external memory, which is another way to increase performance. Note that the Intel IPP signal processing FIR filter is implemented using FFT and you do not need to create a special implementation of the FIR functions.

Intel IPP Threading and OpenMP* Support

Intel IPP supports multi-threading in the dynamic libraries, which gives significant performance gain on multi-processor and multi-core systems. All tested functions are thread-safe.

The static libraries do not support multi-threading.

Some Intel® IPP functions contain OpenMP* code. These functions include color conversion, filtering, convolution, cryptography, cross correlation, matrix computation, square distance, and bit reduction, etc.

Go to http://support.intel.com/support/performancetools/libraries/ipp/sb/CS-026584.htm to downloaded the list for each version.

Getting Information on Number of Threads

To find the number of threads created by the Intel IPP, call the function `ippGetNumThreads`.

Controlling Number of Threads

To set the number of threads created, call the function `ippSetNumThreads`.

Preventing Intel IPP from Creating Threads

To disable multi-threading, use the static libraries which are not multi-threaded or set the variable `ippSetNumThreads` to 1.

Running Intel® IPP Performance Test Tool

The Intel® IPP Performance Test Tool for the Windows* OS based on Intel® Pentium® processors and Intel® Itanium® processors is a very functional timing system designed to accomplish Intel IPP functions performance tests on the same hardware platforms as the related Intel IPP libraries. It contains command line programs for testing the performance of each IPP function in various ways.

You can control the course of tests and generate the results in the desirable format by using command line options. The results are saved in the `.csv` file for further processing with Microsoft* Excel* program. The course of timing is displayed on the console and can be saved in a `.txt` file. You can create a list of functions to be tested and set required parameters with which the function should be called during the performance test. The list of functions to be tested and their parameters can either be defined in the `.ini` file, or entered directly from the console.

In the enumeration mode, the Intel IPP performance test tool creates a list of the timed functions on the console and in the `.txt` or `.csv` file.

Additionally, this performance test tool provides all performance test data in `.csv` format. It contains data covering all domains and CPU types supported in Intel IPP. For example, you can read that reference data referring to ia32 platform in sub-directory `\ia32\tools\perfsys\data`.

Once you fully install the Intel® IPP package, you can find the performance test `.exe` files located in the `\ia32\tools\perfsys` directory. For instance, `ps_ipps.exe` is a tool to measure the Intel IPP Signal Processing functions. Similarly, you will find appropriate executable files for each Intel IPP domain.

The command line format is:

```
<ps_FileName>.exe [switch_1] [switch_2] … [switch_n]
```

A short reference for the command line options can be displayed on the console. To invoke it, just enter `-?` or `-h` in the command line:

```
ps_ipps.exe -h
```

The command line options can be divided into six groups by their functionality. You can enter options in an arbitrary order with at least one space between. Some options (like `–v`, `-V`, `-o`, `-O`) may be entered several times with different file names and option `-f` may be entered several times with different function patterns. See Appendix A Performance Test Tool Command Line Options for descriptions of the possible options.

Examples of Using Performance Test Tool Command Lines

The following examples illustrate how you can use performance tool common command lines to generate IPP function performance data.

Example 1. Running in the standard mode:

`ps_ippch.exe –B –v`

All IPP string functions are tested by the default timing method on standard data (`-B` option). The results are generated in the file `ps_ippch.csv` (`-v` option).

Example 2. Testing selected functions:

`ps_ipps.exe -fFIRLMS_32f -v firlms.csv`

Measure signal domain function `FIRLMS_32f` (`-f` option), and generate a `.csv` file named `firlms.csv` (`-v` option).

Example 3. Retrieving function lists:

`ps_ippvc.exe -e –o vc_list.txt`

The output file `vc_list.txt` (`-o` option) will list all IPP video coding functions (`-e` option).

`ps_ippvc.exe -e -v H264.csv -f H264`

The list of functions with names containing `H264` (`-f` option) that may be tested (`-e` option) is displayed on the console and stored in the file `H264.csv` (`-v` option).

Example 4. Launching performance test tool with the .ini file:

`ps_ipps.exe –B –I`

The `.ini` file `ps_ipps.ini` is created after the first run (`-l` option).

`ps_ippi.exe –i –v`

The second run tests all functions, reading timing procedure, and all function parameters values from the `ps_ipps.ini` file (`-i` option), and the output file `ps_ipps.csv` (`-v` option) is generated.

Language-Specific Usage Options

7

This chapter discusses some special aspects of using Intel® Integrated Performance Primitives (Intel® IPP) with different programming languages and the Windows* OS development environments.

In addition to the C programming language, Intel® IPP functions are compatible with C++, Fortran, C#*, Visual Basic*, Object Pascal, Java*.

Using Intel® IPP in Java* Applications

You may call Intel IPP functions in your Java application by using the Java* Native Interface (JNI*). There is some overhead associated with JNI use, especially when the input data size is small. Combining several functions into one JNI call and using managed memory will help improve the overall performance.

Using Intel® IPP with Borland C++ Builder*

You cannot directly link to the Intel® IPP static library in Borland C++ Builder*. Instead, try the following:

1. Use the Intel IPP merged static libraries to create a custom DLL (export the functions called in your application). The custom DLL and merged static library tools are available as part of the integration sample package. The Intel IPP static libraries are located in the `IPP\5.3\ia32\lib` directory.
2. Use `IMPLIB` (on the DLL) to create an import library in the Borland environment.

Performance Test Tool Command Line Options

A

The following table gives brief descriptions of the possible performance test tool command line options.

Table A-1 Performance test tool command line options

Groups	Options	Descriptions
1. Adjusting Console Input	-A	ask all parameters from console
	-B	batch mode
2. Managing Output	-v[<filename>]	create .csv file and write PS results
	-V[<filename>]	add PS results to csv-file
	-o[<file-name>]	create .txt file and write console output
	-O[<file-name>]	add console output to .txt file
	-L<ERR|WARN|PARM|INFO|TRACE>	set detail level of the console output
	-e	enumerate tests and exit
	-g[<file-name>]	signal file is created just at the end of the whole testing
	-FL	Write result as number of float operations per second. This option is supported for few functions only, because it's not required by users
3. Selecting Functions for Testing	-f < or-pattern>	run tests of functions with pattern in name, case sensitive
	-f-<not-pattern>	Not test functions with pattern in name, case sensitive
	-f+<and-pattern>	run only tests of functions with pattern in name, case sensitive
	-f=< eq-pattern>	run tests of functions with this full name, case sensitive

Table A-1 Performance test tool command line options (continued)

Groups	Options	Descriptions
4. Operation with .ini Files	-i[<file-name>]	read PS parameters from i.ni file
	-I[<file-name>]	write PS parameters to ini-file and exit
	-P	read tested function names from .ini file
5. Direct Data Input	-d<name>=<value>	set PS parameter value
6. Muti-Thread Timing	-MT<numThreads>	run timing in several threads simultaneously
	-T<HIGH|NORMAL|LOW>	set high or normal priority for threads, the priority level may be specified by entering only the first letter -T H[IGH] High priority. It is a default value if a multi-thread timing is not set, or if the number of threads is equal to 1. -T N[ORMAL] Normal priority. It is a default value if the number of threads is greater than 1. -T L[OW] Low priority. It is recommended for a multi-thread timing if functions use OpenMP* technology.

Index

P

R

S

T

U

V