Intel® Integrated Performance Primitives

User's Guide

IPP 8.0 Update 1

328847-002US

Legal Information

Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

BlueMoon, BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Cilk, Core Inside, E-GOLD, Flexpipe, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel CoFluent, Intel Core, Intel Inside, Intel Insider, the Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel vPro, Intel Xeon Phi, Intel XScale, InTru, the InTru logo, the InTru Inside logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Pentium, Pentium Inside, Puma, skoool, the skoool logo, SMARTi, Sound Mark, Stay With It, The Creators Project, The Journey Inside, Thunderbolt, Ultrabook, vPro Inside, VTune, Xeon, Xeon Inside, X-GOLD, XMM, X-PMU and XPOSYS are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Java is a registered trademark of Oracle and/or its affiliates.

Introducing the Intel® Integrated Performance Primitives

Use the Intel® Integrated Performance Primitives (Intel® IPP) to improve performance of multimedia, enterprise data, embedded, communications, and scientific/technical applications. The primitives are a common interface for thousands of commonly used algorithms. Using these primitives enables you to automatically tune your application to many generations of processors without changes in your application.

Intel IPP library provides high performance implementations of signal, image, and data processing functions for several hardware/instruction set generations. Code written with Intel IPP automatically takes advantage of available CPU capabilities. This can provide tremendous development and maintenance savings. You can write programs with one optimized execution path, avoiding the alternative of multiple paths (Intel® Streaming SIMD Extensions 2, Supplemental Streaming SIMD Extensions 3, Intel® Advanced Vector Extensions , etc.) to achieve optimal performance across multiple generations of processors.

Intel IPP provides a set of examples to help you get started with common operations like resize and fast Fourier transform (FFT).

NOTE
The high-level multimedia API samples from previous versions of Intel IPP are no longer in active development.

The goal of the Intel IPP software is to provide algorithmic building blocks with

- a simple "primitive" C interface and data structures to enhance usability and portability
- faster time-to-market
- scalability with Intel® hardware

What's New

This User's Guide documents the Intel® Integrated Performance Primitives (Intel® IPP) 8.0 Update 1 release. Multiple fixes have been made in the document.

Getting Help and Support

If you did not register your Intel® software product during installation, please do so now at the Intel® Software Development Products Registration Center. Registration entitles you to free technical support, product updates, and upgrades for the duration of the support term.

For general information about Intel technical support, product updates, user forums, FAQs, tips and tricks and other support questions, please visit http://www.intel.com/software/products/support/.

NOTE
If your distributor provides technical support for this product, please contact them rather than Intel.

For technical information about the Intel IPP library, including FAQ's, tips and tricks, and other support information, please visit the Intel IPP forum: http://software.intel.com/en-us/forums/intel-integrated-performance-primitives/ and browse the Intel IPP knowledge base: http://software.intel.com/en-us/articles/intel-ipp-kb/all/.

Notational Conventions

The following font and symbols conventions are used in this document:

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/ia32/perfsys`
`Monospace lowercase mixed with UPPERCASE`	Indicates commands and command-line options, for example: `ps_ipps.exe -f FIRLMS_32f -r firlms.csv`
`UPPERCASE MONOSPACE`	Indicates system variables, for example: `$PATH`.
`monospace italic`	Indicates a parameter in discussions, such as routine parameters, for example: *`pSrc`*; makefile parameters, for example: *`function_list`*. 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 can be selected. A vertical bar (|) separates the items.

The following notations are used to refer to Intel IPP directories:

`<parent product directory>`	The installation directory for the larger product that includes Intel IPP; for example, Intel® C++ Composer XE.
`<ipp directory>`	The main directory where Intel IPP is installed: *`<ipp directory>`*=*`<parent product directory>`*`/ipp`. Replace this placeholder with the specific pathname in the configuring, linking, and building instructions.

Getting Started with Intel® Integrated Performance Primitives

1

This chapter helps you start using Intel® Integrated Performance Primitives (Intel® IPP) by giving a quick overview of some fundamental concepts and showing how to build an Intel® IPP program.

Finding Intel® IPP on Your System

Intel® Integrated Performance Primitives (Intel® IPP) installs in the subdirectory referred to as *<ipp directory>* inside *<parent product directory>*. For example, when installed as part of the Intel® C++ Composer XE, the installation directory is as follows:

- On Windows* OS: `C:/Program Files (x86)/Intel/Composer XE 2013 SP1/ipp` (on certain systems, instead of `Program Files (x86)`, the directory name is `Program Files`)
- On Linux* OS and OS X*: `/opt/intel/composer_xe_2013_sp1/ipp`

The tables below describe the structure of the high-level directories on:

- Windows* OS
- Linux* OS
- OS X*

Windows* OS:

Directory	Contents
Subdirectories of *<ipp directory>*	
`bin`	Batch files to set environmental variables in the user shell
`bin/ia32`	Batch files for the IA-32 architecture
`bin/intel64`	Batch files for the Intel® 64 architecture
`include`	Header files for the library functions
`lib/ia32`	Single-threaded static and dynamic libraries for the IA-32 architecture
`lib/intel64`	Single-threaded static and dynamic libraries for the Intel® 64 architecture
`lib/`*<arch>*`/threaded`, where *<arch>* is one of `{ia32, intel64}`	Multi-threaded libraries
`examples`	Intel IPP example files
Subdirectories of *<parent product directory>*	
`tool/perfsys`	Command-line tools for Intel IPP performance testing
`tool/staticlib`	Header files for redefining Intel IPP functions to processor-specific counterparts
`redist/ia32/ipp`	Single-threaded DLLs for applications running on processors with the IA-32 architecture

Directory	Contents
redist/intel64/ipp	Single-threaded DLLs for applications running on processors with the Intel® 64 architecture
redist/<*arch*>/threaded	Multi-threaded DLLs
redist/<*arch*>/1033	Intel IPP message catalog
Documentation/<*locale*>/ipp, where <*locale*> is one of {en_US, ja_JP}	Intel IPP documentation
Documentation/vshelp/1033/intel.ippdocs	Help2-format files for integration of the Intel IPP documentation with the Microsoft Visual Studio* 2008 IDE
Documentation/msvhelp/1033/ipp	Microsoft Help Viewer*-format files for integration of the Intel MKL documentation with the Microsoft Visual Studio* 2010/2012 IDE.

Linux* OS:

Directory	Contents
Subdirectories of <*ipp directory*>	
bin	Scripts to set environmental variables in the user shell
bin/ia32	Shell scripts for the IA-32 architecture
bin/intel64	Shell scripts for the Intel® 64 architecture
include	Header files for the library functions
lib/ia32	Single-threaded static and dynamic libraries for the IA-32 architecture
lib/intel64	Single-threaded static and dynamic libraries for the Intel® 64 architecture
lib/<*arch*>/threaded	Multi-threaded libraries
lib/<*arch*>/nonpic	Non-PIC single-threaded static libraries
lib/<*arch*>/<*locale*>	Intel IPP message catalog
examples	Intel IPP example files
Subdirectories of <*parent product directory*>	
tool/perfsys	Command-line tools for Intel IPP performance testing
tool/staticlib	Header files for redefining Intel IPP functions to processor-specific counterparts
Documentation/<*locale*>/ipp	Intel IPP documentation

OS X*:

Directory	Contents
Subdirectories of <*ipp directory*>	
bin	Scripts to set environmental variables in the user shell
include	Header files for the library functions
lib	Single-threaded static and dynamic FAT libraries

Directory	Contents
lib/<arch>/threaded	Multi-threaded FAT libraries
lib/<arch>/<locale>	Intel IPP message catalog
examples	Intel IPP examples files
Subdirectories of <parent product directory>	
tool/perfsys	Command-line tools for Intel IPP performance testing
Documentation/<locale>/ipp	Intel IPP documentation

See Also

Notational Conventions

Setting Environment Variables

When the installation of Intel IPP is complete, set the environment variables in the command shell using one of the script files in the `bin` subdirectory of the Intel IPP installation directory:

On Windows* OS:

`ia32/ippvars_ia32.bat`	for the IA-32 architecture,
`intel64/ippvars_intel64.bat`	for the Intel® 64 architecture,
`ippvars.bat`	for the IA-32 and Intel® 64 architectures.

On Linux* OS and OS X*:

Architecture	Shell	Script File
IA-32	C	`ia32/ippvars_ia32.csh`
IA-32	Bash	`ia32/ippvars_ia32.sh`
Intel® 64	C	`intel64/ippvars_intel64.csh`
Intel® 64	Bash	`intel64/ippvars_intel64.sh`
IA-32 and Intel® 64	C	`ippvars.csh`
IA-32 and Intel® 64	Bash	`ippvars.sh`

When using the `ippvars` script, you need to specify the architecture as a parameter. For example:

- `ippvars.bat ia32`

  sets the environment for Intel IPP to use the IA-32 architecture on Windows* OS.
- `. ippvars.sh intel64`

  sets the environment for Intel IPP to use the Intel® 64 architecture on Linux* OS and OS X*.

The scripts set the following environment variables:

Windows* OS	Linux* OS	OS X*	Purpose
IPPROOT	IPPROOT	IPPROOT	Point to the Intel IPP installation directory
LIB	LD_LIBRARY_PATH	DYLD_LIBRARY_PATH	Add the search path for the Intel IPP single-threaded libraries
PATH	n/a	n/a	Add the search path for the Intel IPP single-threaded DLLs

Windows* OS	Linux* OS	OS X*	Purpose
INCLUDE	n/a	n/a	Add the search path for the Intel IPP header files
n/a	NLSPATH	NLSPATH	Add the search path for the Intel IPP message catalogs

Compiler Integration

Intel® C++ Compiler and Microsoft Visual Studio* compilers simplify developing with Intel® IPP.

On Windows* OS, a default installation of Intel® C++ Composer XE and Intel® IPP installs integration plug-ins. These enable the option to configure your Microsoft Visual Studio* project for automatic linking with Intel IPP.

Intel® C++ Compiler also provides command-line parameters to set the link/include directories:

- On Windows* OS:
  `/Qipp-link` and `/Qipp`
- On Linux* OS and OS X*:
  `-ipp-link` and `-ipp`

See Also

Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP
Linking Your Application with Intel® Integrated Performance Primitives

Building Intel® IPP Applications

The code example below represents a short application to help you get started with Intel® IPP:

```
#include "ipp.h"
#include <stdio.h>
int main(int argc, char* argv[])
{
        const IppLibraryVersion *lib;
        Ipp64u fm;
        IppStatus status;

        status= ippInit();            //IPP initialization with the best optimization layer
        if( status != ippStsNoErr ) {
                printf("IppInit() Error:\n");
                printf("%s\n", ippGetStatusString(status) );
                return -1;
        }

        //Get version info
        lib = ippiGetLibVersion();
        printf("%s %s\n", lib->Name, lib->Version);

        //Get CPU features enabled with selected library level
        fm=ippGetEnabledCpuFeatures();
        printf("SSE    :%c\n",(fm>>1)&1?'Y':'N');
        printf("SSE2   :%c\n",(fm>>2)&1?'Y':'N');
        printf("SSE3   :%c\n",(fm>>3)&1?'Y':'N');
        printf("SSSE3  :%c\n",(fm>>4)&1?'Y':'N');
        printf("SSE41  :%c\n",(fm>>6)&1?'Y':'N');
        printf("SSE42  :%c\n",(fm>>7)&1?'Y':'N');
        printf("AVX    :%c\n",(fm>>8)&1 ?'Y':'N');
        printf("AVX2   :%c\n", (fm>>15)&1 ?'Y':'N' );
        printf("----------\n");
        printf("OS Enabled AVX :%c\n", (fm>>9)&1 ?'Y':'N');
```

```
        printf("AES             :%c\n", (fm>>10)&1?'Y':'N');
        printf("CLMUL           :%c\n", (fm>>11)&1?'Y':'N');
        printf("RDRAND          :%c\n", (fm>>13)&1?'Y':'N');
        printf("F16C            :%c\n", (fm>>14)&1?'Y':'N');

        return 0;
}
```

This application consists of three sections:

1. IPP initialization. This stage is required to take advantage of full IPP optimization. If the Intel IPP program runs without `ippInit()`, by default the least optimized implementation is chosen. With `ippInit()` the best optimization layer will be dispatched at runtime. In certain debugging scenarios, it is helpful to force a specific implementation layer by using `ippInitCpu`, instead of the best as chosen by the dispatcher.
2. Get the library layer name and version. You can also get the version information by using the `ippversion.h` file located in the `/include` directory.
3. Show the hardware optimizations used by this library layer.

Building the First Example with Microsoft Visual Studio* 2010/2012 Integration on Windows* OS

On Windows* OS, Intel IPP applications are significantly easier to build with Microsoft* Visual Studio* 2010 and Microsoft* Visual Studio* 2012. To build the code example above, follow the steps:

1. Start Microsoft Visual Studio* and create an empty C++ project.
2. Add a new c file and paste the code into it.
3. Set the include directories and the linking model as described in Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP.
4. Compile and run the application.

If you did not install the integration plug-in, configure your Microsoft* Visual Studio* IDE to build Intel IPP applications following the instructions provided in Configuring the Microsoft Visual Studio* IDE to Link with Intel® IPP.

Building the First Example on Linux* OS and OS X*

To build the code example above on Linux* OS and OS X*, follow the steps:

1. Paste the code into the editor of your choice.
2. Make sure the compiler variables are set in your shell. For information on how to set environment variables see Setting Environment Variables.
3. Compile with the following command: `icc ipptest.cpp -o ipptest -I $IPPROOT/include -L $IPPROOT/lib/intel4 -lippi -lipps -lippcore`. For more information about which Intel IPP libraries you need to link to, see Library Dependencies by Domain and Linking Options.
4. Run the application.

> **NOTE**
> Internally threaded (multi-threaded) version of Intel® IPP library is deprecated but still available for legacy applications. It is strongly recommended to use the single-threaded version of the libraries for new development.

See Also

Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP
Configuring the Microsoft* Visual Studio* IDE to Link with Intel® IPP
Setting Environment Variables
Library Dependencies by Domain
Linking Options
Dispatching

Using Intel® IPP Examples

This section provides information on Intel IPP examples directory structure and examples build system.

Intel® IPP Examples Directory Structure

The Intel IPP package includes code examples, located in the `ipp-examples.zip` archive at the *<ipp directory>/examples/* subdirectory. The examples archive contains the following directories:

Directory	Contents
`builder`	Command files and scripts to build Intel IPP examples
`documentation`	Documentation for the Intel IPP examples (`ipp-examples.html`)
`sources/ipp-examples`	Intel IPP examples source code to maintain common build procedure with the Intel IPP Samples
`sources/ipp-examples/common`	Source code files (`.cpp`/`.h`) for all examples
`sources/ipp-examples/ipp_fft`	Fast Fourier transformation example
`sources/ipp-examples/ipp_resize_mt`	Image resizing example
`sources/ipp-examples/ipp_thread`	Example of external threading of Intel IPP functions

NOTE
Intel® IPP samples are no longer in active development and available as a separate download.

See Also

Finding Intel® IPP on Your System

Building Intel® IPP Examples

You can build Intel® IPP examples automatically using CMake*. This build system provides the following advantages:

- Automatically generates build files across all permutations of samples, operating systems, architectures, IDE versions, static/dynamic linkage, single-/multi-threaded functions, and debug/release.
- Configuration files are cross-platform, simplifying maintenance of consistent behavior across all operating systems that Intel IPP supports.
- Creates build files for a wide variety of IDEs including the latest versions of Microsoft Visual Studio*, Xcode*, and Eclipse*.
- Creates build files specifically for your environment.

For building instructions refer to `documentation/ipp-examples.html` provided with the */<ipp directory>/examples/ipp-examples.zip* archive.

Examples Build System Requirements

- CMake* 2.8.8 or higher, available for download from http://www.cmake.org/.
- Perl 5.12 or higher. Build scripts for Windows* OS are validated with the community edition of ActiveState* Perl.

See Also

Intel® IPP Examples Directory Structure

Finding the Intel® IPP Documentation

The <*ipp directory*>`/Documentation/en_US/ipp` directory, set up during installation, includes a lot of helpful documentation related to Intel® IPP. See the `ipp_documentation.htm` file for a listing of all the available documents with links or pointers to their locations.

Additional documentation on the Intel IPP examples (`documentation/ipp-examples.html`) is available in the `<ipp directory>/examples/ipp-examples.zip` archive.

The Intel IPP forum and knowledge base can be useful locations to search for questions not answered by the documents above. Please see: http://software.intel.com/en-us/forums/intel-integrated-performance-primitives/ .

See Also

Finding Intel® IPP on Your System

Intel® Integrated Performance Primitives Theory of Operation 2

This section discusses dispatching of the Intel® Integrated Performance Primitives (Intel® IPP) libraries to specific processors, provides functions and parameters naming conventions, and explains the data types on which Intel IPP performs operations. This section also provides Intel IPP domain details, including existing library dependencies by domain.

Dispatching

Intel® IPP uses multiple function implementations optimized for various CPUs. Dispatching refers to detection of your CPU and selecting the corresponding Intel IPP binary path. For example, the `ippie9-8.0` library in the `/redist/intel64/ipp` directory contains the image processing libraries optimized for 64-bit applications on processors with Intel® Advanced Vector Extensions (Intel® AVX) enabled such as the 2nd Generation Intel® Core™ processor family.

A single Intel IPP function, for example `ippsCopy_8u()`, may have many versions, each one optimized to run on a specific Intel® processor with specific architecture, for example, the 64-bit version of this function optimized for the 2nd Generation Intel® Core™ processor is `e9_ippsCopy_8u()`, and version optimized for 64-bit applications on processors with Intel® Streaming SIMD Extensions 4.1 (Intel® SSE 4.1) is `y8_ippsCopy_8u()`. This means that a prefix before the function name determines CPU model. However, during normal operation the dispatcher determines the best version and you can call a generic function (`ippsCopy_8u` in this example).

Intel® IPP is designed to support application development on various Intel® architectures. This means that the API definition is common for all processors, while the underlying function implementation takes into account the strengths of each hardware generation.

By providing a single cross-architecture API, Intel IPP enables you to port features across Intel® processor-based desktop, server, and mobile platforms. You can use your code developed for one processor architecture for many processor generations.

The following table shows processor-specific codes that Intel IPP uses:

Description of Codes Associated with Processor-Specific Libraries

IA-32 Intel® architecture	Intel® 64 architecture	Description
px	**mx**	Generic code optimized for processors with Intel® Streaming SIMD Extensions (Intel® SSE)
w7		Optimized for processors with Intel SSE2
	m7	Optimized for processors with Intel SSE3
v8	**u8**	Optimized for processors with Supplemental Streaming SIMD Extensions 3 (SSSE3), including the Intel® Atom™ processor
p8	**y8**	Optimized for processors with Intel SSE4.1
g9	**e9**	Optimized for processors with Intel® Advanced Vector Extensions (Intel® AVX) and Intel® Advanced Encryption Standard New Instructions (Intel® AES-NI)
h9	**l9**	Optimized for processors with Intel AVX2

Function Naming Conventions

Intel IPP functions have the same naming conventions for all domains.

Function names in Intel IPP have the following general format:

`ipp<data-domain><name>_<datatype>[_<descriptor>](<parameters>)`

NOTE
The core functions in Intel IPP do not need an input data type. These functions have `ipp` as a prefix without the data-domain field. For example, `ippGetStatusString`.

See Also

Core and Support Functions

Data-domain

The *data-domain* element is a single character indicating type of input data. Intel IPP supports the following data-domains:

`s`	one-dimensional operations on signals, vectors, buffers
`i`	two-dimensional operations on images, video frames
`m`	vector or matrix operations
`g`	generated functions (deprecated)
`r`	rendering functionality and three-dimensional data processing (deprecated)

Primitive vs. Variant Name

The *name* element identifies the algorithm or operation of the function. The low-level algorithm that function implements is a *primitive*. This algorithm often has several *variants* for different data types and implementation variations.

For example, the `CToC` modifier in the `ippsFFTInv_CToC_32fc` function signifies that the inverse fast Fourier transform operates on complex floating point data, performing the complex-to-complex (CToC) transform.

Functions for matrix operations have an object type description as a modifier. For example, `ippmMul_mv` signifies multiplication of a matrix by a vector.

Data Types

The *datatype* element indicates data types used by the function, in the following format:

`<bit depth><bit interpretation>,`

where

`bit depth = <1|8|16|32|64>`

and

`bit interpretation<u| s|f>[ c]`

Here *u* indicates "unsigned integer", *s* indicates "signed integer", *f* indicates "floating point", and *c* indicates "complex".

For functions that operate on a single data type, the *datatype* element contains only one value.

If a function operates on source and destination signals that have different data types, the respective data type identifiers are listed in the function name in order of source and destination as follows:

<datatype> = <src1Datatype>[src2Datatype][dstDatatype]

For more information about supported data types see the *Intel® IPP Reference Manual* available in the Intel® Software Documentation Library.

See Also

Intel® Software Documentation Library

Descriptor

The optional *descriptor* element describes the data associated with the operation. Descriptors are individual characters that indicate additional details of the operation.

The Intel IPP functions use the following descriptors:

Descriptor	Description	Example
A	Image data contains an alpha channel as the last channel, requires C4, alpha-channel is not processed.	ippiFilterMax_8u_AC4R
A0	Image data contains an alpha channel as the first channel, requires C4, alpha-channel is not processed.	ippiLUTPalette_8u_C3A0C4R
A*xx*	Advanced arithmetic operations with *xx* bits of accuracy.	ippsPowx_32f_A11
C	The function operates on a specified channel of interest (COI) for each source image.	ippiSet_8u_C3CR
C*n*	Image data consists of *n* channels. Possible values for *n*: 1, 2, 3, 4.	ippiFilterBorder_32f_C1R
D*x*	Signal is x-dimensional (default is D1).	ippiGenScaleLevel8x8_H264_8u16s_D2
D		
I	Operation is in-place (default is not-in-place).	ippsAdd_16s_I
L	One pointer is used for each row (in D2).	ippsJoin_32f16s _D2L
M	Operation uses a mask to determine pixels to be processed.	ippiCopy_8u_C1MR

Descriptor	Description	Example
P*n*	Image data consists of *n* discrete planar (not-interleaved) channels with a separate pointer to each plane. Possible values for *n*: 1, 2, 3, 4.	ippiReconstructLumaIntra8x8_H264High_32s1
R	Function operates on a defined region of interest (ROI) for each source image.	ippiMean_8u_C4R
S	Function with standard description of the objects (for matrix operation).	ippmCopy_va_32f_SS
s	Saturation and no scaling (default).	ippiConvert_16s16u_C1Rs
Sfs	Saturation and fixed scaling mode (default is saturation and no scaling).	ippsConvert_16s8s_Sfs

The descriptors in function names are presented in the function name in alphabetical order.

Some data descriptors are default for certain operations and not added to the function names. For example, the image processing functions always operate on a two-dimensional image and saturate the results without scaling them. In these cases, the implied descriptors `D2` (two-dimensional signal) and `s` (saturation and no scaling) are not included in the function name.

Parameters

The *parameters* element specifies the function parameters (arguments).

The order of parameters is as follows:

- All source operands. Constants follow vectors.
- All destination operands. Constants follow vectors.
- Other, operation-specific parameters.

A parameter name has the following conventions:

- All parameters defined as pointers start with *p*, for example, *pPhase*, *pSrc*; parameters defined as double pointers start with *pp*, for example, *ppState*. All parameters defined as values start with a lowercase letter, for example, *val*, *src*, *srcLen*.
- Each new part of a parameter name starts with an uppercase character, without underscore; for example, *pSrc*, *lenSrc*, *pDlyLine*.
- Each parameter name specifies its functionality. Source parameters are named *pSrc* or *src*, in some cases followed by names or numbers, for example, *pSrc2*, *srcLen*. Output parameters are named *pDst* or *dst* followed by names or numbers, for example, *pDst2*, *dstLen*. For in-place operations, the input/output parameter contains the name *pSrcDst* or *srcDst*.

Intel® Integrated Performance Primitives Domain Details

Intel IPP is divided into groups of related functions. Each subdivision is called *domain*, and has its own header file, static libraries, dynamic libraries, and tests. The table below lists each domain's code, header and functional area.

The file `ipp.h` includes Intel IPP header files with the exception of cryptography and generated functions. If you do not use cryptography and generated functions, include `ipp.h` in your application for forward compatibility. If you want to use cryptography or generated functions, you must directly include `ippcp.h` and `ippgen.h` in your application.

Code of Domain	Header file	Prefix	Description
AC	ippac.h	ipps	audio coding

Code of Domain	Header file	Prefix	Description
CC	ippcc.h	ippi	color conversion
CH	ippch.h	ipps	string operations
CP	ippcp.h	ipps	cryptography
CV	ippcv.h	ippi	computer vision
DC	ippdc.h	ipps	data compression
DI*	ippdi.h	ipps	data integrity
GEN*	ippgen.h	ippg	spiral generated
I	ippi.h	ippi	image processing
J	ippj.h	ippi	JPEG compression
M	ippm.h	ippm	small matrix operations
R*	ippr.h	ippr	realistic rendering and 3D data processing
S	ipps.h	ipps	signal processing
SC	ippsc.h	ipps	speech codecs
VC	ippvc.h	ippi	video codecs
VM	ippvm.h	ipps	vector math

* refers to deprecated domain.

NOTE
The data integrity, generated, and rendering domains are deprecated. Please avoid starting any new projects with these domains.

Library Dependencies by Domain

When you link to a certain Intel® IPP domain library, you must also link to the libraries on which it depends. The following table lists library dependencies by domain.

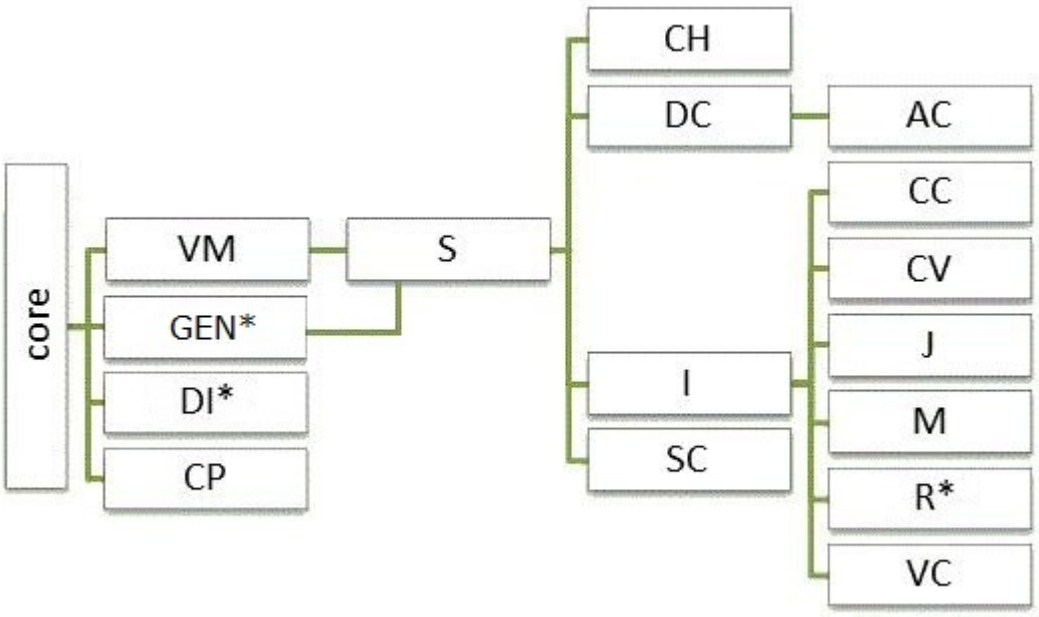
Library Dependencies by Domain

Domain	Domain Code	Depends on
Audio Coding	AC	Core, VS, S, DC
Color Conversion	CC	Core, VM, S, I
String Operations	CH	Core, VM, S
Cryptography	CP	Core
Computer Vision	CV	Core, VM, S, I
Data Compression	DC	Core, VM, S
Data Integrity*	DI*	Core
Generated Functions*	GEN*	Core, S

Domain	Domain Code	Depends on
Image Processing	I	Core, VM, S
Image Compression	J	Core, VM, S, I
Small Matrix Operations	M	Core, VM, S, I
Realistic Rendering and 3D Data Processing*	R	Core, VM, S, I
Signal Processing	S	Core, VM
Speech Coding	SC	Core, VM, S
Video Coding	VC	Core, VM, S, I
Vector Math	VM	Core

* refers to deprecated domain

The figure below represents the domain internal dependencies graph.



To find which domain your function belongs to, refer to the *Intel© IPP Reference Manual* available in the Intel® Software Documentation Library.

See Also

Intel® Software Documentation Library

Linking Your Application with Intel® Integrated Performance Primitives

3

This section discusses linking options available in Intel® Integrated Performance Primitives (Intel® IPP).

The current version of Intel IPP introduces several changes in linking approaches by starting the deprecation for internal threading.

The Intel IPP library supports the following linking options:

- Single-threaded dynamic
- Single-threaded static
- Multi-threaded dynamic (deprecated)
- Multi-threaded static (deprecated)

Optimization Notice
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Linking Options

Intel® Integrated Performance Primitives (Intel® IPP) is distributed as:

- **Static library:** static linking results in a standalone executable
- **Dynamic/shared library:** dynamic linking defers function resolution until runtime and requires that you bundle the redistributable libraries with your application

Previous generations of Intel® IPP implemented threading inside some functions (listed in the `ThreadedFunctionsList.txt` file). This simplified getting started with threading but, due to issues with performance and interoperability with other threading models, is no longer recommended for new development. Versions of the library with internal threading enabled for a subset of functions are distributed as a separate package.

The following table provides description of libraries available for linking.

	Single-threaded	**Multi-threaded**
	(non-threaded)	(internally threaded)
Description	Suitable for application-level threading	Use only when no other form of threading is active
Found in	Main package	Separate download
	After installation: *<ipp directory>*`/lib/`*<arch>*	After installation: *<ipp directory>*`/lib/`*<arch>*`/threaded`
Static linking	Windows* OS: `mt` suffix in a library name (`ipp`*<domain>*`mt.lib`)	Windows* OS: `mt` suffix in a library name (`ipp`*<domain>*`mt.lib`)

Dynamic Linking	Linux* OS and OS X*: no suffix in a library name (`libipp<domain>.a`) Default (no suffix) Windows* OS: `ipp<domain>.lib` Linux* OS: `libipp<domain>.` OS X*: `libipp<domain>.dylib`	Linux* OS and OS X*: no suffix in a library name (`libipp<domain>.a`) Default (no suffix) Windows* OS: `ipp<domain>.lib` Linux* OS: `libipp<domain>.` OS X*: `libipp<domain>.dylib`

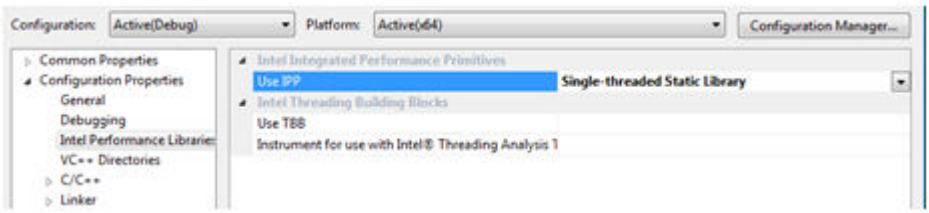
NOTE
Internally threaded (multi-threaded) versions of Intel® IPP libraries are deprecated but made available for legacy applications. It is strongly recommended to use the single-threaded version of the libraries for new development.

See Also
Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP
Configuring the Microsoft* Visual Studio* IDE to Link with Intel® IPP

Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP

After a default installation of the Intel® IPP and Intel® C++ Composer XE, you can easily configure your project to automatically link with Intel IPP. Configure your Microsoft* Visual Studio* project for automatic linking with Intel IPP as follows:

- For the Visual Studio* 2010/2012 development environment:
  1. Go to **Project**>**Properties**>**Configuration Properties**>**Intel Performance Libraries**.
  2. Change the **Use IPP** property setting by selecting one of the options to set the include directories and the linking model, as shown on the screen shot below.



- For the Visual Studio* 2008 development environment:
  1. On the main toolbar select **Project**>***{your project name}*** **Properties**. **Property Pages** dialog box will show up.
  2. Select **Configuration Properties**>**Linker**>**Input** and add the required Intel IPP libraries (for example, dynamic link: `ippi.lib ipps.lib ippcore.lib` or static link: `ippimt.lib ippsmt.lib ippcoremt.lib`) to the **Additional Dependencies** field.

Programming Considerations

4

Core and Support Functions

There are several general purpose functions that simplify using the library and report information on how it is working:

- `Init/InitCPU`
- `GetStatusString`
- `GetLibVersion`
- `Malloc/Free`

Init/InitCpu

The `ippInit` function detects the processor type and sets the dispatcher to use the processor-specific code of the Intel® IPP library corresponding to the instruction set capabilities available.

In some cases like debugging and performance analysis, you may want to get the data on the difference between various processor-specific codes on the same machine. Use the `ippInitCpu` function for this. This function sets the dispatcher to use the processor-specific code according to the specified processor type without querying the system.

The `ippInit` and `ippInitCpu` functions are a part of the `ippCore` library.

GetStatusString

The `ippGetStatusString` function decodes the numeric status return value of Intel® IPP functions and converts them to a human readable text:

```
status= ippInit();
if( status != ippStsNoErr ) {
        printf("IppInit() Error:\n");
        printf("%s\n", ippGetStatusString(status) );
        return -1;
}
```

The `ippGetStatusString` function is a part of the `ippCore` library.

GetLibVersion

Each domain has its own `GetLibVersion` function that returns information about the library layer in use from the dispatcher. The code snippet below demonstrates the usage of the `ippiGetLibVersion` from the image processing domain:

```
const IppLibraryVersion* lib = ippiGetLibVersion();
printf("%s %s %d.%d.%d.%d\n", lib->Name, lib->Version,
lib->major, lib->minor, lib->majorBuild, lib->build);
```

Use this function in combination with `ippInitCpu` to compare the output of different implementations on the same machine.

Malloc/Free

Intel IPP functions provide better performance if they process data with aligned pointers. Intel IPP provides the following functions to ensure that data is 32-byte aligned:

```
void* ippMalloc(int length)
void ippFree(void* ptr)
```

The `ippMalloc` function provides a 32-byte aligned buffer, and the `ippFree` function frees it.

The signal and image processing libraries provide `ippsMalloc` and `ippiMalloc` functions, respectively, to allocate a 32-byte aligned buffer that can be freed by the `ippsFree` and `ippiFree` functions.

NOTE

- When using buffers allocated with routines different from Intel IPP, you may get better performance if the starting address is aligned. If the buffer is created without alignment, use the `ippAlignPtr` function.
- Intel® IPP `Malloc` functions fail to allocate buffers of size more than 2GB-1 because the size parameters are 32-bit signed integers. It is not recommended to use larger buffers from other allocators. See Cache Optimizations for more details.

For more information about the Intel IPP functions see the *Inte® Integrated Performance Primitives for Intel® Architecture Reference Manual* available in Intel® Software Documentation Library.

See Also

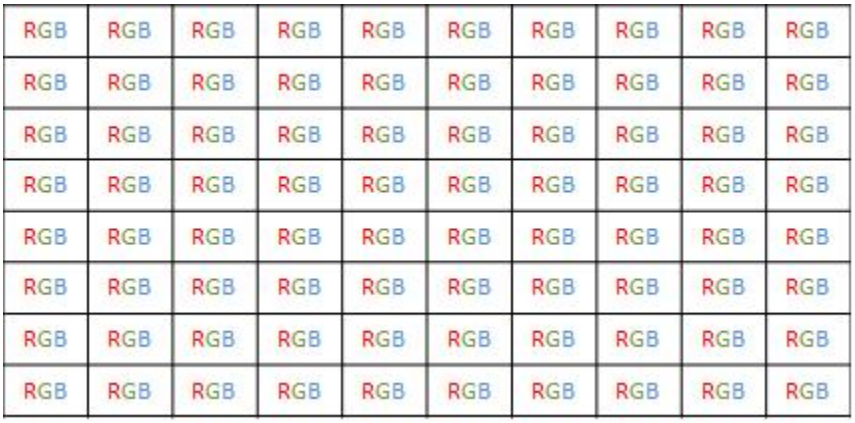
Cache Optimizations
Intel® Software Documentation Library

Channel and Planar Image Data Layouts

Intel® IPP functions operate on two fundamental data layouts: channel and planar.

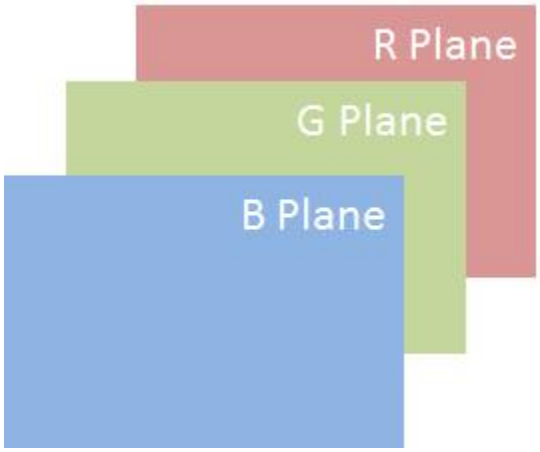
In channel format, all values share the same buffer and all values for the same pixel position are interleaved together. Functions working with channel data have a `_Cn` descriptor, where *n* can take one of the following values: 1, 2, 3, or 4. The figure below shows 24 bit per pixel RGB data, which is represented as `_C3`.

RGB data in `_C3` layout



For planar format, there is one value per pixel but potentially several related planes. Functions working with planar data have a `_Pn` descriptor, where *n* can take one of the following values: 1, 2, 3, or 4. The figure below shows 24 bit per pixel RGB data represented as `_P3`.

RGB data in `_P3` layout



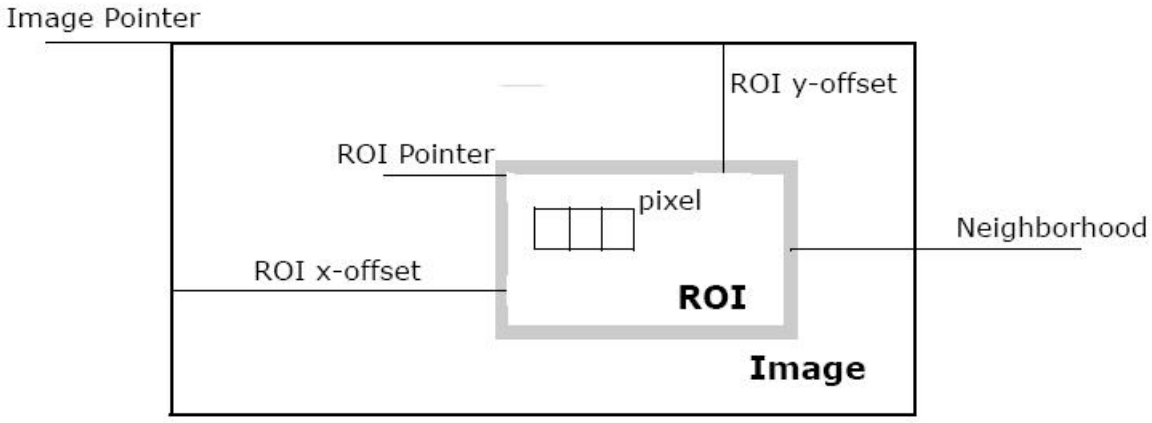
NOTE
For many video and image processing formats planes may have different sizes.

Regions of Interest

Many Intel® IPP image processing functions operate with a region of interest (ROI). These functions include an `R` descriptor in their names.

A ROI can be the full image or a subset. This can simplify thread or cache blocking.

Many functions sample a neighborhood and cannot provide values for an entire image. In this case a ROI must be defined for the subset of the destination image that can be computed.



Managing Memory Allocations

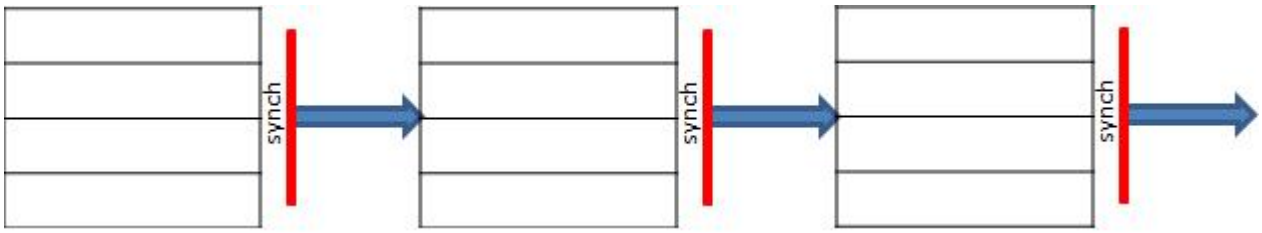
Depending on implementation layer and specific parameters, some Intel© IPP functions need different amount of memory for coefficients storage and working buffers. To address this, follow the steps below:

1. Compute the size of the required buffer using the <*operation function*>`GetSize` function.
2. Set up any buffers needed for initialization.
3. Initialize the specification structure for the operation.
4. Free the buffers need for initialization only.
5. Set up working buffers for the main operation.
6. Do the main operation.
7. Free the specification and working buffers.

Cache Optimizations

To get better performance, work should be grouped to take advantage of locality in the lowest/fastest level of cache possible. This is the same for threading or cache blocking optimizations.

For example, when operations on each pixels in an image processing pipeline are independent, the entire image is processed before moving to the next step. This may cause many inefficiencies, as shown in a figure below.



In this case cache may contain wrong data, requiring re-reading from memory. If threading is used, the number of synchronization point/barriers is more than the algorithm requires.

You can get better performance after combining steps on local data, as shown in a figure below. In this case each thread or cache-blocking iteration operates with ROIs, not full image.

NOTE
It is recommended to subdivide work into smaller regions considering cache sizes, especially for very large images/buffers.

Programming with Intel® Integrated Performance Primitives in the Microsoft* Visual Studio* IDE

5

This section provides instructions on how to configure your Microsoft* Visual Studio* IDE to link with the Intel® IPP, explains how to access Intel IPP documentation and use IntelliSense* Sense features.

Configuring the Microsoft* Visual Studio* IDE to Link with Intel® IPP

Steps for configuring Microsoft Visual C/C++* development system for linking with Intel® Integrated Performance Primitives (Intel® IPP) depend on whether you installed the C++ Integration(s) in Microsoft Visual Studio* component of the Intel® Composer XE:

- If you installed the integration component, see Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP
- If you did not install the integration component or need more control over Intel IPP libraries to link, you can configure the Microsoft Visual Studio* by performing the following steps. Though some versions of the Visual Studio* development system may vary slightly in the menu items mentioned below, the fundamental configuring steps are applicable to all these versions.

  1. In Solution Explorer, right-click your project and click **Properties** (for Visual Studio* 2010 or higher) or select **Tools**>**Options** (for Visual Studio* 2008)
  2. Select **Configuration Properties**>**VC++ Directories** (for Visual Studio* 2010 or higher) or select **Projects and Solutions** (for Visual Studio* 2008) and set the following from the **Select directories for** drop down menu:

     - **Include Files** menu item, and then type in the directory for the Intel IPP include files (default is `<ipp directory>\include`)
     - **Library Files** menu item, and then type in the directory for the Intel IPP library files (default is `<ipp directory>\lib`)
     - **Executable Files** menu item, and then type in the directory for the Intel IPP executable files (default is `<parent product directory>\redist\<arch>\ipp\`)

Viewing Intel® IPP Documentation in Visual Studio* IDE

Viewing Intel IPP Documentation in Document Explorer (Visual Studio* 2008 IDE)

Intel IPP documentation integrates into the Visual Studio IDE help collection. To open Intel IPP help, do the following:

1. Select **Help**>**Contents** from the menu.
2. From the list of **VS Help** collections choose **Intel® Integrated Performance Primitives for Intel® Architecture Documentation**.

To open the help index, select **Help**>**Index** from the menu. To search for help topics, select **Help**>**Search** from the menu and enter a search string.

You can filter Visual Studio help collections to show only content related to installed Intel tools. To do this, select "Intel" from the **Filtered by** list. This hides the contents and index entries for all collections that do not refer to Intel.

Viewing Intel IPP Documentation in Visual Studio* 2010 IDE

To access the Intel IPP documentation in Visual Studio* 2010 IDE, do the following:

- Configure the IDE to use local help. To do this, go to **Help**>**Manage Help Settings** and check **I want to use local help**.
- Select **Help**>**View Help** menu item to view a list of available help collections and open Intel IPP documentation.

Viewing Intel IPP Documentation in Visual Studio* 2012 IDE

To access the Intel IPP documentation in Visual Studio* 2012 IDE, do the following:

- Configure the IDE to use local help. To do this, go to **Help**>**Set Help Preference** and check **Launch in Help Viewer**.
- Select **Help**>**View Help** menu item to view a list of available help collections and open Intel IPP documentation.

Using Context-sensitive Help

Context-sensitive help enables easy access to the description of a function whose name is typed in the Code Editor. You can use *F1 Help* and/or *Dynamic Help.*

To enable Dynamic Help, go to **Help > Dynamic Help**. The **Dynamic Help** window opens that displays links relevant to your current selection.

To access the function description,

1. Select the function name in the Code Editor
2. Do one of the following:
   - Click **F1**
   - Click the link to the description in the Dynamic Help window.

The topic with the function description opens in the window that displays search results.

Using the IntelliSense* Features

Intel IPP supports two Microsoft* Visual Studio IntelliSense* features that support language references: Complete Word and Parameter Info.

> **NOTE**
> Both features require header files. Therefore, to benefit from IntelliSense, make sure the path to the include files is specified in the Visual Studio solution settings. On how to do this, see Configuring the Microsoft Visual Studio* IDE to Link with Intel® IPP.

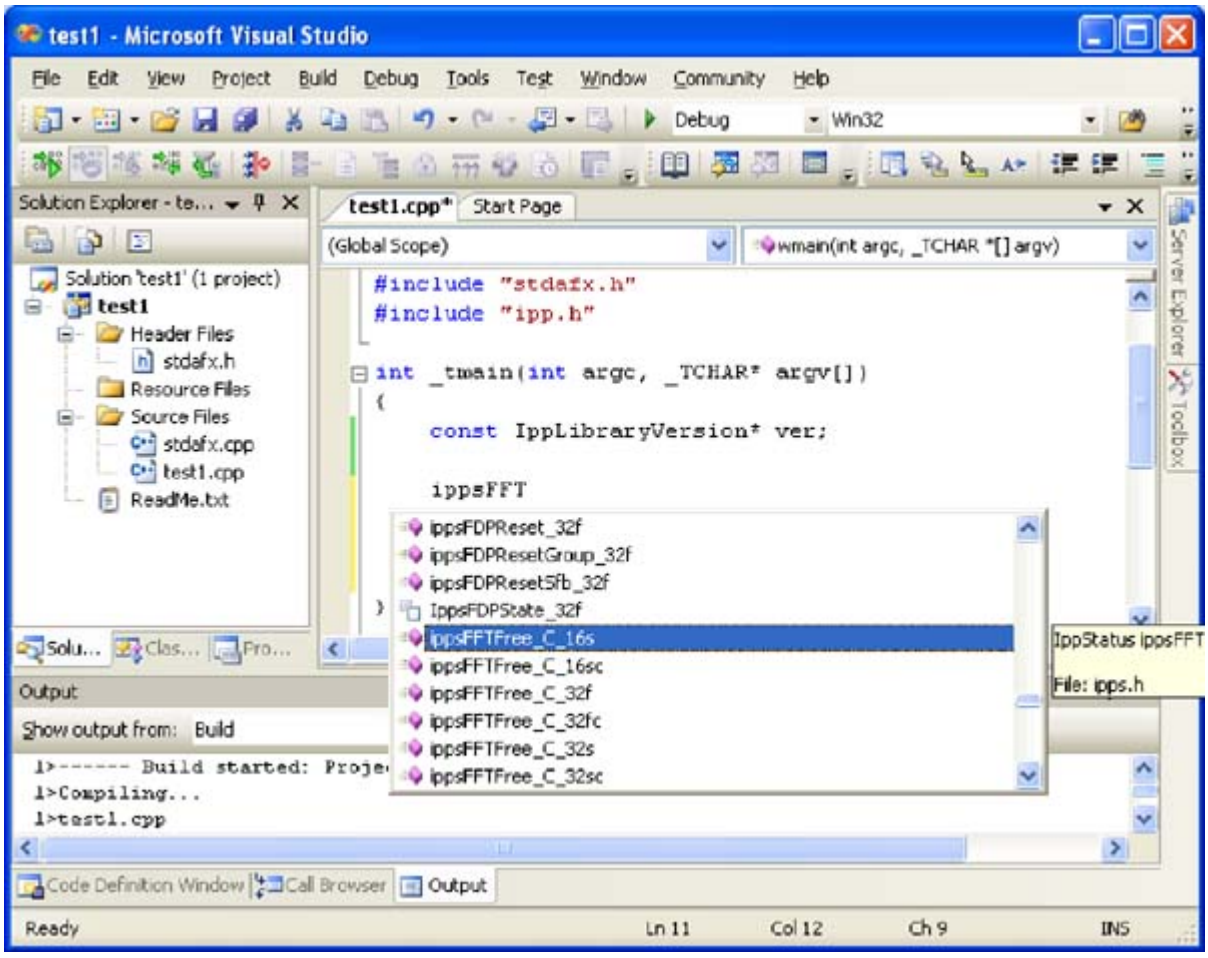
Complete Word

For a software library, the *Complete Word* feature types or prompts for the rest of the name defined in the header file once you type the first few characters of the name in your code.

Provided your C/C++ code contains the include statement with the appropriate Intel IPP header file, to complete the name of the function or named constant specified in the header file, follow these steps:

1. Type the first few characters of the name (for example, `ippsFFT`).

2. Press **Alt** + **RIGHT ARROW** or **Ctrl** + **SPACEBAR** If you have typed enough characters to eliminate ambiguity in the name, the rest of the name is typed automatically. Otherwise, the pop-up list of the names specified in the header file opens - see the figure below.



3. Select the name from the list, if needed.

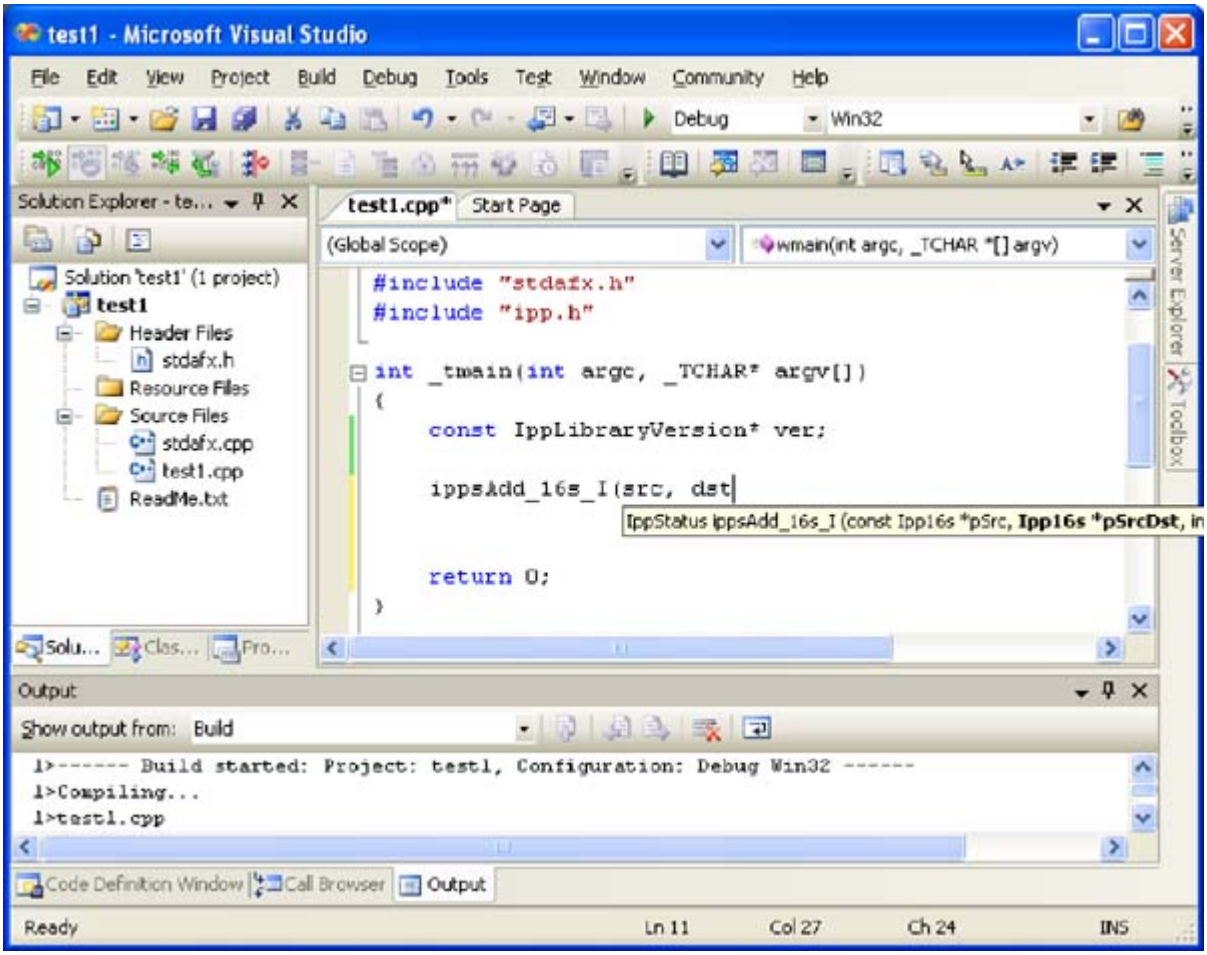
Parameter Info

The *Parameter Info* feature displays the parameter list for a function to give information on the number and types of parameters.

To get the list of parameters of a function specified in the header file, follow these steps:

1. Type the function name
2. Type the opening parenthesis

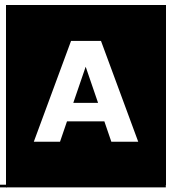
A tooltip appears with the function API prototype, and the current parameter in the API prototype is highlighted - see the figure below.



See Also

Configuring the Microsoft* Visual Studio* IDE to Link with Intel© IPP

Appendix A: Performance Test Tool (perfsys) Command Line Options



Intel® Integrated Performance Primitives (Intel® IPP) installation includes command-line tools for performance testing in the *<parent product directory>*/tools/perfsys directory. There is one perfsys tool for each domain. For example, ps_ipps executable measures performance for all Intel IPP signal processing domain functions.

Many factors may affect Intel IPP performance. One of the best way to understand them is to run multiple tests in the specific environment you are targeting for optimization. The purpose of the perfsys tools is to simplify performance experiments and empower developers with useful information to get the best performance from Intel IPP functions.

With the command-line options you can:

- Create a list of functions to test
- Set parameters for each function
- Set image/buffer sizes

To simplify re-running specific tests, you can define the functions and parameters in the initialization file, or enter them directly from the console.

The command-line format is:

```
ps_ipp*.exe [option_1] [option_2] ... [option_n]
```

To invoke the short reference for the command-line options, use -? or -h commands:

```
ps_ipp*.exe -h
```

The command-line options are divided into several groups by functionality. You can enter options in arbitrary order with at least one space between each option name. Some options (like -r, -R, -o, -O) may be entered several times with different file names, and option -f may be entered several times with different function patterns. For detailed descriptions of the perfsys command-line options see the following table:

Performance Test Tool Command Line Options

Group	Option	Description
Set optimization layer to test	-T[*cpu-name*]	Call ippInitStaticCpu(ippCpu<*cpu-name*>)
Report Configuration	-A<Timing|Params|Misalign|All>	Prompt for the parameters before every test from console
	-o[<*file-name*>]	Create <*file-name*>.txt file and write console output to it
	-O[<*file-name*>]	Add console output to the file <*file-name*>.txt
	-L <*ERR*|*WARN*|*PARM*|*INFO*|*TRACE*>	Set detail level of the console output
	-r[<*file-name*>]	Create <*file-name*>.csv file and write perfsys results to it
	-R[<*file-name*>]	Add test results to the file <*file-name*>.csv
	-q[<*file-name*>]	Create <*file-name*>.csv and write function parameter name lines to it

Group	Option	Description
	-q+	Add function parameter name lines to perfsys results table file
	-Q	Exit after creation of the function parameter name table
	-u[<*file-name*>]	Create <*file-name*>.csv file and write summary table ('_sum' is added to default file name)
	-U[<*file-name*>]	Add summary table to the file <*file-name*>.csv ('_sum' is added to default file name)
	-g[<*file-name*>]	Create signal file at the end of the whole testing
	-l<*dir-name*>	Set default directory for output files
	-k<and|or>	Compose different keys (-f, -t, -m) by logical operation
	-F<*func-name*>	Start testing from function with func-name full name
	-Y<*HIGH/NORMAL*>	Set high or normal process priority (normal is default)
	-H[ONLY]	Add 'Interest' column to .csv file [and run only hot tests]
	-N<*num-threads*>	Call ippSetNumThreads(<*num-treads*>)
	-s[-]	Sort or do not sort functions (sort mode is default)
	-e	Enumerate tests and exit
	-v	Display the version number of the perfsys and exit
	-@<*file-name*>	Read command-line options for the specified file
Set function parameters	-d<*name*>=<*value*>	Set perfsys parameter value
Initialization files	-i[<*file-name*>]	Read perfsys parameters from the file <*file-name*>.ini
	-I[<*file-name*>]	Write perfsys parameters to the file <*file-name*>.ini and exit
	-P	Read tested function names from the .ini file
	-n<*title-name*>	Set default title name for .ini file and output files
	-p<*dir-name*>	Set default directory for .ini file and input test data files
Select functions	-f <*or-pattern*>	Run tests for functions with pattern in their names, case sensitive

Group	Option	Description
	-f-<*not-pattern*>	Do not test functions with `pattern` in their names, case sensitive
	-f+<*and-pattern*>	Run tests only for functions with `pattern` in their names, case sensitive
	-f=<*eq-pattern*>	Run tests for functions with `pattern` full name
	-t[-|+|=]<*pattern*>	Run (do not run) tests with `pattern` in test name
	-m[-|+|=]<*pattern*>	Run (do not run) tests registered in file with `pattern` in file name
Help	-h	Display short help and exit
	-hh	Display extended help and exit
	-h<*key*>	Display extended help for the key and exit

Appendix B: DEPRECATED. Intel® IPP Threading and OpenMP* Support

B

All Intel® Integrated Performance Primitives functions are thread-safe. They support multithreading in both dynamic and static libraries and can be used in multi-threaded applications. However, if an application has its own threading model or if other threaded applications are expected to run at the same time on the system, it is strongly recommended to use non-threaded/single-threaded libraries.

Multi-threaded static and dynamic libraries are available as a separate download to support legacy applications. For new development, please use the single-threaded versions with application-level threading. One way to add application-level threading is to use Intel® Threading Building Blocks (Intel® TBB).

Some Intel IPP functions contain OpenMP* code, which increases performance on multi-processor and multi-core systems. These functions include color conversion, filtering, convolution, cryptography, cross-correlation, matrix computation, square distance, and bit reduction.

To see the list of all threaded APIs, refer to the *ThreadedFunctionsList.txt* file located in the documentation directory of the Intel IPP installation.

NOTE
Internally threaded (multi-threaded) versions of Intel® IPP libraries are deprecated but made available for legacy applications. It is strongly recommended to use single-threaded version of the library.

Setting Number of Threads

By default, the number of threads for Intel IPP threaded libraries follows the OpenMP* default, which is the number of logical processors visible to the operating system. If the value of the `OMP_NUM_THREADS` environment variable is less than the number of processors, then the number of threads for Intel IPP threaded libraries equals the value of the `OMP_NUM_THREADS` environment variable.

To configure the number of threads used by Intel IPP internally, at the very beginning of an application call the `ippSetNumThreads(n)` function, where `n` is the desired number of threads (1, ...). To disable internal parallelization, call the `ippSetNumThreads(1)` function.

Getting Information on Number of Threads

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

Optimization Notice
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Using a Shared L2 Cache

Several functions in the signal processing domain are threaded on two threads intended for the Intel(R) Core™ 2 processor family, and make use of the merged L2 cache. These functions (single and double precision `FFT`, `Div, and Sqrt`) achieve the maximum performance if both two threads are executed on the same die. In this case, the threads work on the same shared L2 cache. For processors with two cores on the die, this condition is satisfied automatically. For processors with more than two cores, set the following OpenMP* environmental variable to avoid performance degradation:

```
KMP_AFFINITY=compact
```

Optimization Notice
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

Avoiding Nested Parallelization

Nested parallelization may occur if you use a threaded Intel IPP function in a multithreaded application. Nested parallelization may cause performance degradation because of thread oversubscription.

For applications that use OpenMP threading, nested threading is disabled by default, so this is not an issue.

However, if your application uses threading created by a tool other than OpenMP*, you must disable multi-threading in the threaded Intel IPP function to avoid this issue.

Disabling Multi-threading (Recommended)

The best option to disable multi-threading is to link your application with the Intel® IPP single-threaded (non-threaded) libraries included in the default package and sdiscontinue use of the separately downloaded multi-threaded versions.

You may also call the `ippSetNumThreads` function with parameter `1`, but this method may still incur some OpenMP* overhead.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Index