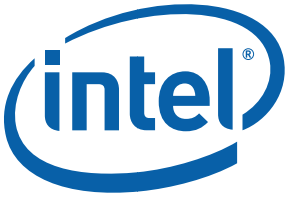

Intel® Energy Checker

Software Developer Kit User Guide

Revision 2.0

December 15, 2010

Revision History

Revision	Revision History	Authors / Contributors	Date
1.0	Initial Release	Jamel Tayeb Kevin Bross	2009/10/12
2.0	2010.12.15 Release	Jamel Tayeb Chang S. Bae Cong Li Stevan Rogers	2010/12/15

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.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

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.

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 by visiting Intel's Web Site.

Intel, the Intel logo, are trademarks or registered trademarks of Intel Corporation in the United States and other countries.

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

Copyright © 2009, 2010 Intel Corporation. All rights reserved.

Contents

Figures

Tables

1 Intel® Energy Checker Components

1.1 Introduction

The Intel® Energy Checker Software Developer Kit (Intel® EC SDK) provides tools to help developers design energy-efficient applications. Using the resources in the Intel EC SDK, programmers can monitor and analyze the effects their code has on the host platform to better control how the application affects energy efficiency of the host. The benefits can extend not only to the energy efficiency of a single host, but across an entire large data center.

This *Intel® Energy Checker Software Developer Kit User Guide* (*Intel® EC SDK User Guide* or *User Guide*) describes the Intel EC SDK and supporting documents. The Intel® Energy Checker is also referred to as the Intel® EC. The Intel EC SDK is also referred to as the SDK. The Intel® EC SDK Application Programming Interface (API) is also referred to as the API.

This *User Guide* covers the following topics:

- Components – brief descriptions of the parts of what is included in the Intel EC SDK
- Overview – a description of the basic concepts for using the Intel EC SDK
- API – a detailed descriptions of the commands in the Intel EC SDK API
- Installation – instructions on how to install the Intel EC SDK
- Code building – suggestions on how to compile your code
- Scripting tools – descriptions of useful scripting tools provided with the Intel EC SDK
- Windows* interoperability – descriptions of tools for working with Windows*
- Ganglia* interoperability – how to work with Ganglia
- ESRV and TSRV – suggestions on working with the energy and temperature servers provided with the Intel EC SDK
- Code sample – listing of sample codes
- Applications – examples of how the Intel EC can be applied in different use cases
- Appendix – helpful information not covered elsewhere in this *User Guide*

1.2 Components Summary

The Intel EC SDK is composed of the following elements:

- Intel EC API
- Build Tools
- Scripting Tools
- Interoperability Tools
- Energy and Temperature Monitoring Tools
- Sample Application Code
- Intel EC SDK Companion Applications
- Three user guides: *Intel EC SDK User Guide (this guide)*, *Intel® Energy Checker Device Driver Kit User Guide*, *Intel® Energy Checker Companion Applications User Guide*

1.3 Intel® Energy Checker API

The Intel EC API provides functions for exporting counters to and importing counters from an application. A counter stores the number of times a particular event or process has occurred (see Section 2.3 for details).

- Exporting a counter means to make a counter and its value accessible to other applications.
- Importing a counter means to read a previously exported counter's value into an application.

Applications can export and import counters as needed.

The Intel EC SDK exports and imports counters through a *Productivity Link (PL)*. See Figure 1.

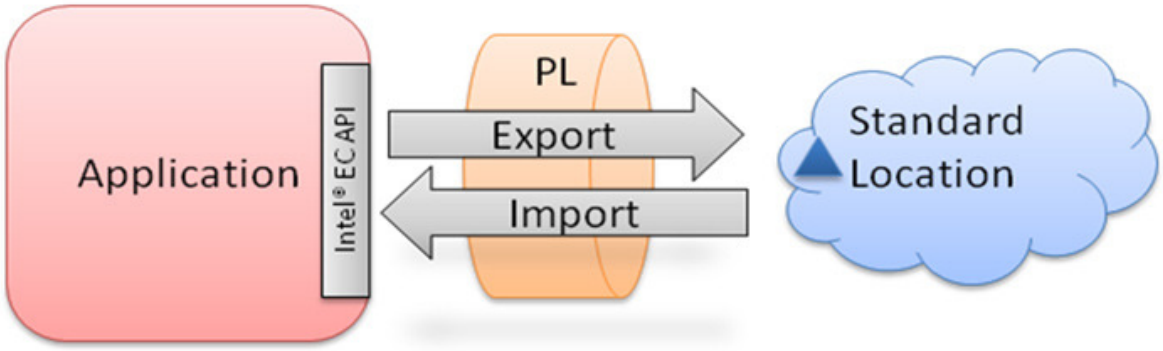

Figure 1: Using the Intel EC to import and export counters

The core Intel EC API is implemented in two files (`productivity_link.h` and `productivity_link.c`). The Intel EC SDK provides the source code. You can

review the code internals and adjust certain limits in the header file, but license is not given to use modified code externally or in a production environment.

NOTE

*Any application instrumented with the API and either distributed externally or used in a production environment **must use the unmodified version** of the API as shipped in the Intel EC SDK without changing any core elements of the API.*

NOTE

Please send any proposed changes to the API along with the rationale for those changes to Intel Corporation. If accepted, they may be incorporated in future versions of the API. Please submit feedback on the SDK to Intel Corporation via the site from which this SDK was downloaded.

NOTE

The Intel EC core API was designed to facilitate and standardize energy efficiency analysis and energy optimization of applications. However, the API can be used to expose any sort of counters and be used to monitor single systems up to complex data center configurations.

1.4 Build Tools

The Intel EC SDK includes makefiles and Microsoft Visual Studio* solution files to aid compilation of the source code provided with the Intel EC SDK.

1.5 Scripting Tools

In some cases, modifying the source code for an application may not be an option or may not be desirable. In those cases, consider manipulating PL counter data from script files.

For example, the breadth of stream processing utilities in Linux may provide a relatively quick way to parse log files created by proprietary applications.

Even when source code access is available, it may be desirable to read and aggregate PL counter data from scripts rather than creating full applications to read/analyze this data.

For these instances, the Intel EC SDK provides command-line utilities that can be called from batch files (Windows) or script files (other OS types).

The SDK offers the following scripting tools:

- pl_open
- pl_read
- pl_write
- pl_desc

Refer to Section 6 for more details on the scripting tools.

1.6 Interoperability Tools

The Interoperability Tools provide conversion facilities between various native counters and PL counters. The Windows* interoperability tools allow the user to dynamically convert Windows native counters into Intel EC counters and vice-versa.

These tools allow you to collect and export any existing registry counter available on the Microsoft Windows platform, as well as import any Intel EC counter with standard Windows monitoring and logging tools, such as Perfmon, Logman, or even the Intel VTune™ Performance Analyzer (with time-based sampling under Windows only).

The Intel EC SDK offers the following Windows interoperability tools:

- pl2w
- w2pl
- wselect
- wremove

The Ganglia* interoperability tool (pl2ganglia) allows you to dynamically convert PL counters into Intel EC counters.

Refer to Section 7 for more details on the scripting tools.

NOTE

Using the Windows interoperability tools and a standard transport mechanism — such as a networked file system — it is possible to monitor Windows-native counters on a Solaris 10-based system, a MacOS* X-based system, Linux*-based system or a MeeGo based system. Similarly, a Linux-based system can monitor a Windows-based system with these utilities. See Section 7 for an example.*

1.7 Energy and Temperature Monitoring Tool

The ESRV (Energy Server) (part of the Intel EC SDK) monitors a platform's energy consumption of monitored or instrumented components.

ESRV provides counters for the cumulative energy consumed, as well as the immediate power draw from the power meter or power supply. More detailed information can be found in the *Intel® Energy Checker Device Driver Kit User Guide*. Section 9 demonstrates how to use ESRV counters to report energy efficiency directly from an application.

The TSRV (Temperature Server) (part of the Intel EC SDK) monitors ambient temperature and relative humidity. TSRV has similar integration capability as ESRV.

1.8 Sample Application Code

The Intel Energy Checker SDK is shipped with the source code of several simple code samples and several full-fledged applications. The SDK offers the following additional sample applets:

- threading
- esrv_client
- java_calling_sample
- csharp_calling_sample
- objective_calling_sample
- sample_logger
- linux_system_information utilities
- cpu_use_histogram
- Cluster Energy Efficiency
- pl_agent
- pl2ganglia
- mti_sample

Refer to Section 10 for more details on the scripting tools.

1.9 SDK Companion Applications

Two very useful utilities are provided with the Intel EC SDK and are described in the *Intel® Energy Checker SDK Companion Applications User Guide*:

- **PL GUI Monitor** provides a graphical tool to dynamically examine Productivity Link (PL) counters created by applications instrumented with the Intel EC SDK.
- **PL CSV Logger** is a console application that dumps PL counters to the console or to an output file, which can be imported into most spreadsheet applications for further analysis.

1.10 Supported Languages and Operating Systems

Table 1 and Table 2 summarize supported languages and operating systems.

Table 1: Intel Energy Checker supported programming languages

Languages	Method
C/C++	Core API C Interface
C#	Dynamic library and C# class
Objective-C	Core API Objective-C class
Java	JNI and Java class
PHP, Perl, and other scripting languages	Scripting tools, SWIG, and language-specific C interface facilities

Table 2: SDK content and supported OS matrix

● Supported
○ Not Supported
▨ Not Applicable
S Source Code
B Binary

		Distribution	Dependency	Windows*		Linux*		Solaris* 10		MacOS* X		MeeGo*	
				32-bit	64-bit	32-bit	64-bit	32-bit	64-bit	32-bit	64-bit	32-bit	64-bit
Core API	pl_open()	S	▨	●	●	●	●	●	●	●	●	●	●
	pl_close()	S	▨	●	●	●	●	●	●	●	●	●	●
	pl_read()	S	▨	●	●	●	●	●	●	●	●	●	●
	pl_write()	S	▨	●	●	●	●	●	●	●	●	●	●
	pl_attach()	S	▨	●	●	●	●	●	●	●	●	●	●
	API unit tests	S	▨	●	●	●	●	●	●	●	●	●	●
	API micro benchmarks	S	▨	●	○	●	○	●	○	●	○	●	○
	Java* interface	S	▨	●	●	●	●	●	●	●	●	●	●
	C# .NET* interface	S	▨	●	●	●	●	●	●	●	●	●	●
	Objective-C* interface	S	▨	▨	▨	▨	▨	▨	▨	●	●	▨	▨
	SDK code samples	S	▨	●	●	●	●	●	●	●	●	●	●
Build Tools	Project & makefile	S	▨	●	●	●	●	●	●	●	●	●	●
Scripting Tools	pl_open	B	▨	●	●	●	●	●	●	●	●	●	●
	pl_read	B	▨	●	●	●	●	●	●	●	●	●	●
	pl_write	B	▨	●	●	●	●	●	●	●	●	●	●
	ps_desc	B	▨	●	●	●	●	●	●	●	●	●	●
Windows* Interop Tools	w2pl	B	▨	●	●	▨	▨	▨	▨	▨	▨	▨	▨
	pl2w	B	1	●	●	▨	▨	▨	▨	▨	▨	▨	▨
	wremove	B	1	●	●	▨	▨	▨	▨	▨	▨	▨	▨
	Wselect	B	▨	●	●	▨	▨	▨	▨	▨	▨	▨	▨
Ganglia* Interop Tool	pl2ganglia	S	4	▨	▨	●	●	●	●	●	●	●	●
Energy Monitoring	Esrv	B	▨	●	●	●	●	●	●	●	●	●	●
	simulated device	B	▨	●	●	●	●	●	●	●	●	●	●
	CPU indexed simulqted device	B	▨	●	●	●	●	○	○	○	○	●	●
	DAQ simulated device	S	▨	●	●	●	●	●	●	●	●	●	●
	APC Switched Rack PDU AP7xxx	B	▨	●	●	●	●	●	●	●	●	●	●
	Yokogawa* WT210	B	▨	●	●	●	●	●	●	●	●	●	●
	Yokogawa WT230	B	▨	●	●	●	●	●	●	●	●	●	●
	Yokogawa WT500	B	2	●	○	○	○	○	○	○	○	○	○
	Yokogawa WT3000	B	2	●	○	○	○	○	○	○	○	○	○
	Yokogawa MW100	B	▨	●	●	●	●	●	●	●	●	●	●
	Extech* e810801	B	▨	●	●	●	●	●	●	●	●	●	●
	Zes Zimmer LMG95	B	▨	●	●	●	●	●	●	●	●	●	●
	Zes Zimmer LMG450	B	▨	●	●	●	●	●	●	●	●	●	●
	Zes Zimmer LMG500	B	▨	●	●	●	●	●	●	●	●	●	●
	Watts?up PRO	B	▨	●	●	●	●	○	○	●	●	●	●
	P3 International* P4400 + Adafruit Industries* Tweet-a-Watt Kit	B	▨	●	●	●	●	●	●	●	●	●	●
	command line interpreter	B	▨	●	●	●	●	●	●	●	●	●	●
	IPMI-based power sensors	B	3	●	●	●	●	●	●	●	●	●	●
Temperature & RH	tsrv	B	▨	●	●	●	●	●	●	●	●	●	●

Monitoring	Digi* Watchport/H	B		●	●	●	●	●	●	●	●	●	●
	command line interpreter	B		●	●	●	●	●	●	●	●	●	●
	IPMI-based temperature sensors	B	3	●	●	●	●	●	●	●	●	●	●
	tsrv simulated device	B		●	●	●	●	●	●	●	●	●	●
Companion Applications	pl_csv_logger	S		●	●	●	●	●	●	●	●	●	●
	pl_gui_monitor	B	5	●	●	●	●	●	●	●	●	●	●
Documentation	User guides in PDF format	B		●	●	●	●	●	●	●	●	●	●

1: Microsoft .NET

2: Device vendor drivers

3: IPMI Tool

4: Ganglia*

5: X11 suppport for non-Windows environments

This page intentionally blank.

2 Intel EC SDK Overview

2.1 Introduction

The Intel EC SDK enables Independent Software Vendors (ISV) to instrument their application source code to export and import counters. Although the initial intent of this SDK is to facilitate energy efficiency analysis and optimizations, it can be used to expose any counter meaningful to the ISV and its customers.

NOTE

This SDK enables source code instrumentation. This should not be confused with other instrumentation technologies such as binary instrumentation or code profiling.

2.2 Basic Concepts

2.2.1 Counters

For simplicity, think of a counter as a file that stores the number of times a particular event or process occurrs, similar to the way an odometer counts the number of miles (or kilometers) a vehicle has traveled. The specific storage mechanism of the counters is implementation-specific, is not exposed to the applications, and may change in the future.

From a software developer point of view, a counter is eight bytes of data (64 bits). This matches the `unsigned long long` C data type.

The core API provides the functions required for exporting and importing counters from an application.

2.2.2 Counter Export and Import

Exporting a counter means to make a counter and its value visible and accessible to other applications (Figure 2). This may be thought of as writing out the value of the counter.

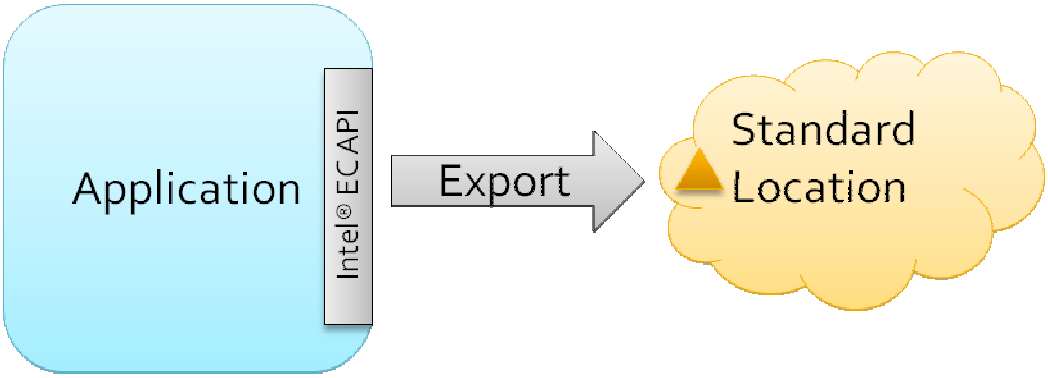

Figure 2: Exporting counters

Importing a counter means to read a previously exported counter's value into an application (Figure 3).

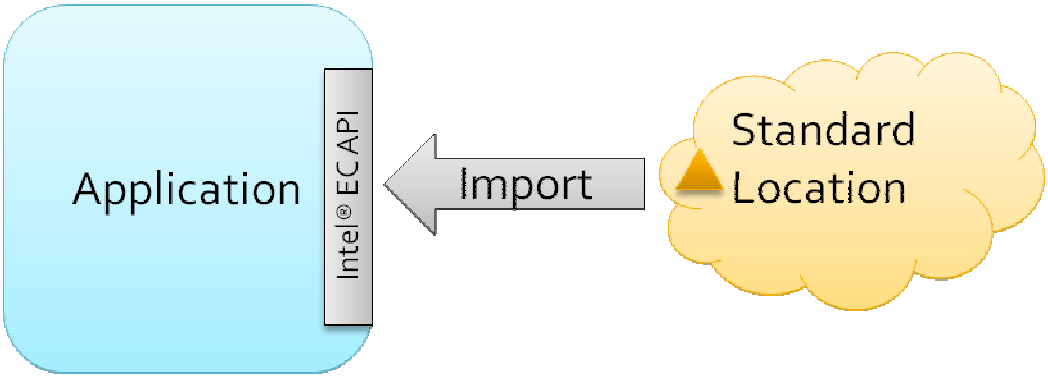

Figure 3: Importing counters

Applications can export their counters and import counters from other applications as needed (Figure 4).

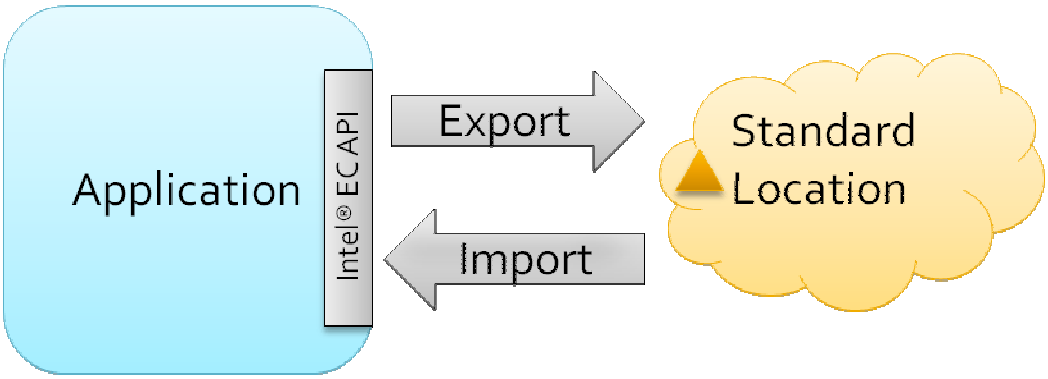

Figure 4: Importing and exporting counters

2.2.3 Productivity Links

A Productivity Link (PL) is a logical organization of a set of counters. Imagine a PL as a pipe through which counters are exported and imported (Figure 5). To use a PL to export and/or import counters, an application executes the following tasks:

1. Opens a PL.
2. Specifies the counters to be created and managed under the PL (see Section 3 for API details).
3. Uses the PL to export and/or import counters.
4. At the end of its run, it closes the PL.

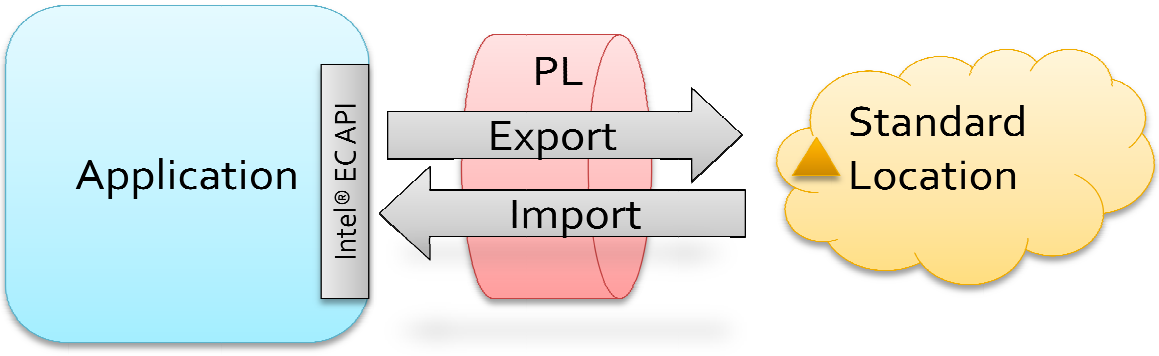

Figure 5: A Productivity Link

NOTE

The API automatically allows multiple instances of an application to maintain separate counters for each instance of the application.

An application can have multiple PLs if needed and each PL can have many counters (Figure 6). Review the Threading sample applet in Section 10.1 for possible — and recommended — implementations in a multi-threaded application.

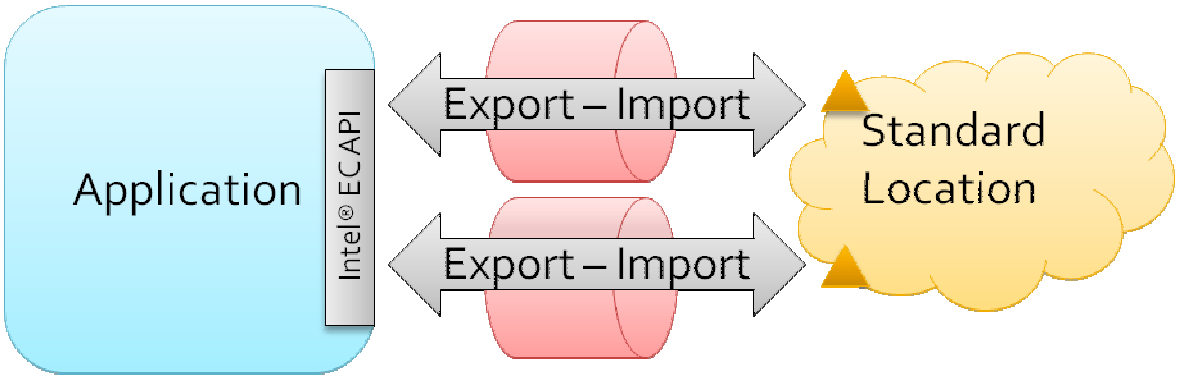

Figure 6: Multiple PL configuration

NOTE

By default, an application can simultaneously open ten (10) PLs, and each PL can have up to 512 counters, providing up to 5,120 counters. These limits can be easily modified on a per-application basis if required. However, most applications are expected to export no more than a few dozen counters. It is also possible to compile the API to support dynamically allocated counters as described in Section 0.

NOTE

Counters are persistent after the application exits. Therefore, if you prefer to re-use a PL on repeated instances of the application, you should reset the PL counters during application initialization. Persistency is only guarantied if __PL_FILESYSTEM_LESS__ symbol is not defined.

Finally, it is possible to export and import counters between multiple applications. Figure 7 depicts such a case. In this scheme, two applications are exporting their counters and one application (i.e., logger software) imports these counters to build a log file.

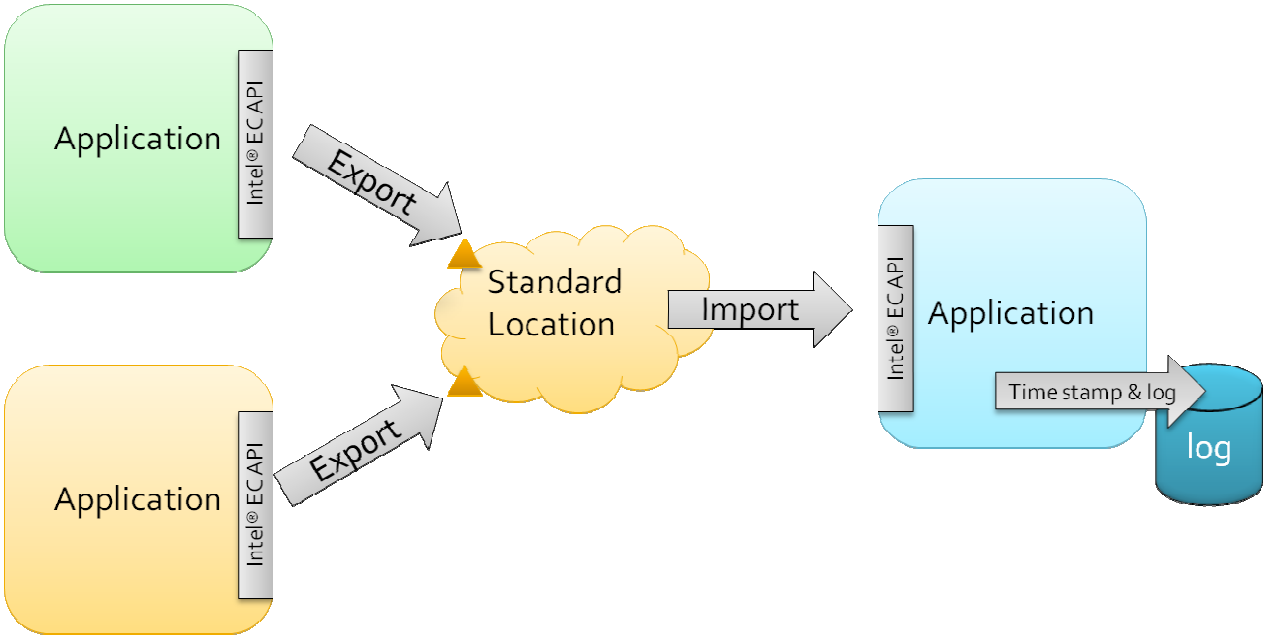

Figure 7: Multiple application exchanges

2.3 Counter Restrictions and Conventions

2.3.1 Storage

PL Counters store only numerical data.

Inside the application, a counter is an unsigned long long that is eight bytes (64 bits) of memory. Outside of the application, counters are stored in a manner determined by the Intel EC API (currently, they are stored in files). The API makes the outside nature of the counters transparent for the instrumented applications.

2.3.2 Naming

Even though the API allows for anonymous counters and applications, anonymous counters should never be used and counter names should be reasonably descriptive. There are no limitations in the names for counters, except:

- A counter name cannot contain a forbidden character for a file name (depending on the operating system). For maximum compatibility, application names and counter names should be comprised exclusively of characters from the following set:
  0-9, A-Z, a-z, '_', '-', '(', ')', '[', ']', '.' and the space character
- The length of the counter name plus the application name cannot exceed 199 characters.

NOTE

Applications are responsible for ensuring that counters meet the naming conventions described above. Therefore, the API and scripting tools don't check for these conditions. If the __PL_EXTRA_INPUT_CHECKS__ is defined, then these conditions are checked by the API and the system error is set to PL_INVALID_PARAMETERS.

2.3.3 Ranges

Counters can hold integer values up to 18,446,744,073,709,551,615 (2^64 -1). Most of the events counted can be stored in eight bytes of data. If one of the events being tracked can exceed this maximum number, additional counters can be used to count the overflows (like a carry).

For example, the ESRV utility contains an overflow counter for the energy measured in joules. This was done for demonstration purposes, as the *esrv --ranges* command shows (below) that it will be hard even for a huge data center to overflow this counter.

```
1
2   Display this message on supported data ranges.
3
4   Usage:  esrv --ranges
5
6      Max Joules.....................................184,467,440,737,095,516.15
7      Max kWh.........................................................51,240,955,760.30
8      Max kWh With Overflows..........945,228,797,002,527,336,667,005,373,644.80
9
10                  Time to Overflow                                  Rate
11     ____________________________________________________________________________
12     ..............5,849,424,173.55 years @.......................1 W
13     ..............584,942,417.36 years @.......................10 W
14     ..............58,494,241.74 years @.......................100 W
15     ..............5,849,424.17 years @...................1,000 W......1 kW
16     ..............584,942.42 years @..................10,000 W.....10 kW
17     ..............58,494.24 years @.................100,000 W....100 kW
18     ..............5,849.42 years @................1,000,000 W......1 MW
19     ..............584.94 years @..............10,000,000 W.....10 MW
20     ..............58.49 years @.............100,000,000 W....100 MW
21     ..............5.85 years @...........1,000,000,000 W......1 GW
22
```

To store non-integer values, fixed point notation is used with the decimals suffix as described in Section 2.3.4.2.

2.3.4 Suffix Counters

In many cases, it is desirable to provide additional information about how to interpret the counters. By following certain conventions for counter name suffixes, these counters can be self-describing and can be more easily used by analysis applications. Since these counters are used to describe the format or meaning of the counters to which they refer, they are static counters (except for the sign suffix) and are not expected to be written out more than once per application session.

Commonly used suffixes include the following:

- sign
- decimals
- offset
- offset.decimals
- offset.sign
- scalar
- scalar.decimals

2.3.4.1. Sign Suffix

For maximum portability across different architectures, the SDK always uses positive values for its counters. However, negative values can easily be represented with PL counters and interpreted by other applications if the following convention is adhered to:

1. Add a supplemental counter with a `.sign` suffix.
2. Write the static sign to the supplemental counter using the following values:
   1 means a negative number.
   0 means a positive number (or zero).

If the sign suffix is omitted, the number is assumed to be positive. For example, suppose a counter named `height` has a value of 3. If `height.sign` does not exist, or it has a value of zero (0), the composite value of the height counter is 3. However, if `height.sign` has a value of 1, then the composite value of the height counter is -3.

NOTE

The sign suffix counter should be created at the time the base counter is created, if it is expected to represent negative values (even if the current value is positive). If the sign suffix is not defined when the base counter is created, monitoring applications may assume that the value is never expected to be negative.

NOTE

The sign suffix is the only suffix counter whose value is expected to change dynamically.

2.3.4.2. Decimals Suffix

The SDK always uses `unsigned long long` values for its counters, but floating point values with a fixed number of decimal places can easily be represented with PL counters and interpreted by other applications, if the following convention is adhered to:

1. Add a supplemental counter with a `.decimals` suffix.
2. Write the static number of decimal places to the supplemental counter.

For example, the ESRV utility makes use of this with its `Energy (kWh)` counter.

To represent the value to two decimal places, the actual number of kiloWatt-hours is multiplied by 100 and stored as an `unsigned long long`. In addition, at program startup, ESRV writes a value of 2 to the `Energy (kWh).decimals` counter to indicate that `Energy (kWh)` has two decimal places.

All applications writing counters representing fixed decimal numbers should use a supplemental static `.decimals` suffix counter as appropriate.

2.3.4.3. Offset Suffix

In some cases, the native source of a counter may represent a differential value from some fixed offset value. Rather than adding the differential value to the offset value each time the counter is written out, you could elect to define a static offset counter and let any consumers of the PL data add in the offset where needed. To do this, the following convention has been established:

3. Add a supplemental counter with a `.offset` suffix.
4. Write the static value of the offset to the supplemental counter.

For example, if an application is counting the number of visitors over 5000 to a theme park, a `Visitor Total` value of 800 might represent a total attendance of 5800 visitors. By adding a `Visitor Total.offset` counter and writing a value of 5000 to that offset counter, you can add that amount to the counter value to get the real value.

2.3.4.4. Scalar Suffix

In most cases, the decimals suffix provides sufficient ability to scale the way numbers are reported out. However, there are some cases where the counter values written through PL need some sort of scaling factor applied to them. In such cases, the following convention should be used:

5. Add a supplemental counter with a `.scalar` suffix.
6. Write the static value of the scaling factor to the supplemental counter.

For example, software for an egg producer could report how many dozens of eggs were produced with an `Egg Dozens` counter. That same value could be written to an `Eggs` counter, provided that `Eggs.scalar` had been set up with a value of twelve.

2.3.4.5. Compound Suffixes

The following compound suffixes are commonly recognized:

- offset.decimals
- offset.sign
- scalar.decimals

For example, consider the following:

MyTotal counter has a value of 5
MyTotal.scalar has a value of 72
MyTotal.scalar.decimals has a value of 1

In this example, the real scalar value is 7.2 and the actual value is 36 (5 * 7.2). Whenever compound static counters are used, `.decimals` must always be to the right (if used) and `.scalar` must always be to the left of the other static counters if used.

In C notation, the real total for a `total` counter can be figured as follows:

```
real_total =  total * (total.sign ? -1 : 1) / (10 ^ total.decimals)
       * total.scalar / (10 ^ total.scalar.decimals)
       + total.offset / (10 ^ total.offset.decimals) * (total.offset.sign ? -1 : 1);
```

NOTE

The offset.sign suffix counter is not expected to change dynamically.

2.3.5 Update Frequency

Applications can use an update frequency that makes sense for that specific application. This frequency can be different for each counter. Some counters will be written only once (i.e., the suffix counters described in Section 0). Other counters will be written at a regular pace. For example, the energy counters of ESRV are updated every second. An update rate higher than once per second is not recommended.

If the granularity of a counter is small, it may be unrealistic to update the associated counter for each counter increment. For example, if the application counts the number of bytes received through a LAN port, then it would be impractical to update the counter for each byte. In this case, updating the counter once every packet or every 100 packets may be a more reasonable approach. The decision of how often to update PL counters could be handled by a **collector** or metrics manager thread as indicated in Section 10.1.

NOTE

The API integrates a write cache that cancels write operations if the value to be written is the same as the last one. However, it is a good practice to avoid writing the same value again and again.

2.3.6 Software Status Counter

Another recommended practice is to define a software status counter (`Status`) to indicate when an application is in a given state. For example, an application may use two status values, one to indicate when the application is active and when it is idle. Other applications may have multiple states that can be represented by the same `Status` counter: 0 = terminated, 1 = idle, 2 = initializing, 3 = active, 4= terminating.

The actual meaning of the `Status` counter is application-specific. However, the presence of the `Status` counter can be very useful in performing benchmarking, in determining system idle power consumption overhead, and in related analysis activities. For example, it enables the user to factor out some phases of the algorithm or, conversely, to focus on a specific phase.

2.3.7 Metrics

Counters are the key elements in developing metrics. Complex metrics can be built from combinations of counters. An example of a compound metric derived from counters is the amount of work per unit of energy consumed.

To use a car analogy, tracking the amount of fuel consumed and the distance traveled can produce a compound metric, commonly expressed as miles per gallon or kilometers per liter.

2.3.7.1. Example Graphics Metrics

If the target is to perform energy efficiency analysis and/or reporting, a minimum of two counters is recommended. The first counter is the amount of useful work done by the software, and the second is the amount of energy consumed while performing the useful work.

Many applications today have some counters for internal or external indications of the application performance. If the application already measures its performance, then it should be straightforward to collect and export this data. Indeed, performance is usually computed as the amount of work divided by the time required to do the work.

For example, Figure 8 below shows the POV-Ray* (Persistence of Vision Ray Tracer) image renderer calculating the number of pixels per second (pps) rendered during the software's active mode as indicated by the blue line.

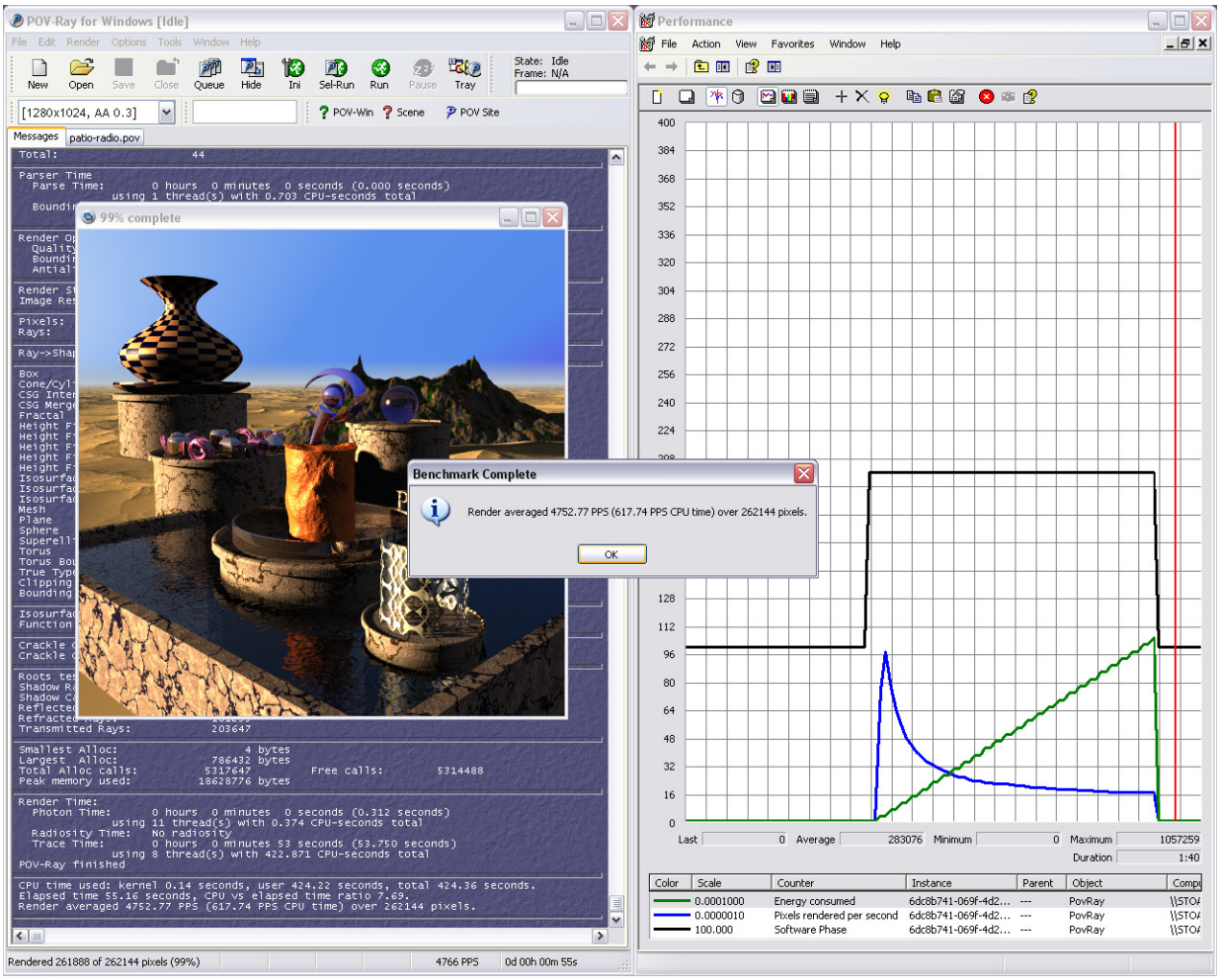

Figure 8: Software state counter example

As noted in section 2.3.6 above, it is often useful to understand the application's software state. Figure 8 shows the trace over time of the state variable (black square curve in the Perfmon window). Using such a variable is of benefit when analyzing the data.

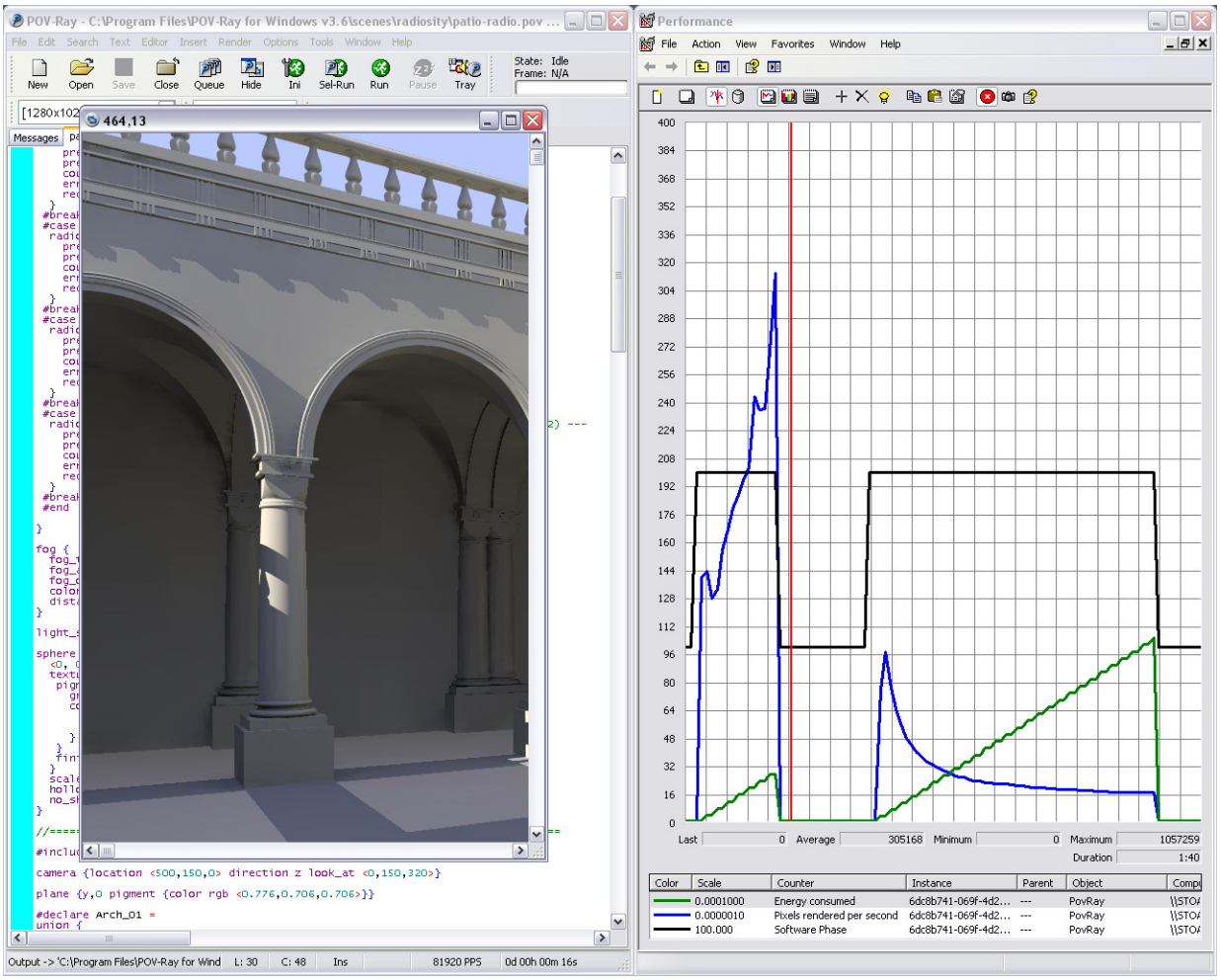

Figure 9: Software state counter tracked in Perfmon*

Figure 9 demonstrates the benefit of the software state. Based on the sampling frequency of the monitoring/logging tool used to analyze the data, all of the software phases may not be accurately exported. With Perfmon (as shown in the figure), a software state change under one second will likely be missed. With POV-Ray, this may depend on the complexity of the image to render. Most images require many minutes, up to multiple days. If transitions are too short to be properly captured, consider the use of a higher granularity software state, extend the test period to make sub-second transitions less relevant, or embrace a statistical analysis of software states.

The SDK enables the analysis of system energy consumption to be correlated with useful work and software state information, as illustrated in Figure 10 below.

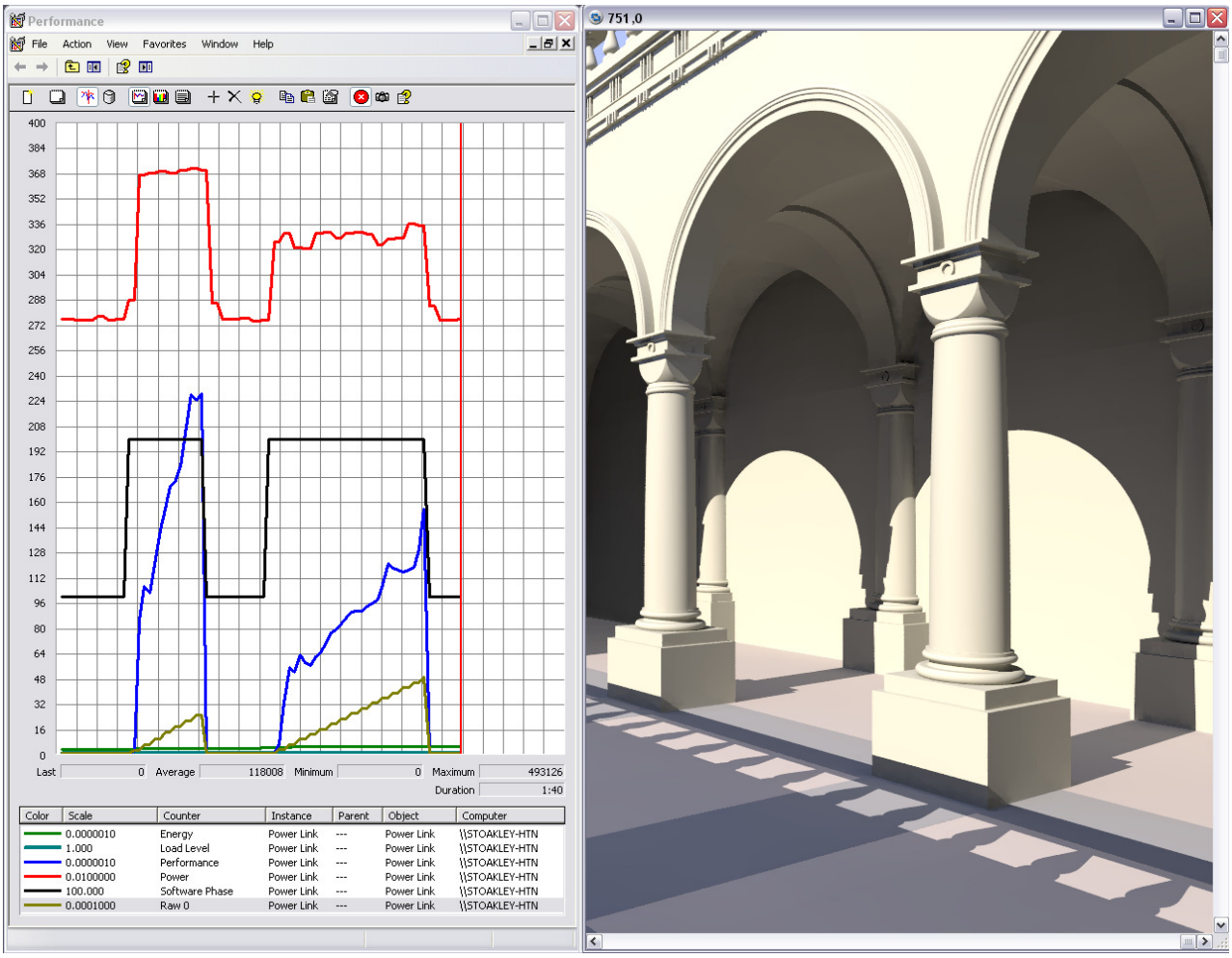

Figure 10: Power, energy and performance counters in Perfmon

2.4 Three Use Cases for Intel® EC SDK

There are many possible use cases for the Intel® Energy Checker (Intel® EC) API, but the three of the most common usages are as follows:

- Lab usage
- Production-level data center historical trending
- Real-time dynamic management

Each of these use cases is described below, along with an iterative optimization process.

NOTE

Section 11 presents the use of the SDK in the scope of several proofs of concepts, demonstrating techniques to save energy without compromising performance.

2.4.1 Lab Usage

The most common use for PL counters is to evaluate application/system performance in a lab environment. Using applications instrumented with Intel EC, engineers can evaluate the impact of changes in elements like the following:

- Hardware (i.e., adding/removing DIMMs)
- Hardware settings (i.e., maximum processor frequency)
- Software algorithms or libraries (which algorithm runs faster?)

In many cases, the value of the Intel EC-instrumented data increases greatly when data from multiple sources can be correlated. One particularly appealing use case is when application performance is correlated with system energy consumption.

The Intel EC API was originally conceived to carry out energy efficiency analysis of software. The rationale was that software drives more or less the underlying hardware's energy consumption. Therefore, it is possible to optimize software to improve its energy efficiency and reduce the platform's energy consumption. This activity is an additional step software engineers can take to provide energy-aware, energy-efficient computer systems and applications to their customers.

To measure software energy efficiency, one needs two distinct sets of data: the amount of **useful work** done by the application to analyze and the amount of **energy consumed** by the system to run the application.

For example, to study the energy efficiency of image rendering software like POV-Ray*, one measure of the useful work could be the number of pixels rendered. If the energy consumed by the system during the rendering process is captured by the image rendering software or by a separate application, it is simple to derive the two following energy efficiency metrics:

- # of pixel(s) rendered per joule consumed
- # of joule(s) consumed by the system per pixel rendered

NOTE

One joule equals one watt-second.

Figure 11 depicts an application that imports energy counters to compute and export its own energy efficiency counters. It can also adapt its behavior according to the energy status of the system. This may require some additional code to implement an energy-aware heuristic. This decision-making could be incorporated into the application itself or it could be offloaded to a middleware stack, using the energy efficiency metrics exported by the application.

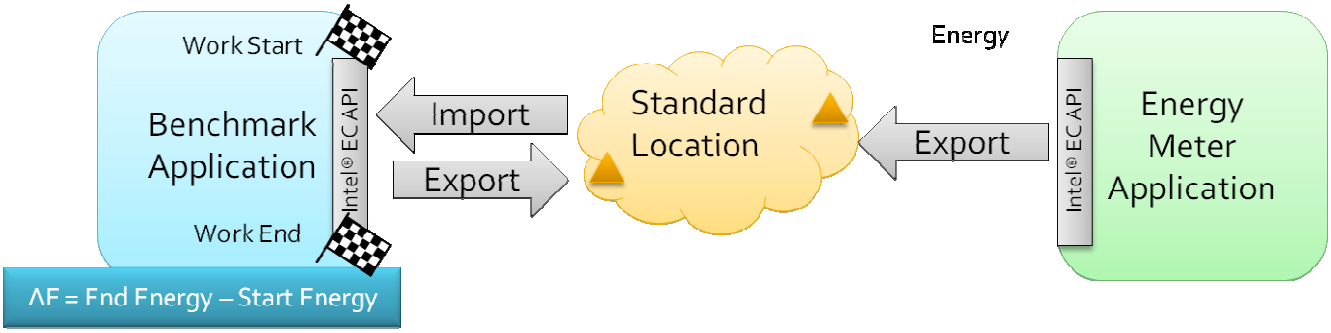

Figure 11: Example energy-aware application

Also note that the application does not have to import energy counters and export energy efficiency counters. This can very well be offloaded to a logging application as shown in Figure 7 or offloaded to the previously referenced middleware.

The PL GUI Monitor application (described in the *Intel® Energy Checker SDK Companion Applications User Guide*) provides a graphical interface for monitoring PL counters that can be very useful in a lab environment.

2.4.2 Production-level Data Center Long-term Trending

There is an engineering maxim that *you can't manage what you can't measure*, and this is true for managers of data centers and other IT facilities. Those managers are often asked to justify the needs for additional hardware/software purchases and to explain facility power costs that have risen over time. Unfortunately, it is often difficult to gather real-world data from production facilities.

For applications where logging information is available, it is often stored in proprietary formats that are only accessible via custom tools that can't easily be correlated with other elements in the system. In some cases, the tools supplied with the application cannot be easily automated and may require special access rights to the application for which the administrator does not have permission. For example, a facility manager may not have access to a payroll database, but

may still want to have information about how much work that payroll application completes month to month.

By instrumenting applications with Intel EC, data center administrators can monitor key measures of useful work over intervals that make the most sense. For example, a mail server could be instrumented to report out the number of mail messages sent and the number of kilobytes in mail messages sent. These counters could then be reported out hourly, daily, weekly, monthly, quarterly, or at whatever interval the administrator desires.

The Intel EC core API itself is very light-weight; its load should have negligible impact on most applications. The micro-benchmarks described in Section 0 can be used to gauge performance impact of the API. The PL CSV Logger application (described in the *Intel Energy Checker SDK Companion Applications User Guide*) provides a simple way to log time-stamped PL counter data to files that can be imported into spreadsheets or other analysis tools.

2.4.3 Real-time Dynamic Management

Taking the production-level trending concept one step further, you can use Intel EC to manage applications in real-time. For example, if a service is provided on a cluster of servers and aggregated PL data from multiple servers indicates that there is little utilization of the cluster at the current time, management software could decide to quiesce extra servers to save power. Alternatively, the management software could choose to reduce the frequency of processors in those servers.

In many cases, the same metrics used for long-term trending can be used for dynamic management of servers in real time.

NOTE

The Intel Energy Checker SDK Device Driver Kit User Guide *provides information on how to perform energy measurements, including how to measure power draw without interrupting the operation of servers.*

2.4.4 Iterative Optimization

Application developers can follow a traditional iterative optimization process that consists of first measuring baseline data and then changing components one at a time to gather comparative data. Incremental improvements can be tested and verified to optimize the system in an iterative fashion. When the targeted energy efficiency level is reached, the process can stop.

Figure 12 shows the output of an instrumented version of the POV-Ray renderer exporting several counters and monitored via the Windows Perfmon (Performance Monitor) utility.

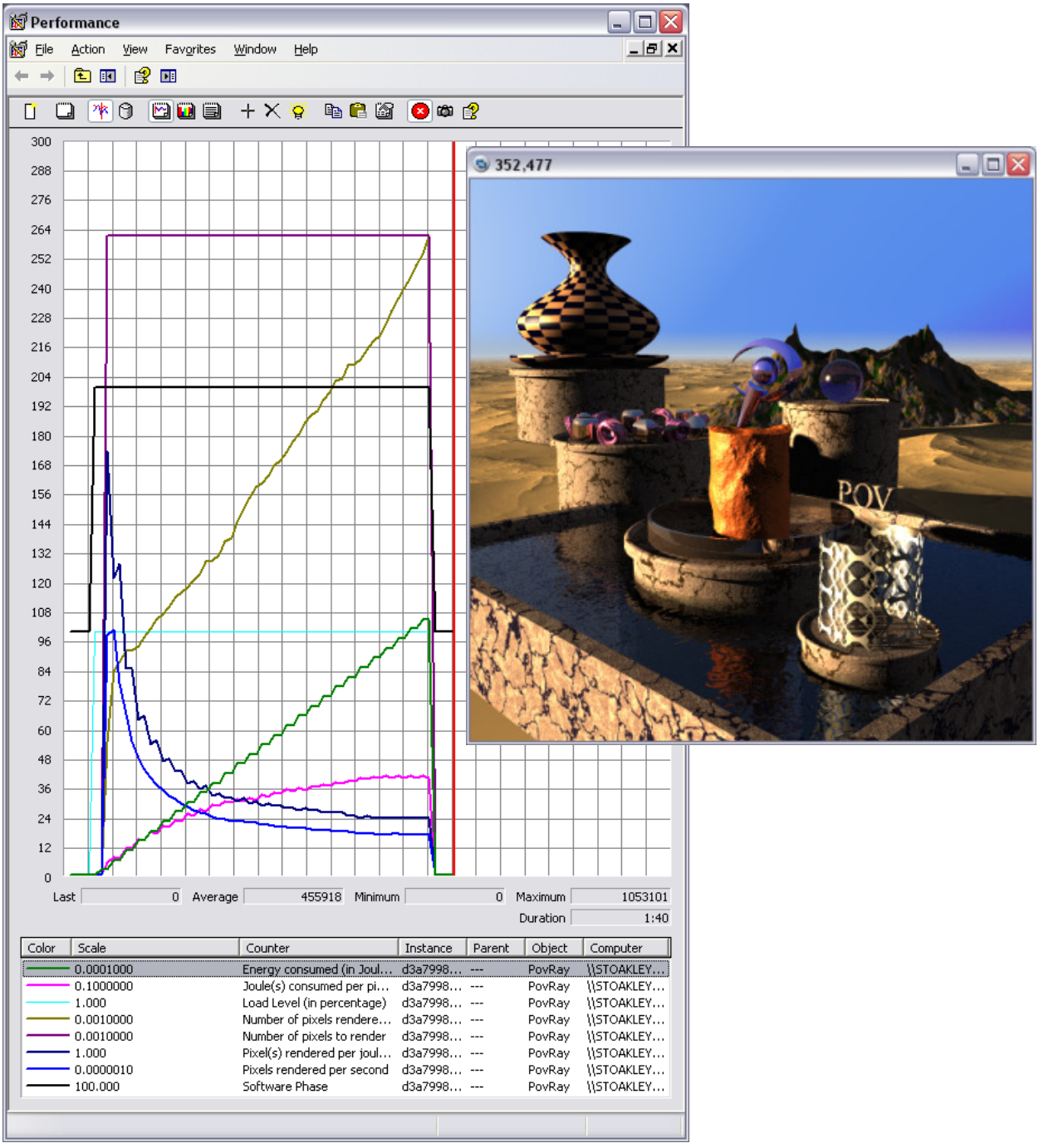

Figure 12: Instrumented POV-Ray* data monitored with Perfmon*

Figure 13 below shows a different image rendered with POV-Ray and monitored with the PL GUI Monitor application provided with the Intel EC SDK (refer to the *Intel Energy Checker SDK Companion Applications User Guide* for more information on the PL GUI Monitor).

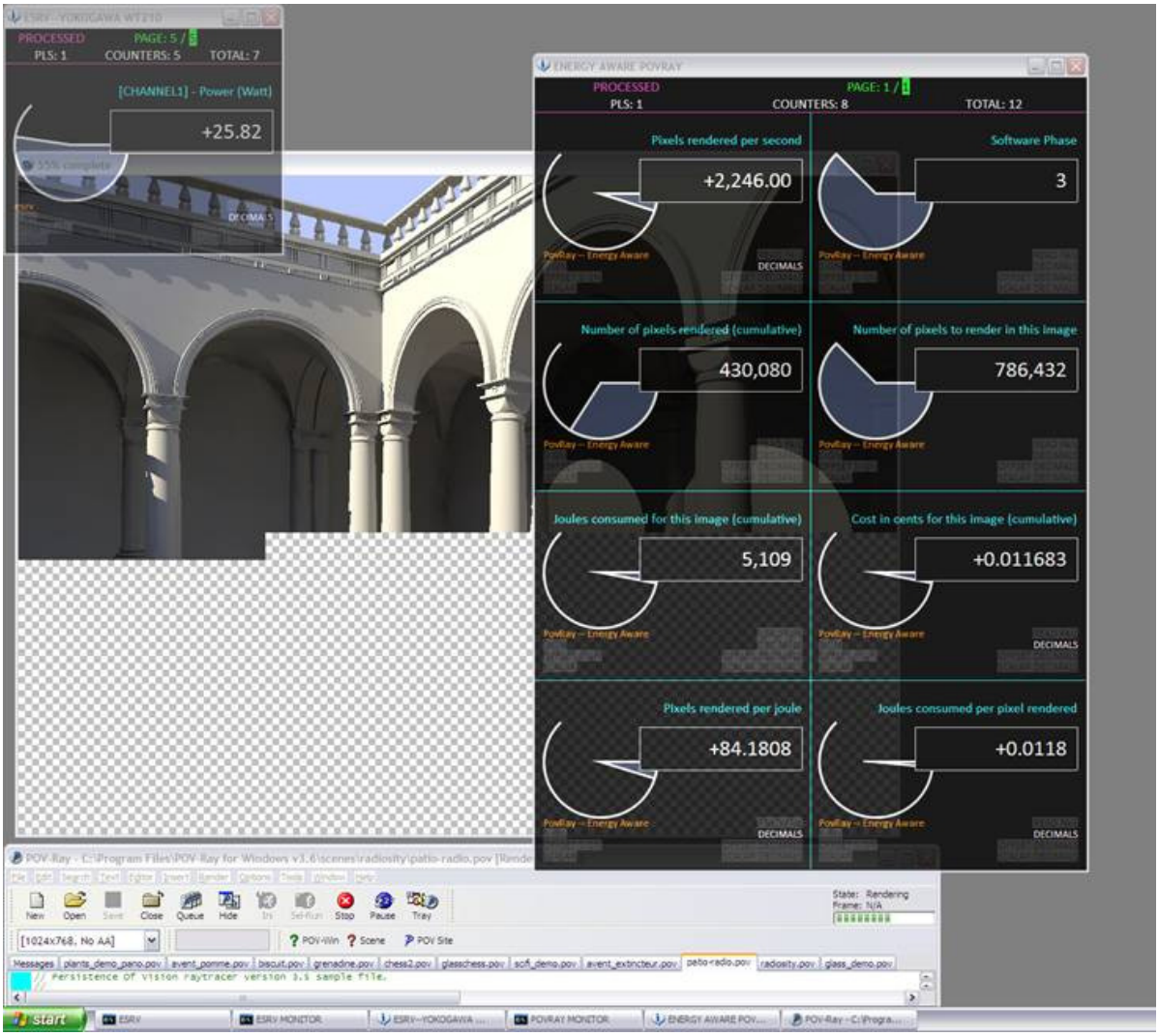

Figure 13: Instrumented POV-Ray* data monitored with PL GUI Monitor

⚠ CAUTION

A fundamental expectation is that the amount of work performed—including the quality of the work produced—during each measurement is kept constant for the energy efficiency analysis or that the amount of useful work performed is otherwise normalized. This allows direct comparison of the energy efficiency metrics between various experimental setups.

3 Intel® Energy Checker API

The Intel EC API offers a series of routines that can be integrated into any application to write out counters in a standard manner. In the scope of this document, adding calls to the API is called instrumenting the code. This should not be confused with binary instrumentation as it is performed by PIN (http://www.pintool.org/) or the Intel VTune™ Performance Analyzer.

3.1 API Quick Overview

The API code is provided as a set of two C source code files (`productivity_link.c` and `productivity_link.h`). Therefore, no external libraries or run-time software is required with the instrumented application; this allows Intel EC-instrumented applications standalone, without imposing any additional library dependencies. Alternatively, Intel EC code can be built into Dynamic Link Libraries (DLLs) or Shared Objects (SOs) to provide dynamic linkage at runtime.

The five functions of the API are:

- pl_open()
- pl_close()
- pl_write()
- pl_read()
- pl_attach()

As a minimal instrumentation, the code has to

7. Create a PL (one call to `pl_open()`, generally performed at initialization time).
8. Expose and update at least one counter (one call to `pl_write()` each time the counter has to be updated).
9. Close the PL previously opened (one call to `pl_close()`, generally at application shutdown).

The `pl_open()` description documentation page has sample code in C language showing how to do this.

To call the API from another language/environment, use the C linkage interface as directed by that programming language. Sections 0 and 0 demonstrate the API use from Java* and .NET* with C#.

3.2 Counter Storage

The location where PL counter data is maintained is different on different types of systems. Refer to Section 4.1 for more information on storage locations.

3.3 Application and Counter Names

None of the application names and counter names provided to Intel EC API calls are allowed to contain invalid characters for a file name (under the target operating system). Refer to Section 2.3.2 for more information on counter naming restrictions.

3.4 pl_open()

Creates a Productivity Link (PL) and defines a set of counters in the PL.

3.4.1 Syntax

```
int pl_open(
  char *application_name,
  unsigned int counter_count,
  const char *counter_names[],
  uuid_t *uuid
);
```

3.4.2 Parameters

application_name

Pointer to a zero terminated ASCII string

counter_count

Number of counters to create

counter_names

Array of pointers to zero terminated ASCII strings

uuid

Pointer to a uuid

3.4.3 Description

This function is declared in the `productivity_link.h` file. The function opens a PL and creates *counter_count* counters as specified by the *counter_names* array's entries. The function returns the PL descriptor to be used by all subsequent operations on this PL. The function also writes the uuid associated with this PL into the memory location pointed to by *uuid*. It is the caller's responsibility to ensure that the memory location has enough space to hold the uuid (`sizeof(uuid_t)`).

Though not recommended, the *application_name* and the *counter_names* array entries can be NULL. If *application_name* is NULL, then the "*anonymous_application*" string is used instead. If any of the *counter_names* array's entry is NULL, then the "*anonymous_counter_"* string is used instead. For an anonymous counter, its rank is appended to the "*anonymous_counter_*" string.

For example, if *counter_names[0]* is NULL, then "*anonymous_counter_1"* is used. If *counter_names[41]* is NULL, then "*anonymous_counter_42"* is used, and so on. However, for the sake of clarity, the use of anonymous counters is highly discouraged.

The uuid is returned to the caller for information purposes only. The instrumented application does not need the uuid to use the PL, unless it wants to

directly share its PL handle with other applications. In such cases, it is often useful to represent the uuid as a string. The following code shows how to use the uuid and convert it into a string.

```
#ifdef __PL_WINDOWS__
   #include <windows.h>
   #include <tchar.h>
#endif // __PL_WINDOWS__
#if defined (__PL_LINUX__) || (__PL_SOLARIS__) || (__PL_MACOSX__)
   #include <string.h>
   #include <unistd.h>
   #include <errno.h>
   #include <uuid/uuid.h>
#endif // __PL_LINUX__ || __PL_SOLARIS__ || __PL_MACOSX__

#include "productivity_link.h"

int main(void) {

   uuid_t uuid;
   int pld = PL_INVALID_DESCRIPTOR;

#ifdef __PL_WINDOWS__
   char *p = NULL;
#endif // __PL_WINDOWS__
#if defined (__PL_LINUX__) || (__PL_SOLARIS__) || (__PL_MACOSX__)
   char buffer[MAX_BUFFER_SIZE];
#endif // __PL_LINUX__ || __PL_SOLARIS__ || __PL_MACOSX__
   pld = pl_open(
      APPLICATION_NAME_STRING,
      MAX_COUNTERS,
      my_counters,
      &uuid
   );
   assert(pld != PL_INVALID_DESCRIPTOR);
#ifdef __PL_WINDOWS__
   UuidToString(
      &uuid,
      (RPC_WSTR *)&p
   );
#endif // __PL_WINDOWS__
#if defined (__PL_LINUX__) || (__PL_SOLARIS__) || (__PL_MACOSX__)
   memset(
      buffer,
      0,
      sizeof(buffer)
   );
   uuid_unparse(
      uuid,
```

```
46          buffer
47      );
48  #endif // __PL_LINUX__ || __PL_SOLARIS__ || __PL_MACOSX__
```

3.4.4 Sample Code

```
1   #include <stdio.h>
2   #include <windows.h> // only non-PL related system calls are OS specific
3   #include <assert.h>
4   #include "productivity_link.h"
5
6   //--------------------------------------------------------------------------
7   // defines
8   //--------------------------------------------------------------------------
9   #define UPDATE_INTERVAL_IN_MS 1000 // 1 second
10  #define APPLICATION_NAME "My_Application"
11  #define COUNTERS_COUNT 2
12  #define COUNTERS_NAMES {
13      "Frames",
14      "Pixels"
15  }
16  enum COUNTERS {
17      FRAMES = 0,
18      PIXELS
19  };
20
21  //--------------------------------------------------------------------------
22  // function prototype
23  //--------------------------------------------------------------------------
24  BOOL signal_handler(DWORD);
25
26  //--------------------------------------------------------------------------
27  // program global -- for clarity only
28  //--------------------------------------------------------------------------
29  int pld = PL_INVALID_DESCRIPTOR;
30
31  //--------------------------------------------------------------------------
32  // program entry point
33  //--------------------------------------------------------------------------
34  int main(void) {
35
36      PL_STATUS ret = PL_FAILURE;
37      uuid_t uuid;
38      BOOL bret = FALSE;
39
40      char *counters[MAX_COUNTERS] = COUNTERS_NAMES;
41
42      unsigned long long frames = 0;
```

```
43      unsigned long long pixels = 0;
44
45      //---------------------------------------------------------------------
46      // install the event handler routine
47      //---------------------------------------------------------------------
48      bret = SetConsoleCtrlHandler(
49          (PHANDLER_ROUTINE)signal_handler,
50          TRUE
51      );
52       assert(bret);
53
54      //---------------------------------------------------------------------
55      // open a Productivity Link
56      //---------------------------------------------------------------------
57      pld = pl_open(
58          APPLICATION_NAME,
59          COUNTERS_COUNT,
60          counters,
61          &uuid
62      );
63      assert(pld != PL_INVALID_DESCRIPTOR);
64
65      //---------------------------------------------------------------------
66      // direction on how to stop monitoring.
67      //---------------------------------------------------------------------
68      fprintf(
69          stdout,
70          "Type <CTRL>+<C> to stop.\n"
71      );
72
73      //---------------------------------------------------------------------
74      // loop until interrupted by user
75      //---------------------------------------------------------------------
76      while(TRUE) {
77          Sleep(UPDATE INTERVAL_IN_MS);
78
79          //-----------------------------------------------------------------
80          // read meter data
81          //-----------------------------------------------------------------
82          frames = get_encoded_frames_count ();
83          pixels = Frames * pixels_per_frame;
84
85          //-----------------------------------------------------------------
86          // write updated counter in the Productivity Link
87          //-----------------------------------------------------------------
88          ret = pl_write(
89              pld,
90              &frames,
91              FRAMES
```

```
92          );
93          assert(ret == PL_SUCCESS);
94          ret = pl_write(
95              pld,
96              &pixels,
97              PIXELS
98          );
99          assert(ret == PL_SUCCESS);
100     }
101
102     //---------------------------------------------------------------------
103     // housekeeping is done in the event controller
104     //---------------------------------------------------------------------
105     return(0);
106 }
107
108 //---------------------------------------------------------------------
109 // event handler
110 //---------------------------------------------------------------------
111 BOOL signal_handler(DWORD c) {
112
113     PL_STATUS ret = 0;
114
115     switch(c) {
116
117         case CTRL_C_EVENT:
118
119             //-----------------------------------------------------------
120             // process user requested abort
121             //-----------------------------------------------------------
122             fprintf(stdout, "Stopping...\n");
123
124             //-----------------------------------------------------------
125             // close Productivity Link -- global pld comes handy here
126             //-----------------------------------------------------------
127             ret = pl_close(pld);
128             assert(ret == PL_SUCCESS);
129             return(FALSE);
130
131          default:
132             return(FALSE);
133      }
134 }
```

3.4.5 Return Values

PL_INVALID_DESCRIPTOR

Indicates an error condition. If PL_INVALID_DESCRIPTOR is returned, then the system's last error code is set as described in the Error Codes and Internal Error Codes section. Any other return value is a valid PL descriptor.

3.4.6 Error Codes

PL_BYPASSED

The call to `pl_open()` was bypassed. This happens when the instrumentation is de-activated at compilation time. This is performed by defining the __PL_BYPASS__ symbol.

PL_INVALID_PARAMETERS

At least one argument provided is invalid. This happens if the mandatory pointers are NULL, or if the number of counters is lower than 1 or greater than PL_MAX_COUNTERS_PER_LINK (512 by default). The later check is not performed if the __PL_DYNAMIC_COUNTERS__ symbol is defined. If the __PL_EXTRA_INPUT_CHECKS__ symbol is defined, then additional checks are performed on user inputs as listed below. If any of these checks fail, then this error is returned. It is highly recommended to define this symbol in general and especially if security is a concern.

- Application and counters names are null terminated
- Application and counters names contain only allowed characters. Allowed characters are: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '_', '-', '(', ')', '.', ' ', '[' and ']'
- Application and counters names lengths are within the allowed ranges
- When built in file system-less mode (when the __PL_FILE_SYSTEM_LESS__ symbol is defined), then the following checks are performed:
  - IPV4 address is well formed (length and composition)
  - IPV4 address belongs to Class A, B, C, D or E
  - Port number is a valid port number

PL_MISSING_DIRECTORY

The PL_FOLDER doesn't exist. Contact the system administrator to create the PL_FOLDER with the appropriated read and write permissions.

PL_NOT_A_DIRECTORY

PL_FOLDER does exist but is not a directory. Contact the system administrator to create the PL_FOLDER with the appropriated read and write permissions.

PL_DIRECTORY_ALREADY_EXISTS

The directory to be used by the PL already exists. This is a collision case that may happen when a uuid was not guaranteed to be unique.

Check the PL_NON_GLOBAL_UUID_DESCRIPTOR and PL_NON_GLOBAL_UUID_DESCRIPTOR_NO_ADDRESS internal error codes for more details.

PL_DIRECTORY_CREATION_FAILED

The directory to be used by the PL cannot be created. Contact the system administrator to check the application's access credentials to the PL_FOLDER.

PL_COUNTER_CREATION_FAILED

The file to be used by a counter cannot be created. Contact the system administrator to check the application's access credentials to the PL_FOLDER.

PL_OUT_OF_MEMORY

There is not enough memory available to allocate data storage. This error may be reported when the counter or the table allocation is performed dynamically. Try freeing up system memory.

PL_FILESYSTEM_LESS_INVALID_IPV4_ADDRESS

An invalid and / or malformed IPV4 address was specified via the PL_AGENT_ADDRESS environment variable. Check the validity and the class of the IPV4 address. This error is returned if any of the following checks fails. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined.

- IPV4 address is well formed (length and composition)
- IPV4 address belongs to Class A, B, C, D or E

PL_FILESYSTEM_LESS_INVALID_PORT

An invalid port number was specified via the PL_AGENT_PL_PORT environment variable. Check the validity of the port. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined.

3.4.7 Internal Error Codes

PL_DESCRIPTOR_TABLE_FULL

There is no room to open the PL. The maximum number of PLs is defined by PL_MAX_PRODUCTIVITY_LINKS (ten by default).

PL_NON_GLOBAL_UUID_DESCRIPTOR

The uuid returned and used is not guaranteed to be unique. Therefore, a collision may occur.

PL_NON_GLOBAL_UUID_DESCRIPTOR_NO_ADDRESS

The uuid generation process is missing a network address. Therefore, a collision may occur.

PL_GLOBAL_UUID_DESCRIPTOR_CREATION_FAILED

No uuid was generated. This is a critical error.

PL_GLOBAL_UUID_DESCRIPTOR_TO_STRING_FAILED

It was not possible to convert the uuid into a string. This error may happen on very low memory conditions.

PL_PATH_NOT_FOUND

The directory to be used by the PL was not found. This may happen if the directory was deleted by an external application or a user.

PL_COUNTER_SEMAPHORE_CREATION_FAILED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_CONFIG_FILE_GENERATION_FAILED

The configuration file (`pl_config.ini` as defined by PL_CONFIG_FILE_NAME) cannot be created. Either the application's access credentials to the PL_FOLDER are not high enough, or the storage space is limited. Contact the system administrator to check both conditions.

The configuration file is generated if the __PL_GENERATE_INI__ symbol is defined (this should be the default). It is highly recommended to generate the configuration file; otherwise, it will not be possible to use the `pl_attach()` function or numerous utilities (such as PL GUI Monitor) to attach to an existing PL.

PL_SYNCHRONIZATION_FAILED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_CRITICAL_FAILURE

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_OUT_OF_BUFFER_SPACE

An internal buffer ran out of space. This error is not related to the PL_OUT_OF_MEMORY error and cannot be solved by freeing up system memory. Please return a test case reproducing this error to Intel Corporation.

PL_FILESYSTEM_LESS_INITIALIZATION_FAILED

An internal error happened while the networking subsystem was initialized. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. Please return a test case reproducing this error to Intel Corporation.

PL_FILESYSTEM_LESS_SOCKET_FAILED

An internal error happened while a socket was created. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_CLOSE_SOCKET_FAILED

An internal error happened while a socket was closed. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_CONNECTION_FAILED

An internal error happened while a connection to a socket was attempted. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_SEND_FAILED

An internal error happened while data was sent through a socket. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_RECV_FAILED

An internal error happened while data was received from a socket. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_REMOTE_CRITICAL_FAILURE

A critical error happened on the agent or server side. This error can have multiple causes. However, this error code is returned when the PL protocol is not followed and a bogus message encoding is detected. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

3.5 pl_close()

Closes a previously opened or attached Productivity Link (PL) and releases the associated resources.

3.5.1 Syntax

```
int pl_close(
  int pl_descriptor
);
```

3.5.2 Parameters

pl_descriptor

A valid Productivity Link descriptor

3.5.3 Description

This function is declared in the `productivity_link.h` file. The function closes the PL identified by the *pl_descriptor* argument and frees up any memory or other resources associated with that PL. The function returns a status code.

3.5.4 Return Values

PL_SUCCESS

Indicates a successful operation.

PL_FAILURE

Indicates an error condition. If PL_FAILURE is returned, then the system's last error code is set as described in the Error Codes and Internal Error Codes sections.

3.5.5 Error Codes

PL_BYPASSED

The call to `pl_close()` was bypassed. This happens when the instrumentation is de-activated at compilation time. This is performed by defining __PL_BYPASS__.

PL_DESCRIPTOR_TABLE_UNINITIALIZED

The close operation was performed before a successful `pl_open()` or `pl_attach()` call. Open a Productivity Link before attempting to close one.

PL_INVALID_PARAMETERS

The *pl_descriptor* argument provided is invalid. This happens if the Productivity Link descriptor is lower than 1 or bigger than PL_MAX_PRODUCTIVITY_ LINKS (ten by default).

3.5.6 Internal Error Codes

PL_SYNCHRONIZATION_FAILED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_CRITICAL_FAILURE

An internal synchronization related critical error happened. Please return a test case reproducing this error to Intel Corporation.

PL_FILESYSTEM_LESS_SOCKET_FAILED

An internal error happened while a socket was created. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_CLOSE_SOCKET_FAILED

An internal error happened while a socket was closed. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_CONNECTION_FAILED

An internal error happened while a connection to a socket was attempted. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_SEND_FAILED

An internal error happened while data was sent through a socket. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_RECV_FAILED

An internal error happened while data was received from a socket. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_REMOTE_CRITICAL_FAILURE

A critical error happened on the agent or server side. This error can have multiple causes. However, this error code is returned when the PL protocol is not followed and a bogus message encoding is detected. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

3.6 pl_attach()

Attaches to an existing Productivity Link (PL).

3.6.1 Syntax

```
int pl_attach (
   char *pl_config_ini_file_name
);
```

3.6.2 Parameters

pl_config_ini_file_name

An ASCII zero terminated string fully qualifying the location of the Productivity Link configuration file (typically called `pl_config.ini`; see section 4.1 for information on PL storage locations).

3.6.3 Description

This function is declared in the `productivity_link.h` file. The function attaches to an existing PL, using its configuration file. The function returns the read-only PL descriptor to be used by all subsequent operations on this PL. The PL opened could still be in use by the application that created it, or it could refer to a PL from an application that terminated long ago.

NOTE

The pl_attach() function can be used to reference an existing Productivity Link on any folder that can be mapped to a file system; pl_attach() is not limited to the folders specified in Table 7. This capability enables one system to monitor PL counters on another system, provided that the monitoring system has permissions to access the other system's folders.

3.6.4 Sample Code

```
1  #include <stdio.h>
2  #include <windows.h> // only non-PL related system calls are OS specific
3  #include <assert.h>
4  #include "productivity_link.h"
5
6  //--------------------------------------------------------------------------
7  // defines
8  //--------------------------------------------------------------------------
9  #define UPDATE_INTERVAL 1000 // 1 second
10 enum COUNTERS { FRAMES = 0 };
11
12 //--------------------------------------------------------------------------
13 // function prototype
```

```
14  //--------------------------------------------------------------------------
15  BOOL control_handler(DWORD);
16
17  //--------------------------------------------------------------------------
18  // program global
19  //--------------------------------------------------------------------------
20  int pld = PL_INVALID_DESCRIPTOR;
21
22  //--------------------------------------------------------------------------
23  // program entry point
24  //--------------------------------------------------------------------------
25  int main(int argc, char *argv[]) {
26
27       PL_STATUS ret = PL_FAILURE;
28       BOOL bret = FALSE;
29
30       unsigned long long frames = 0;
31
32       //--------------------------------------------------------------------
33       // attach to a Productivity Link
34       //--------------------------------------------------------------------
35       pld = pl_attach(argv[1]);
36       assert(pld != PL_INVALID_DESCRIPTOR);
37
38       //--------------------------------------------------------------------
39       // loop until interrupted by user -- pl_config.ini location in argv[1]
40       //--------------------------------------------------------------------
41       while(TRUE) {
42
43          Sleep(UPDATE_INTERVAL);
44
45          //-----------------------------------------------------------------
46          // read energy data
47          //-----------------------------------------------------------------
48          ret = pl_read(
49              pld,
50              &frames,
51              FRAMES
52          );
53          assert(ret == PL_SUCCESS);
54          do_something_with(frames);
55       }
56       /* do a pl_close in the signal handler on the PL */
57  }
```

3.6.5 Return Values

PL_INVALID_DESCRIPTOR

Indicates an error condition. If PL_INVALID_DESCRIPTOR is returned, then the system's last error code is set as described in the Error Codes and the Internal Error Codes section. Any other value is a valid PL descriptor.

3.6.6 Error Codes

PL_BYPASSED

The call to `pl_attach()` was bypassed. This happens when the instrumentation is de-activated at compilation time. This is performed by defining __PL_BYPASS__.

PL_OUT_OF_MEMORY

There is not enough memory available to allocate data storage. This error may be reported when the counter or the table allocation is performed dynamically. Try freeing up system memory.

PL_INVALID_PARAMETERS

The *pl_config_ini_file_name* arguments provided is invalid. This happens if the mandatory pointer is NULL. If the __PL_EXTRA_INPUT_CHECKS__ symbol is defined, then additional checks are performed on the PL configuration file content as listed below. If any of these checks fail, then this error is returned. It is highly recommended to define this symbol in general and especially if security is a concern.

- PL configuration fully qualified name is null terminated
- PL configuration fully qualified name is well formed
- Application and counters names are null terminated
- Application and counters names contain only allowed characters. Allowed characters are: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '_', '-', '(', ')', '.', ' ', '[' and ']'
- Application and counters names lengths are within the allowed ranges
- When built in file system-less mode (when the __PL_FILE_SYSTEM_LESS__ symbol is defined), then the following checks are performed:
  - IPV4 address is well formed (length and composition)
  - IPV4 address belongs to Class A, B, C, D or E
  - Port number is a valid port number

3.6.7 Internal Error Codes

PL_DESCRIPTOR_TABLE_FULL

There is no room to attach the PL. The maximum number of PL is defined by PL_MAX_PRODUCTIVITY_LINKS (ten by default).

PL_COUNTER_SEMAPHORE_CREATION_FAILED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_CONFIG_FILE_OPENING_FAILED

The configuration file (`pl_config.ini` as defined by PL_CONFIG_FILE_NAME) cannot be opened. Either the application's access credentials to the PL_FOLDER are not high enough, or the storage space is limited or the file does not exist. Contact the system administrator to check if any of these conditions exist. The configuration file is generated during the `pl_open()` process if the __PL_GENERATE_INI__ symbol is defined, and that symbol should always be enabled (the default condition).

PL_SYNCHRONIZATION_FAILED

An internal synchronization error has occurred. Please return a test case reproducing this error to Intel Corporation.

PL_OUT_OF_BUFFER_SPACE

An internal buffer ran out of space. This error is not related to the PL_OUT_OF_MEMORY error and cannot be solved by freeing up system memory. Please return a test case reproducing this error to Intel Corporation.

PL_FILESYSTEM_LESS_INITIALIZATION_FAILED

An internal error happened while the networking subsystem was initialized. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. Please return a test case reproducing this error to Intel Corporation.

PL_FILESYSTEM_LESS_SOCKET_FAILED

An internal error happened while a socket was created. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_CLOSE_SOCKET_FAILED

An internal error happened while a socket was closed. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_CONNECTION_FAILED

An internal error happened while a connection to a socket was attempted. Check if the agent or server is functional and strictly follows the PL protocol. This error

can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_SEND_FAILED

An internal error happened while data was sent through a socket. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_RECV_FAILED

An internal error happened while data was received from a socket. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_REMOTE_CRITICAL_FAILURE

A critical error happened on the agent or server side. This error can have multiple causes. However, this error code is returned when the PL protocol is not followed and a bogus message encoding is detected. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

3.7 pl_write()

Writes a value into a Productivity Link (PL) counter.

3.7.1 Syntax

```
int pl_write(
  int pl_descriptor,
  const void *pointer_to_data,
  unsigned int counter_offset
);
```

3.7.2 Parameters

pl_descriptor

A valid Productivity Link descriptor.

pointer_to_data

A valid pointer to a memory location storing an unsigned long long value.

counter_offset

A valid index in the PL's counters list (zero-relative).

3.7.3 Description

This function is declared in the `productivity_link.h` file. The function writes the value of the variable at *pointer_to_data* into the PL counter identified by *counter_offset*. The variable is an unsigned long long value. *pl_descriptor* identifies the PL to be used.

For example, if the target PL is opened using the counter arguments as shown in the code snippet below (this information is fully documented in the `pl_config.ini` file), then a subsequent *counter_offset* of 0 writes the data into the *Frames* counter. A *counter_offset* of 1 writes the data into the *Pixels* counter.

```
1  #define COUNTERS_COUNT 2
2  #define COUNTERS_NAMES { "Frames", "Pixels" }
3  char *counters[COUNTERS_COUNT] = COUNTERS_NAMES;
```

NOTE

The write operation is cached. This means that no write operation occurs if the value to be written is identical to the previously written value.

The Intel EC API initializes counters in its cache with a value of 2^64-2. An initial value of 2^64-2 will not be written out to the persistent counter location unless some other value (such as zero) is written to that counter first.

3.7.4 Sample Code

```
#include "productivity_link.h"

//--------------------------------------------------------------------------
// defines
//--------------------------------------------------------------------------
enum COUNTERS { FRAMES = 0 };

unsigned long long frames = 0;
PL_STATUS ret = PL_FAILURE;

//--------------------------------------------------------------------------
// write frame data
//--------------------------------------------------------------------------
frames = get_encoded_frames_count();
ret = pl_write(
    pld,
    &frames,
    FRAMES
);
assert(ret == PL_SUCCESS);
...
```

3.7.5 Return Values

PL_SUCCESS

Indicates a successful operation.

PL_FAILURE

Indicates an error condition. If PL_FAILURE is returned, then the system's last error code is set as described in the Error Codes and Internal Error Codes sections.

3.7.6 Error Codes

PL_BYPASSED

The call to `pl_write()` was bypassed. This happens when the instrumentation is de-activated at compilation time. This is performed by defining __PL_BYPASS__.

PL_DESCRIPTOR_TABLE_UNINITIALIZED

The write operation was performed before a successful `pl_open()` or `pl_attach()` call. Open a Productivity Link before attempting to close one.

PL_INVALID_PARAMETERS

At least one of the arguments provided is invalid. This happens if the Productivity Link descriptor is less than 1 or greater than PL_MAX_PRODUCTIVITY_LINKS (ten by default). This also happens when the destination pointer is NULL.

3.7.7 Internal Error Codes

PL_SYNCHRONIZATION_FAILED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_COUNTER_FILE_LOCK_FAILED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_COUNTER_FILE_ALREADY_LOCKED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_COUNTER_FILE_UNLOCK_FAILED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_COUNTER_FILE_RESET_FILE_POINTER_FAILED

An internal IO error happened. Please return a test case reproducing this error to Intel Corporation.

PL_COUNTER_WRITE_FAILED

An internal IO error happened. Please return a test case reproducing this error to Intel Corporation.

PL_FILESYSTEM_LESS_SOCKET_FAILED

An internal error happened while a socket was created. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_CLOSE_SOCKET_FAILED

An internal error happened while a socket was closed. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_CONNECTION_FAILED

An internal error happened while a connection to a socket was attempted. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_SEND_FAILED

An internal error happened while data was sent through a socket. Check if the agent or server is functional and strictly follows the PL protocol. This error can

be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_RECV_FAILED

An internal error happened while data was received from a socket. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_REMOTE_CRITICAL_FAILURE

A critical error happened on the agent or server side. This error can have multiple causes. However, this error code is returned when the PL protocol is not followed and a bogus message encoding is detected. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

3.8 pl_read()

Reads the value of a Productivity Link (PL) counter.

3.8.1 Syntax

```
int pl_read(
  int pl_descriptor,
  const void *ponter_to_data,
  unsigned int counter_offset
);
```

3.8.2 Parameters

pl_descriptor

A valid Productivity Link descriptor.

pointer_to_data

A valid pointer to a memory location large enough to receive an unsigned long long value.

counter_offset

A valid index in the PL's counters list (zero-relative).

3.8.3 Description

This function is declared in the `productivity_link.h` file. The function reads the PL counter identified by the *counter_offset* argument. The counter value is stored in memory at *pointer_to_data*. *pl_descriptor* indicates the PL to be used.

For example, if the target PL was opened using the following counter arguments as shown in the code snippet below, then a subsequent *counter_offset* of 0 reads the data from the *Frames* counter. A *counter_offset* of 1 reads the data from the *Pixels* counter and so on.

```
1  #define COUNTERS_COUNT 2
2  #define COUNTERS_NAMES { "Frames", "Pixels" }
3  char *counters[COUNTERS_COUNT] = COUNTERS_NAMES;
```

3.8.4 Sample Code

```
#include "productivity_link.h"

//------------------------------------------------------------------------------
// defines
//------------------------------------------------------------------------------
enum COUNTERS { FRAMES = 0 };
...
unsigned long long frames = 0;
PL_STATUS ret = PL_FAILURE;
...
//------------------------------------------------------------------------------
// read energy data
//------------------------------------------------------------------------------
ret = pl_read(
    pld,
    &frames,
    FRAMES
);
do_something_with(frames);
assert(ret == PL_SUCCESS);
...
```

3.8.5 Return Values

PL_SUCCESS

Indicates a successful operation.

PL_FAILURE

Indicates an error condition. If PL_FAILURE is returned, then the system's last error code is set as described in the Error Codes and Internal Error Codes sections.

3.8.6 Error Codes

PL_BYPASSED

The call to `pl_read()` was bypassed. This happens when the instrumentation is de-activated at compilation time. This is performed by defining __PL_BYPASS__.

PL_DESCRIPTOR_TABLE_UNINITIALIZED

The read operation was performed before a successful `pl_open()` or `pl_attach()` call. Open a Productivity Link before attempting to read one.

PL_INVALID_PARAMETERS

At least one of the arguments provided is invalid. This happens if the Productivity Link descriptor is lower than 1 or bigger than PL_MAX_PRODUCTIVITY_ LINKS (ten by default). This also happens when the destination pointer is null.

PL_COUNTER_VALUE_OUT_OF_RANGE

The counter value is not a valid unsigned long long.

3.8.7 Internal Error Codes

PL_SYNCHRONIZATION_FAILED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_COUNTER_FILE_LOCK_FAILED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_COUNTER_FILE_ALREADY_LOCKED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_COUNTER_FILE_UNLOCK_FAILED

An internal synchronization error happened. Please return a test case reproducing this error to Intel Corporation.

PL_COUNTER_FILE_RESET_FILE_POINTER_FAILED

An internal IO error happened. Please return a test case reproducing this error to Intel Corporation.

PL_COUNTER_READ_FAILED

An internal IO error happened. Please return a test case reproducing this error to Intel Corporation.

PL_FILESYSTEM_LESS_SOCKET_FAILED

An internal error happened while a socket was created. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_CLOSE_SOCKET_FAILED

An internal error happened while a socket was closed. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_CONNECTION_FAILED

An internal error happened while a connection to a socket was attempted. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_SEND_FAILED

An internal error happened while data was sent through a socket. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_RECV_FAILED

An internal error happened while data was received from a socket. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

PL_FILESYSTEM_LESS_REMOTE_CRITICAL_FAILURE

A critical error happened on the agent or server side. This error can have multiple causes. However, this error code is returned when the PL protocol is not followed and a bogus message encoding is detected. Check if the agent or server is functional and strictly follows the PL protocol. This error can be reported only if the __PL_FILESYSTEM_LESS__ symbol is defined. If the agent or the server can be excluded as a root cause of the error, please return a test case reproducing this error – with a binary of the agent or server – to Intel Corporation.

3.9 API Configuration Symbols

The API is provided as source code and uses several symbols to adapt its behavior. It is important to understand the use of these symbols, especially if the plan is to distribute instrumented applications to end customers.

3.9.1 Mandatory Symbols (Public Use)

Several symbols or preprocessor definitions are required to ensure that code written to use the SDK works properly. Table 3 below provides a list of the mandatory symbols for each supported environment.

Table 3: Required compilation symbols

Target Operating System	Preprocessor Definitions
Windows*	__PL_WINDOWS__ __PL_GENERATE_INI__ __PL_GENERATE_INI_VERSION_TAGGING__ __PL_GENERATE_INI_BUILD_TAGGING__ __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ __PL_BLOCKING_COUNTER_FILE_LOCK__
Windows DLL	_USRDLL __PL_WINDOWS__ __PL_GENERATE_INI__ __PL_GENERATE_INI_VERSION_TAGGING__ __PL_GENERATE_INI_BUILD_TAGGING__ __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ __PL_BLOCKING_COUNTER_FILE_LOCK__ __PL_WINDOWS_DLL_EXPORTS__
Windows JNI DLL	_USRDLL __PL_WINDOWS__ __PL_JNI_EXPORTS__ __PL_GENERATE_INI__ __PL_GENERATE_INI_VERSION_TAGGING__ __PL_GENERATE_INI_BUILD_TAGGING__ __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ __PL_BLOCKING_COUNTER_FILE_LOCK__ __PL_WINDOWS_DLL_EXPORTS__
Linux*	__PL_LINUX__ __PL_GENERATE_INI__ __PL_GENERATE_INI_VERSION_TAGGING__ __PL_GENERATE_INI_BUILD_TAGGING__ __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ __PL_BLOCKING_COUNTER_FILE_LOCK__
Linux SO	__PL_LINUX__ __PL_LINUX_SO_EXPORTS__ __PL_GENERATE_INI__ __PL_GENERATE_INI_VERSION_TAGGING__ __PL_GENERATE_INI_BUILD_TAGGING__ __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ __PL_BLOCKING_COUNTER_FILE_LOCK__

Target Operating System	Preprocessor Definitions
Solaris*	__PL_SOLARIS__ __PL_GENERATE_INI__ __PL_GENERATE_INI_VERSION_TAGGING__ __PL_GENERATE_INI_BUILD_TAGGING__ __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ __PL_BLOCKING_COUNTER_FILE_LOCK__
Solaris SO	__PL_SOLARIS__ __PL_SOLARIS_SO_EXPORTS__ __PL_GENERATE_INI__ __PL_GENERATE_INI_VERSION_TAGGING__ __PL_GENERATE_INI_BUILD_TAGGING__ __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ __PL_BLOCKING_COUNTER_FILE_LOCK__
MacOS* X	__PL_MACOSX__ __PL_GENERATE_INI__ __PL_GENERATE_INI_VERSION_TAGGING__ __PL_GENERATE_INI_BUILD_TAGGING__ __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ __PL_BLOCKING_COUNTER_FILE_LOCK__
MacOS X SO	__PL_MACOSX__ __PL_MACOSX_SO_EXPORTS__ __PL_GENERATE_INI__ __PL_GENERATE_INI_VERSION_TAGGING__ __PL_GENERATE_INI_BUILD_TAGGING__ __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ __PL_BLOCKING_COUNTER_FILE_LOCK__

The use of symbol `__PL_DYNAMIC_COUNTERS_ALLOCATION__` is optional but recommended on all supported OS types.

The use of symbol `__PL_EXTRA_INPUT_CHECKS __` is optional but recommended on all supported OS types.

All applications for use in production environments, or by organizations outside the developer's company **must use the mandatory compilation symbols listed in Table 3.**

Use of these mandatory compilation symbols expands the value of the Intel® Energy Checker for end users. Data exposed by applications can be used seamlessly by customer middleware, analysis or management tools.

3.9.2 Generic Build Configuration Symbols

_DEBUG

Define this symbol when building a debug version of the code. This symbol activates the compilation of specific debug code.

UNICODE and _UNICODE

Define both these symbols when compiling code for Windows operating systems. It is possible to define these symbols solely for the Core API source files.

__PL_LITTLE_ENDIAN__

Define this symbol when compiling a JNI dynamic library out of the Core API source code files. This symbol is required since the endianess of Java virtual machines (always big endian) may differ from the target platform's processor endianess. This symbol is used when returning the PLs' UUID bytes to the JVM.

__PL_DYNAMIC_COUNTERS_ALLOCATION__

Define this symbol to activate the dynamic allocation of PL counters' data. If this symbol is not defined, then the default counter count limitation (512 counters per PL – 10 PLs maximum) applies to the code. It is recommended to define this symbol.

3.9.3 OS Build Configuration Symbols

__PL_WINDOWS__

Define this symbol when building for Windows operating systems.

__PL_LINUX__

Define this symbol when building for Linux operating systems.

__PL_SOLARIS__

Define this symbol when building for the Solaris 10 operating system.

__PL_MACOSX__

Define this symbol when building for the MacOS X operating system.

NOTE

Bulding code for the MeeGo operating system requires using the __PL_LINUX__ symbol.*

3.9.4 Dynamic Library Build Configuration Symbols

__PL_WINDOWS_DLL_EXPORTS__

Define this symbol when building a dynamic library for Windows operating systems. This also applies to JNI dynamic library generation.

NOTE

By default, the DLLs are compiled with cdecl linkage. Some languages such as Microsoft Visual Basic require stdcall linkage. In this case, update the project's linker setting appropriately.

__PL_LINUX_SO_EXPORTS__

Define this symbol when building a shared object for Linux operating systems. This also applies to Java Native Interface (JNI) dynamic library generation.

__PL_SOLARIS_SO_EXPORTS__

Define this symbol when building a shared object for the Solaris 10 operating system. This also applies to Java Native Interface (JNI) dynamic library generation.

__PL_MACOSX_SO_EXPORTS__

Define this symbol when building a shared object for the MacOS X operating system. This also applies to Java Native Interface (JNI) dynamic library generation.

__PL_JNI_EXPORTS__

Define this symbol when building a dynamic library to be used via the Java Native Interface (JNI).

3.9.5 PL Functional Build Configuration Symbols

__PL_BYPASS__

Define this symbol to de-activate the Intel EC core API functions. When defined, each Intel EC core API function returns an error code and the system's last error code is set to `PL_BYPASSED`. Applications should gracefully handle this non-error code.

__PL_GENERATE_INI__

This mandatory symbol must be defined. When defined, the generation of a `pl_config.ini` file for each PL opened is activated (via `pl_open()` function call). The `pl_attach()` function and many utilities will not function properly if this symbol is not defined when the application is compiled.

__PL_GENERATE_INI_VERSION_TAGGING__

This recommended symbol should be defined. When defined, the `pl_config.ini` file is appended with version info. Version info has the following form: YYYY.MM.DD(O) where YYYY is the release year, MM is the release month, DD is the release day and O is the target operating system: `W` for Windows, `L` for Linux and MeeGo*, `S` for Solaris 10 and `M` for MacOS X. For example: `2009.07.01(L)`.

__PL_GENERATE_INI_BUILD_TAGGING__

This recommended symbol should be defined. When defined, the `pl_config.ini` file is appended with build info. Build info consist in the list of symbols defined during compilation. For example:

```
__PL_LINUX__
__PL_GENERATE_INI__
__PL_GENERATE_INI_VERSION_TAGGING__
__PL_GENERATE_INI_BUILD_TAGGING__
__PL_GENERATE_INI_DATE_AND_TIME_TAGGING__
__PL_BLOCKING_COUNTER_FILE_LOCK__
```

__PL_GENERATE_INI_DATE_AND_TIME_TAGGING__

This recommended symbol should be defined. When defined, the `pl_config.ini` file is appended with creation date and time info. For example:

```
PL created on Mon Sep 14 13:48:50 2009
```

__PL_BLOCKING_COUNTER_FILE_LOCK__

This mandatory symbol must be defined. When defined, the access to counter data is synchronized.

__PL_TIMESTAMPING__

This symbol is reserved for future revisions and should not be used.

__PL_EXTRA_INPUT_CHECKS__

This recommended symbol should be defined. When defined, the following extra checks are performed on the input. Input can be function arguments, environment variables or PL configuration file content.

- Application name contains only allowed characters
- Application name length matches length restrictions
- UUID is a well formed UUID (length and composition)
- Counter names contain only allowed characters
- Counter names lengths match length restrictions
- IPV4 address is well formed (length and composition)
- IPV4 address belongs to Class A, B, C, D or E
- Port number is a valid port number
- PL protocol encoding is respected

__PL_FILESYSTEM_LESS__

Define this symbol to activate the file system-less mode of the Intel EC Core API functions. Refer to section 0 for more details on this mode.

__PL_AGENT__

Define this symbol if you build an agent or server code.

3.9.6 Architecture-specific Build Configuration Symbol

Most of the code and symbols in the SDK are intended to be processor-agnostic. This section identifies a specific symbol for Intel® architecture processors.

__PL_NO_SSE_PAUSE_SUPPORT__

Define this symbol to deactivate the use of the pause IA instruction during the lock acquisition loops in the API functions.

The pause IA instruction delays execution of the next instruction for an implementation-specific amount of time. The delay is finite and can be zero for some processors. This instruction does not change the architectural state of the processor but is useful in situations where it is beneficial to moderate execution speed, such as in spin loops. The pause instruction is backward compatible with all IA-32 processors.

```
1   //-------------------------------------------------------------------------
2   // lock the counter file against other processes.
3   //-------------------------------------------------------------------------
4   #ifdef __PL_BLOCKING_COUNTER_FILE_LOCK__
5
6   overlapped.hEvent = 0;
7   overlapped.Offset = 0;
8   overlapped.OffsetHigh = 0;
9
10  pl_read_try_lock:
11
12  b_ret = LockFileEx(
13      pl_table.pl_array[pld].pl_counter_handles[counter],
14      LOCKFILE_EXCLUSIVE_LOCK,
15      0,
16      pl_counter_file_bytes_to_read,
17      0,
18      &overlapped
19  );
20  if(b_ret == FALSE) {
21     if(lock_attempt_count++ < PL_MAX_COUNTER_FILE_LOCK_ATTEMPTS) {
22  #ifndef __PL_NO_SSE_PAUSE_SUPPORT__
23        _mm_pause();
24  #endif // __PL_NO_SSE_PAUSE_SUPPORT__
25        goto pl_read_try_lock;
26     } else {
27        pl_last_error = PL_COUNTER_FILE_LOCK_FAILED;
28        goto pl_read_unlock_counter;
29     }
30  }
31
32  #else // __PL_BLOCKING_COUNTER_FILE_LOCK__
33
```

NOTE

Since inline assembly code (__asm) is not allowed with 64-bit code, the API requests the use of the pause intrinsic in compiler-generated spin loops. Compilers used with the SDK should support this intrinsic, otherwise use the __PL_NO_SSE_PAUSE_SUPPORT__ symbol.

NOTE

The SDK source code may support the IA64 (Intel® Itanium® processor) architecture in a future release.

3.9.7 Restricted PL Configuration Symbols

None of the following symbols are allowed to generate binaries to be deployed in production environments or outside the application developer's organization.

__PL_CONSTANT_UUID__

If defined, this private symbol forces the API `pl_open` function to always generate a UUID equal to `00000000-0000-0000-0000-000000000000`. This is intended for debug purposes only and should never be used.

⚠WARNING

*Use of this symbol will **automatically** lead to data collision. This mode requires that any previous PL is erased prior to re-running the application.*

__PL_BINARY_COUNTER__

If defined, this private symbol forces the API functions to use binary counters.

__PL_DYNAMIC_TABLE_ALLOCATION__

If defined, this private symbol forces the API functions to dynamically allocate the PLs. By default, the API is limited to ten (10) PLs per applications. By defining this symbol, this limitation is waved. The sole reason why this symbol is marked private use only is because there is no function in the API to de-allocate the memory used to store PL data (not the counter data). Even if at application's end this memory is de-allocated by the operating system, this symbol may lead to memory leaks. If more than ten (10) PLs are required in an application, modify the `PL_MAX_PRODUCTIVITY_LINKS` definition (line 143 of `productivity_link.h` file).

__PL_UNIT_TESTS__

This private symbol must be defined when building the API unit tests.

__PL_LOGGER_ONLY__

If defined, this private symbol forces the API `pl_write()` function to timestamp and log all non-cached counter updates within the counter files. This symbol causes the API to append (rather than overwrite) counters. Code using the `pl_read()` function will fail to compile if this symbol is defined.

3.9.8 Unsupported Build Configuration Symbols

These are internal symbols used for the development of the SDK and should never be used. The symbols described in this section are listed because they occur in the source code distributed with the SDK. Code built with this symbol may behave in unpredictable ways.

__PL_BLOCKING_PL_READ__

This symbol may be used in a future revision of the SDK.

__PL_IN_MEMORY __

This symbol may be used in a future revision of the SDK.

NOTE

When using Microsoft Visual Studio to build the code under Windows, a summary listing is printed while compiling the `productivity_link.c` *file. The listing below is an example of such output.*

```
1>productivity_link.c
1>//------------------------------------------------------------------------
1>// PL Build configuration report.
1>//------------------------------------------------------------------------
1>NOTE: Building using _DEBUG.
1>NOTE: Building using _UNICODE.
1>NOTE: Building using UNICODE.
1>NOTE: Building using __PL_DYNAMIC_COUNTERS_ALLOCATION__.
1>NOTE: Building using __PL_FILESYSTEM_LESS__.
1>NOTE: Building using __PL_EXTRA_INPUT_CHECKS__.
1>NOTE: Building using __PL_WINDOWS__.
1>NOTE: Building using __PL_GENERATE_INI__.
1>NOTE: Building using __PL_GENERATE_INI_VERSION_TAGGING__.
1>NOTE: Building using __PL_GENERATE_INI_BUILD_TAGGING__.
1>NOTE: Building using __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__.
1>NOTE: Building using __PL_BLOCKING_COUNTER_FILE_LOCK__.

```

3.10 File System-less Mode

The API uses a local or a distributed file system to store the counter data by default. However, some devices – such as mobile phones, tablet PCs or any file system-less embedded system – may not have access to a file system. For these extreme cases, the API can be compiled to run in file system-less mode. To do so, simply define the `__PL_FILESYSTEM_LESS__` symbol during the build process. This will turn the instrumented application into a TCP/IP V4 client, using the PL protocol (see Section 0) to communicate with at least one reachable network agent. Agents are the servers in this scenario.

The SDK is shipped with a sample agent; its source code is available (`productivity_link_agent.c` and `productivity_link_agent.h` in `iecsdk\src\core_api` folder). This sample code can be studied to get acquainted with an implementation of the PL protocol in an agent, so a custom server can be devised. Such agent code requires the definition of the `__PL_AGENT__` symbol.

3.11 PL Protocol

The PL protocol is a simple network protocol designed to encapsulate and send API calls to a networked agent, and to receive and decode a networked agent's answer to the API calls. From the application's point of view, there are no functional differences between the file-system-based and the file system-less mode. Performance-wise, the network overhead and jitter must be considered when using the API in file system-less mode. The application should also handle error conditions thoroughly since a network is generally less reliable than a file system.

NOTE

Information provided on the protocol itself is useful only when implementing an agent, or building a server into an application. Many details provided in this section target server developers willing to add support for the PL protocol to their software and serve API calls autonomously. See section 10.11 for an example of such code.

NOTE

Only the API is impacted by the _PL_FILESYSTEM_LESS_ symbol. Therefore, no other helper code or the unit test code will work in file system-less mode. This is because these routines directly access the file system without using the API.

3.11.1 PL Message Format

A PL message is a well defined string of bytes. Non-textual data is encoded as binary data. Binary data is encoded in LE (little-endian) order. This means that the LSB (Least Significant Byte) is the first byte, and the MSB (Most Significant Byte) is the last byte received. A well formed PL message is composed of a:

- Header (4 bytes)
- Body (variable size, in bytes)
- EOR – End of Record (1 byte)

NOTE

The message header encodes the size (in bytes) of the body and the end of record. This size doesn't include the size of the header itself. For example, a header, a message body and the end of record composed of 100 bytes will have its first four bytes equal to 0x60 0x00 0x00 0x00 (96 in decimal). This allows for a fast send/receive mechanism. Indeed, it suffices for the receiver to read four bytes from a socket to know how many bytes must be read to fully receive a message. The emitter sends the entire message in a single operation.

3.11.2 Status Code

The protocol defines a `PL_PROTOCOL_STATUS`. A PL protocol status can be equal to `PL_PROTOCOL_FAILURE` or to `PL_PROTOCOL_SUCCESS`. These codes are used by the agent and the API code compiled in file system-less mode. Applications using the core API do not need to refer to these status codes. Instead, they must use the `PL_STATUS` (`PL_FAILURE` or `PL_SUCCESS`). If a network- or protocol-related error is detected, then `PL_FAILURE` is returned by the function and the system error code is set accordingly. Additional details on error codes can be found in Section 0.

3.11.3 String Encoding

The API uses strings to represent the application name, the counter names, and the PL configuration file name and path. A well formed string is composed of a:

- Header (4 bytes, encoding the value N, number of characters)
- Body (N bytes, each of them encoding a single character)

NOTE

Strings are NOT terminated by a null character.

NOTE

In this section, the following color codes are used to highlight message elements:

- *Header*
- *Operation code*
  UUID
- *PL descriptor*
  String size
- *String data*
  Counter count
  Counter offset
  Counter value
  Status
  End of record

3.11.4 pl_open() Encoding

When called in file system-less mode, pl_open() builds a PL message with a body composed of the following:

- Operation code (1 byte)
- Counter count (4 bytes, encoding the value N – number of counters)
- String (variable size – application name)
- N Strings (variable size – counter names)

An agent should return a message with a body composed of the following:

- Status code (4 bytes)
- UUID (16 bytes)
- PL descriptor (4 bytes)

As an example, assume the following call to pl_open(). Note that only relevant data is shown in the listing below.

```
const char *counters[5] = {
   "Hello",
   "World",
   NULL,
   "A",
   "b"
};

pld = pl_open(
   "Application in filesystem-less mode",
   5,
   counters,
   &uuid
);
```

The following HEX memory dump shows the PL message sent to the agent. Addresses are arbitrary.

```
0x0012DC38  60 00 00 00 01 05 00 00 00 23 00 00 00 41 70 70  `........#...App
0x0012DC48  6c 69 63 61 74 69 6f 6e 20 69 6e 20 66 69 6c 65  lication in file
0x0012DC58  73 79 73 74 65 6d 2d 6c 65 73 73 20 6d 6f 64 65  system-less mode
0x0012DC68  05 00 00 00 48 65 6c 6c 6f 05 00 00 00 57 6f 72  ....Hello....Wor
0x0012DC78  6c 64 13 00 00 00 61 6e 6f 6e 79 6d 6f 75 73 5f  ld....anonymous_
0x0012DC88  63 6f 75 6e 74 65 72 5f 32 01 00 00 00 41 01 00  counter_2....A..
0x0012DC98  00 00 62 0d 00 00 00 00 00 00 00 00 00 00 00 00  ..b.............
```

The returned PL descriptor is zero (0) and the UUID is CF8C9562-561D-4A20-A5BB-443061652328. The call is successful (also visible in the PL protocol status equal to *PL_PROTOCOL_SUCCESS*). The following HEX memory dump shows the PL message received from the agent. Addresses are arbitrary.

```
0x0012EC40  19 00 00 00 00 00 00 00 62 95 8c cf 1d 56 20 4a  ........b.ŒÏ.V J
0x0012EC50  a5 bb 44 30 61 65 23 28 01 00 00 00 0d 00 00 00  ¥»D0ae#(........
```

When compiled in debug mode, the sample agent prints – among other data – a HEX dump and a clear decode of the PL messages received from clients and sent to the clients. This is a facility that can be used to analyze the use of the PL protocol. The listing below shows the log extract for the previous example.

```
Pool thread [0] is serving a PL API call.
...Pool thread [0] has received...
......Pool thread [0]: Bytes in full message: [100]d – [64]h.
......Pool thread [0]: Bytes in message (skipping size header): [96]d – [60]h.
......Pool thread [0]: 60 00 00 00 01 05 00 00 00 23 00 00 00 41 70 70 6C 69 63 61 74
69 6F 6E 20 69 6E 20 66 69 6C 65 73 79 73 74 65 6D 2D 6C 65 73 73 20 6D 6F 64 65 05 00
00 00 48 65 6C 6C 6F 05 00 00 00 57 6F 72 6C 64 13 00 00 00 61 6E 6F 6E 79 6D 6F 75 73
5F 63 6F 75 6E 74 65 72 5F 32 01 00 00 00 41 01 00 00 00 62 0D
......Pool thread [0]: xx xx xx xx 01 05 00 00 00 23 00 00 00 41 70 70 6C 69 63 61 74
69 6F 6E 20 69 6E 20 66 69 6C 65 73 79 73 74 65 6D 2D 6C 65 73 73 20 6D 6F 64 65 05 00
00 00 48 65 6C 6C 6F 05 00 00 00 57 6F 72 6C 64 13 00 00 00 61 6E 6F 6E 79 6D 6F 75 73
5F 63 6F 75 6E 74 65 72 5F 32 01 00 00 00 41 01 00 00 00 62 0D
......Pool thread [0]: Op code = [PL_PROTOCOL_OPCODE_OPEN].
......Pool thread [0]: Counters count = [5].
......Pool thread [0]: Application name length = [35].
......Pool thread [0]: Application name = [Application in filesystem-less mode].
......Pool thread [0]: Counter [0] length = [5].
......Pool thread [0]: Counter [0] name = [Hello].
......Pool thread [0]: Counter [1] length = [5].
......Pool thread [0]: Counter [1] name = [World].
......Pool thread [0]: Counter [2] length = [19].
......Pool thread [0]: Counter [2] name = [anonymous_counter_2].
......Pool thread [0]: Counter [3] length = [1].
......Pool thread [0]: Counter [3] name = [A].
......Pool thread [0]: Counter [4] length = [1].
......Pool thread [0]: Counter [4] name = [b].
......Pool thread [0]: Last byte = [13] – [PL_PROTOCOL_EOR].
...Pool thread [0] is sending...
......Pool thread [0]: Bytes in full message: [29]d – [1d]h.
......Pool thread [0]: Bytes in message (skipping size header): [25]d – [19]h.
......Pool thread [0]: 19 00 00 00 00 00 00 00 62 95 8C CF 1D 56 20 4A A5 BB 44 30 61
65 23 28 01 00 00 00 0D
......Pool thread [0]: xx xx xx xx 00 00 00 00 62 95 8C CF 1D 56 20 4A A5 BB 44 30 61
65 23 28 01 00 00 00 0D
......Pool thread [0]: Status = [PL_PROTOCOL_SUCCESS].
......Pool thread [0]: Answer to op code = [PL_PROTOCOL_OPCODE_OPEN].
......Pool thread [0]: uuid = [cf8c9562-561d-4a20-a5bb-443061652328].
......Pool thread [0]: pld = [1].
......Pool thread [0]: Last byte = [13] – [PL_PROTOCOL_EOR].
```

In file system-less mode, two important items need to be remembered. First, in file system-less mode, the PL descriptor values are different between the client (the application) and the server (the agent). The client PL descriptor is 0 (in this

example code) and the server descriptor is 1 (in this example code). (The descriptors will likely be different in other codes.) This is due to the fact that the agent is serving multiple clients and that its PL descriptor table may already be in use. In this example case, another client already performed a call to pl_open() in files system-less mode. However, the pl_open() call is the first for this client and predictably, the PL descriptor return is 0.

Second, the UUID is also different between the client and the server. The client UUID is CF8C9562-561D-4A20-A5BB-443061652328 and the server UUID is CF8C9562-561D-4A20-A5BB-443061652328. The cause of this discrepancy is the same as for the PL descriptor discrepancy. The API is automatically performing the mapping between the client and the server PL descriptors and UUIDs.

3.11.5 pl_attach() Encoding

When called in file system-less mode, pl_attach() builds a PL message with a body composed of a the following:

- Operation code (1 byte)
- String (variable size – fully qualified PL configuration file name)

An agent should return a message with a body composed of the following:

- Status code (4 bytes)
- UUID (16 bytes)
- PL descriptor (4 bytes)
- Counter count (4 bytes)

Besides the status code and the PL descriptor, the agent returns a UUID and a counter count. This data is required to populate the local PL table, so subsequent API calls can be carried out. For example, the counter count can be used to perform several checks and to store the counter values used by pl_read() and pl_write().

Assume the following call to pl_attach(). Note that only relevant data is shown in the listing below.

```
1  pld = pl_attach("C:\\productivity_link\\esrv_5de9d14a-5522-4091-89c2-
2  1317abc067c6\\pl_config.ini");
```

The following HEX memory dump shows the PL message sent to the agent. Addresses are arbitrary.

```
0x0012DD58   52 00 00 00 02 4c 00 00 00 43 3a 5c 70 72 6f 64   R....L...C:\prod
0x0012DD68   75 63 74 69 76 69 74 79 5f 6c 69 6e 6b 5c 65 73   uctivity_link\es
0x0012DD78   72 76 5f 35 64 65 39 64 31 34 61 2d 35 35 32 32   rv_5de9d14a-5522
0x0012DD88   2d 34 30 39 31 2d 38 39 63 32 2d 31 33 31 37 61   -4091-89c2-1317a
0x0012DD98   62 63 30 36 37 63 36 5c 70 6c 5f 63 6f 6e 66 69   bc067c6\pl_confi
0x0012DDA8   67 2e 69 6e 69 0d 00 00 00 00 00 00 00 00 00 00   g.ini...........
```

The returned PL descriptor is zero (0). The call is successful (also visible in the PL protocol status equal to `PL_PROTOCOL_SUCCESS`). The following HEX memory dump shows the PL message received from the agent. Addresses are arbitrary. The UUID generated by the agent for this call is A412405A-4511-4D73-AFF1-8D09A7E3E519. The counter count is 0x1f8 (504 in decimal) which corresponds to the counter count stored in the PL configuration file (see last listing of this section for a partial pl_config.ini listing.)

```
0x0012ED60   1d 00 00 00 00 00 00 00 5a 40 12 a4 11 45 73 4d   ........Z@.¤.EsM
0x0012ED70   af f1 8d 09 a7 e3 e5 19 00 00 00 00 f8 01 00 00   ¯ñ..§ãå.....ø...
0x0012ED80   0d 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

When compiled in debug mode, the sample agent prints – among other data – a HEX dump of the messages received and the messages sent. This is a facility that can be used to analyze the use of the PL protocol. The listing below shows the log for the previous example.

```
...Pool thread [0] has received...
......Pool thread [0]: Bytes in full message: [86]d - [56]h.
......Pool thread [0]: Bytes in message (skipping size header): [82]d - [52]h.
......Pool thread [0]: 52 00 00 00 02 4C 00 00 00 43 3A 5C 70 72 6F 64 75 63 74 69 76
69 74 79 5F 6C 69 6E 6B 5C 65 73 72 76 5F 35 64 65 39 64 31 34 61 2D 35 35 32 32 2D 34
30 39 31 2D 38 39 63 32 2D 31 33 31 37 61 62 63 30 36 37 63 36 5C 70 6C 5F 63 6F 6E 66
69 67 2E 69 6E 69 0D
......Pool thread [0]: xx xx xx xx 02 4C 00 00 00 43 3A 5C 70 72 6F 64 75 63 74 69 76
69 74 79 5F 6C 69 6E 6B 5C 65 73 72 76 5F 35 64 65 39 64 31 34 61 2D 35 35 32 32 2D 34
30 39 31 2D 38 39 63 32 2D 31 33 31 37 61 62 63 30 36 37 63 36 5C 70 6C 5F 63 6F 6E 66
69 67 2E 69 6E 69 0D
......Pool thread [0]: Op code = [PL_PROTOCOL_OPCODE_ATTACH].
......Pool thread [0]: PL configuration file name length = [76].
......Pool thread [0]: PL configuration file name =
[C:\productivity_link\esrv_5de9d14a-5522-4091-89c2-1317abc067c6\pl_config.ini].
......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
...Pool thread [0] is sending...
......Pool thread [0]: Bytes in full message: [33]d - [21]h.
......Pool thread [0]: Bytes in message (skipping size header): [29]d - [1d]h.
......Pool thread [0]: 1D 00 00 00 00 00 00 00 5A 40 12 A4 11 45 73 4D AF F1 8D 09 A7
E3 E5 19 00 00 00 00 F8 01 00 00 0D
......Pool thread [0]: xx xx xx xx 00 00 00 00 5A 40 12 A4 11 45 73 4D AF F1 8D 09 A7
E3 E5 19 00 00 00 00 F8 01 00 00 0D
```

```
......Pool thread [0]: Status = [PL_PROTOCOL_SUCCESS].
......Pool thread [0]: Answer to op code = [PL_PROTOCOL_OPCODE_ATTACH].
......Pool thread [0]: uuid = [a412405a-4511-4d73-aff1-8d09a7e3e519].
......Pool thread [0]: pld = [0].
......Pool thread [0]: counters count = [504].
......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
```

The listing below is a partial reproduction of the pl_config.ini attached to the example above. Note that this PL was generated by ESRV running an eight-channel power analyzer, hence the unusually high number of counters (504).

```
esrv
5de9d14a-5522-4091-89c2-1317abc067c6
C:\productivity_link\esrv_5de9d14a-5522-4091-89c2-1317abc067c6\
504
C:\productivity_link\esrv_5de9d14a-5522-4091-89c2-1317abc067c6\[CHANNEL1] - Energy
(Joule)
...
C:\productivity_link\esrv_5de9d14a-5522-4091-89c2-1317abc067c6\[CHANNEL8] - Channel(s)
C:\productivity_link\esrv_5de9d14a-5522-4091-89c2-1317abc067c6\[CHANNEL8] - Status
C:\productivity_link\esrv_5de9d14a-5522-4091-89c2-1317abc067c6\[CHANNEL8] - Version
2010.12.15.(W)
_DEBUG
_UNICODE
UNICODE
__PL_DYNAMIC_COUNTERS_ALLOCATION__
__PL_EXTRA_INPUT_CHECKS__
__PL_WINDOWS__
__PL_GENERATE_INI__
__PL_GENERATE_INI_VERSION_TAGGING__
__PL_GENERATE_INI_BUILD_TAGGING__
__PL_GENERATE_INI_DATE_AND_TIME_TAGGING__
__PL_BLOCKING_COUNTER_FILE_LOCK__
PL created on Tue Nov 02 13:08:21 2010

```

3.11.6 pl_close() Encoding

When called in file system-less mode, pl_close() builds a PL message with a body composed of the following:

- Operation code (1 byte)
- UUID (16 bytes)
- PL descriptor (4 bytes)

An agent should return a message with a body composed of the following:

- Status Code (4 bytes)

Assume the following call to pl_close(). Note that only relevant data is shown in the listing below.

```
1  pld = pl_open(
2     "Application in filesystem-less mode",
3     5,
4     counters,
5     &uuid
6  );
7  pl_ret = pl_close(pld);
```

The following HEX memory dump shows the PL message sent to the agent. Addresses are arbitrary.

```
1  0x0012DD64  16 00 00 00 03 81 ca 56 92 a3 97 f7 4b 8b 37 f1  ......ÊV'£—÷K.7ñ
2  0x0012DD74  a0 51 68 8e c8 00 00 00 00 0d 00 00 00 00 00 00   QhŽÈ.....
```

The returned status is `PL_SUCCESS`. The call is successful (also visible in the PL protocol status equal to `PL_PROTOCOL_SUCCESS`). The following HEX memory dump shows the PL message received from the agent. Addresses are arbitrary.

```
1  0x0012ED6C  05 00 00 00 00 00 00 00 0d 00 00 00 00 00 00 00  ................
```

When compiled in debug mode, the sample agent prints – among other data – a HEX dump of the messages received and the messages sent. This is a facility that can be used to analyze the use of the PL protocol. The listing below shows the log for the previous example.

```
...Pool thread [0] has received...
......Pool thread [0]: Bytes in full message: [26]d - [1a]h.
......Pool thread [0]: Bytes in message (skipping size header): [22]d - [16]h.
......Pool thread [0]: 16 00 00 00 03 81 CA 56 92 A3 97 F7 4B 8B 37 F1 A0 51 68 8E C8
00 00 00 00 0D
......Pool thread [0]: xx xx xx xx 03 81 CA 56 92 A3 97 F7 4B 8B 37 F1 A0 51 68 8E C8
00 00 00 00 0D
......Pool thread [0]: Op code = [PL_PROTOCOL_OPCODE_CLOSE].
......Pool thread [0]: uuid = [761f44a8-2409-4d9e-bb56-2234718c03a1].
......Pool thread [0]: pld = [0].
......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
...Pool thread [0] is sending...
......Pool thread [0]: Bytes in full message: [9]d - [9]h.
......Pool thread [0]: Bytes in message (skipping size header): [5]d - [5]h.
......Pool thread [0]: 05 00 00 00 00 00 00 00 0D
......Pool thread [0]: xx xx xx xx 00 00 00 00 0D
......Pool thread [0]: Status = [PL_PROTOCOL_SUCCESS].
......Pool thread [0]: Answer to op code = [PL_PROTOCOL_OPCODE_CLOSE].
......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
```

3.11.7 pl_read() Encoding

When called in file system-less mode, pl_read() builds a PL message with a body composed of the following:

- Operation code (1 byte)
- UUID (16 bytes)
- PL descriptor (4 bytes)
- Counter offset (4 bytes)

An agent should return a message with a body composed of the following:

- Status Code (4 bytes)
- Counter value (8 bytes)

Assume the following call to pl_read(). Note that only relevant data is shown in the listing below.

```
pld = pl_open(
   "Application in filesystem-less mode",
   5,
   counters,
   &uuid
);

value = PL_MAX_COUNTER_VALUE;

pl_ret = pl_write(
   pld,
   &value,
   1
);

value = PL_MIN_COUNTER_VALUE;

pl_ret = pl_read(
   pld,
   &value,
   1
);

pl_ret = pl_close(pld);
```

The following HEX memory dump shows the PL message sent to the agent. Addresses are arbitrary.

```
0x0012DD58   1a 00 00 00 04 79 7c d6 fd 0f 7a 52 4d b5 00 6c   .....y|Öý.zRMµ.l
0x0012DD68   f4 a1 e7 89 26 00 00 00 00 01 00 00 00 0d 00 00   ô¡ç.&...........
```

The returned status is `PL_SUCCESS`. The call is successful (also visible in the PL protocol status equal to `PL_PROTOCOL_SUCCESS`). The following HEX memory dump shows the PL message received from the agent. Addresses are arbitrary.

```
0x0012ED60   0d 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff   ........ÿÿÿÿÿÿÿÿ
0x0012ED70   0d 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

When compiled in debug mode, the sample agent prints – among other data – a HEX dump of the messages received and the messages sent. This is a facility that can be used to analyze the use of the PL protocol. The listing below shows the log for the previous example.

```
...Pool thread [0] has received...
......Pool thread [0]: Bytes in full message: [30]d - [1e]h.
......Pool thread [0]: Bytes in message (skipping size header): [26]d - [1a]h.
......Pool thread [0]: 1A 00 00 00 04 79 7C D6 FD 0F 7A 52 4D B5 00 6C F4 A1 E7 89 26
00 00 00 00 01 00 00 00 0D
......Pool thread [0]: xx xx xx xx 04 79 7C D6 FD 0F 7A 52 4D B5 00 6C F4 A1 E7 89 26
00 00 00 00 01 00 00 00 0D
......Pool thread [0]: Op code = [PL_PROTOCOL_OPCODE_READ].
......Pool thread [0]: uuid = [fdd67c79-7a0f-4d52-b500-6cf4a1e78926].
......Pool thread [0]: pld = [0].
......Pool thread [0]: counter offset = [1].
......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
...Pool thread [0] is sending...
......Pool thread [0]: Bytes in full message: [17]d - [11]h.
......Pool thread [0]: Bytes in message (skipping size header): [13]d - [d]h.
......Pool thread [0]: 0D 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF 0D
......Pool thread [0]: xx xx xx xx 00 00 00 00 FF FF FF FF FF FF FF FF 0D
......Pool thread [0]: Status = [PL_PROTOCOL_SUCCESS].
......Pool thread [0]: Answer to op code = [PL_PROTOCOL_OPCODE_READ].
......Pool thread [0]: Value = [18446744073709551615].
......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
```

3.11.8 pl_write() Encoding

When called in file system-less mode, pl_write() builds a PL message with a body composed of the following:

- Operation code (1 byte)
- UUID (16 bytes)
- PL descriptor (4 bytes)
- Counter offset (4 bytes)
- Counter value (8 bytes)

An agent should return a message with a body composed of the following:

- Status Code (4 bytes)

Assume the following call to pl_write(). Note that only relevant data is shown in the listing below.

```
1   pld = pl_open(
2      "Application in filesystem-less mode",
3      5,
4      counters,
5      &uuid
6   );
7
8   value = PL_MAX_COUNTER_VALUE;
9
10  pl_ret = pl_write(
11     pld,
12     &value,
13     1
14  );
15
16  pl_ret = pl_close(pld);
```

The following HEX memory dump shows the PL message sent to the agent. Addresses are arbitrary.

```
1   0x0012DD58   22 00 00 00 05 26 0a 09 90 3e f0 cf 48 8d b3 d2   "....&...>ðÏH..Ò
2   0x0012DD68   2e 11 3c 38 d1 00 00 00 00 01 00 00 00 ff ff ff   ..<8Ñ........ÿÿÿ
3   0x0012DD78   ff ff ff ff ff 0d 00 00 00 00 00 00 00 00 00 00   ÿÿÿÿÿ...........
```

The returned status is `PL_SUCCESS`. The call is successful (also visible in the PL protocol status equal to `PL_PROTOCOL_SUCCESS`). The following HEX memory dump shows the PL message received from the agent. Addresses are arbitrary.

```
1   0x0012ED60   05 00 00 00 00 00 00 00 0d 00 00 00 00 00 00 00   ................
```

When compiled in debug mode, the sample agent prints – among other data – a HEX dump of the messages received and the messages sent. This is a facility that can be used to analyze the use of the PL protocol. The listing below shows the log for the previous example.

```
1  ...Pool thread [0] has received...
2  ......Pool thread [0]: Bytes in full message: [38]d - [26]h.
3  ......Pool thread [0]: Bytes in message (skipping size header): [34]d - [22]h.
4  ......Pool thread [0]: 22 00 00 00 05 26 0A 09 90 3E F0 CF 48 8D B3 D2 2E 11 3C 38 D1
5  00 00 00 00 01 00 00 00 FF FF FF FF FF FF FF FF 0D
6  ......Pool thread [0]: xx xx xx xx 05 26 0A 09 90 3E F0 CF 48 8D B3 D2 2E 11 3C 38 D1
7  00 00 00 00 01 00 00 00 FF FF FF FF FF FF FF FF 0D
8  ......Pool thread [0]: Op code = [PL_PROTOCOL_OPCODE_WRITE].
9  ......Pool thread [0]: uuid = [90090a26-f03e-48cf-8db3-d22e113c38d1].
10 ......Pool thread [0]: pld = [0].
11 ......Pool thread [0]: counter offset = [1].
12 ......Pool thread [0]: counter value = [18446744073709551615].
13 ......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
14 ...Pool thread [0] is sending...
15 ......Pool thread [0]: Bytes in full message: [9]d - [9]h.
16 ......Pool thread [0]: Bytes in message (skipping size header): [5]d - [5]h.
17 ......Pool thread [0]: 05 00 00 00 00 00 00 00 0D
18 ......Pool thread [0]: xx xx xx xx 00 00 00 00 0D
19 ......Pool thread [0]: Status = [PL_PROTOCOL_SUCCESS].
20 ......Pool thread [0]: Answer to op code = [PL_PROTOCOL_OPCODE_WRITE].
21 ......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
```

3.11.9 Complete Transaction Example

Assume the following program. Note that only relevant data is shown in the listing below.

```
1   Unsigned long long value = PL_MAX_COUNTER_VALUE;
2   const char *counters[5] = {
3      "Hello",
4      "World",
5      NULL,
6      "A",
7      "b"
8   };
9
10  pld = pl_open(
11     "Application in filesystem-less mode",
12     5,
13     counters,
14     &uuid
15  );
16
17  pl_ret = pl_write(
18     pld,
19     &value,
20     1
21  );
22
23  value = PL_MIN_COUNTER_VALUE;
24
25  pl_ret = pl_read(
26     pld,
27     &value,
28     1
29  );
30
31  pl_ret = pl_close(pld);
```

When compiled in debug mode, the sample agent prints – among other data – a HEX dump of the messages received and the messages sent. This is a facility that can be used to analyze the use of the PL protocol. The listing below shows the log for the previous, full transaction example. Note that to limit the log size, the sample agent was configured with a single worker thread in the pool.

```
pl_agent has started.
Parsing user input.
pl_agent version [2010.06.08].
Using PL helper version [2009.05.18].
Using PL version [2010.12.15(W)].
Initializing Windows socket system.
Agent is running on [10.24.0.35].
ADMIN port is [49252] and PL port is [49253].
Allocating thread pool data.
Creating synchronization objects.
Creating thread pool.
Agent has [1] thread(s) in the pool.
Creating admin port listener thread.
Creating pl port listener thread.
Pool thread [0] has started.
To interrupt the agent, type <CTRL>+<C>.
Pool thread [0] is waiting for main thread to be done.
Signaling main thread done.
Waiting for all threads to end.
Pool thread [0] has received the main thread done signal.
Pool thread [0] is waiting for a PL API call to serve.
Admin port listener thread has started.
Pl port listener thread has started.
Admin port listener thread is waiting for main thread to be done.
Pl port listener thread is waiting for main thread to be done.
Admin port listener thread has received the main thread done signal.
Pl port listener thread has received the main thread done signal.
Admin port listener thread is setting-up IPC.
Pl port listener thread is setting-up IPC.
...Admin port listener thread is initializing IPC data.
...Pl port listener thread is initializing IPC data.
...Admin port listener thread is setting-up socket IPC data.
...Pl port listener thread is setting-up socket IPC data.
...Admin port listener thread is resolving IPC address & port.
...Pl port listener thread is resolving IPC address & port.
...Admin port listener thread is attempting to create & bind IPC socket.
...Pl port listener thread is attempting to create & bind IPC socket.
...Pl port listener thread is listening on IPC bound socket.
...Admin port listener thread is listening on IPC bound socket.
...Admin port listener thread is accepting connections.
...Pl port listener thread is accepting connections.
...Pl port listener thread has received a request.
...Pl port listener thread is searching a thread in the pool to serve the request.
......Pl port listener thread is trying to lock pool thread [0].
.......Pl port listener thread has successfully locked pool thread [0].
...Pl port listener thread has triggered pool thread [0].
...Pl port listener thread is accepting connections.
Pool thread [0] is serving a PL API call.
...Pool thread [0] has received...
```

```
50  ......Pool thread [0]: Bytes in full message: [100]d - [64]h.
51  ......Pool thread [0]: Bytes in message (skipping size header): [96]d - [60]h.
52  ......Pool thread [0]: 60 00 00 00 01 05 00 00 00 23 00 00 00 41 70 70 6C 69 63 61 74
53  69 6F 6E 20 69 6E 20 66 69 6C 65 73 79 73 74 65 6D 2D 6C 65 73 73 20 6D 6F 64 65 05 00
54  00 00 48 65 6C 6C 6F 05 00 00 00 57 6F 72 6C 64 13 00 00 00 61 6E 6F 6E 79 6D 6F 75 73
55  5F 63 6F 75 6E 74 65 72 5F 32 01 00 00 00 41 01 00 00 00 62 0D
56  ......Pool thread [0]: xx xx xx xx 01 05 00 00 00 23 00 00 00 41 70 70 6C 69 63 61 74
57  69 6F 6E 20 69 6E 20 66 69 6C 65 73 79 73 74 65 6D 2D 6C 65 73 73 20 6D 6F 64 65 05 00
58  00 00 48 65 6C 6C 6F 05 00 00 00 57 6F 72 6C 64 13 00 00 00 61 6E 6F 6E 79 6D 6F 75 73
59  5F 63 6F 75 6E 74 65 72 5F 32 01 00 00 00 41 01 00 00 00 62 0D
60  ......Pool thread [0]: Op code = [PL_PROTOCOL_OPCODE_OPEN].
61  ......Pool thread [0]: Counters count = [5].
62  ......Pool thread [0]: Application name length = [35].
63  ......Pool thread [0]: Application name = [Application in filesystem-less mode].
64  ......Pool thread [0]: Counter [0] length = [5].
65  ......Pool thread [0]: Counter [0] name = [Hello].
66  ......Pool thread [0]: Counter [1] length = [5].
67  ......Pool thread [0]: Counter [1] name = [World].
68  ......Pool thread [0]: Counter [2] length = [19].
69  ......Pool thread [0]: Counter [2] name = [anonymous_counter_2].
70  ......Pool thread [0]: Counter [3] length = [1].
71  ......Pool thread [0]: Counter [3] name = [A].
72  ......Pool thread [0]: Counter [4] length = [1].
73  ......Pool thread [0]: Counter [4] name = [b].
74  ......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
75  ...Pool thread [0] is sending...
76  ......Pool thread [0]: Bytes in full message: [29]d - [1d]h.
77  ......Pool thread [0]: Bytes in message (skipping size header): [25]d - [19]h.
78  ......Pool thread [0]: 19 00 00 00 00 00 00 00 90 4D A5 D1 26 A5 BD 40 85 47 6A F8 CD
79  07 DA 18 00 00 00 00 0D
80  ......Pool thread [0]: xx xx xx xx 00 00 00 00 90 4D A5 D1 26 A5 BD 40 85 47 6A F8 CD
81  07 DA 18 00 00 00 00 0D
82  ......Pool thread [0]: Status = [PL_PROTOCOL_SUCCESS].
83  ......Pool thread [0]: Answer to op code = [PL_PROTOCOL_OPCODE_OPEN].
84  ......Pool thread [0]: uuid = [d1a54d90-a526-40bd-8547-6af8cd07da18].
85  ......Pool thread [0]: pld = [0].
86  ......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
87  ...Pool thread [0] is closing IPC.
88  ...Pool thread [0] has unlocked itself.
89  Pool thread [0] is waiting for a PL API call to serve.
90  ...Pl port listener thread has received a request.
91  ...Pl port listener thread is searching a thread in the pool to serve the request.
92  ......Pl port listener thread is trying to lock pool thread [0].
93  ......Pl port listener thread has successfully locked pool thread [0].
94  ...Pl port listener thread has triggered pool thread [0].
95  ...Pl port listener thread is accepting connections.
96  Pool thread [0] is serving a PL API call.
97  ...Pool thread [0] has received...
98  ......Pool thread [0]: Bytes in full message: [38]d - [26]h.
```

```
99  ......Pool thread [0]: Bytes in message (skipping size header): [34]d - [22]h.
100 ......Pool thread [0]: 22 00 00 00 05 90 4D A5 D1 26 A5 BD 40 85 47 6A F8 CD 07 DA 18
101 00 00 00 00 01 00 00 00 FF FF FF FF FF FF FF FF 0D
102 ......Pool thread [0]: xx xx xx xx 05 90 4D A5 D1 26 A5 BD 40 85 47 6A F8 CD 07 DA 18
103 00 00 00 00 01 00 00 00 FF FF FF FF FF FF FF FF 0D
104 ......Pool thread [0]: Op code = [PL_PROTOCOL_OPCODE_WRITE].
105 ......Pool thread [0]: uuid = [d1a54d90-a526-40bd-8547-6af8cd07da18].
106 ......Pool thread [0]: pld = [0].
107 ......Pool thread [0]: counter offset = [1].
108 ......Pool thread [0]: counter value = [18446744073709551615].
109 ......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
110 ...Pool thread [0] is sending...
111 ......Pool thread [0]: Bytes in full message: [9]d - [9]h.
112 ......Pool thread [0]: Bytes in message (skipping size header): [5]d - [5]h.
113 ......Pool thread [0]: 05 00 00 00 00 00 00 00 0D
114 ......Pool thread [0]: xx xx xx xx 00 00 00 00 0D
115 ......Pool thread [0]: Status = [PL_PROTOCOL_SUCCESS].
116 ......Pool thread [0]: Answer to op code = [PL_PROTOCOL_OPCODE_WRITE].
117 ......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
118 ...Pool thread [0] is closing IPC.
119 ...Pool thread [0] has unlocked itself.
120 Pool thread [0] is waiting for a PL API call to serve.
121 ...Pl port listener thread has received a request.
122 ...Pl port listener thread is searching a thread in the pool to serve the request.
123 ......Pl port listener thread is trying to lock pool thread [0].
124 ......Pl port listener thread has successfully locked pool thread [0].
125 ...Pl port listener thread has triggered pool thread [0].
126 ...Pl port listener thread is accepting connections.
127 Pool thread [0] is serving a PL API call.
128 ...Pool thread [0] has received...
129 ......Pool thread [0]: Bytes in full message: [30]d - [1e]h.
130 ......Pool thread [0]: Bytes in message (skipping size header): [26]d - [1a]h.
131 ......Pool thread [0]: 1A 00 00 00 04 90 4D A5 D1 26 A5 BD 40 85 47 6A F8 CD 07 DA 18
132 [redacted] 01 00 00 00 0D
133 ......Pool thread [0]: xx xx xx xx 04 90 4D A5 D1 26 A5 BD 40 85 47 6A F8 CD 07 DA 18
134 [redacted] 01 00 00 00 0D
135 ......Pool thread [0]: Op code = [PL_PROTOCOL_OPCODE_READ].
136 ......Pool thread [0]: uuid = [d1a54d90-a526-40bd-8547-6af8cd07da18].
137 ......Pool thread [0]: pld = [0].
138 ......Pool thread [0]: counter offset = [1].
139 ......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
140 ...Pool thread [0] is sending...
141 ......Pool thread [0]: Bytes in full message: [17]d - [11]h.
142 ......Pool thread [0]: Bytes in message (skipping size header): [13]d - [d]h.
143 ......Pool thread [0]: 0D 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF 0D
144 ......Pool thread [0]: xx xx xx xx 00 00 00 00 FF FF FF FF FF FF FF FF 0D
145 ......Pool thread [0]: Status = [PL_PROTOCOL_SUCCESS].
146 ......Pool thread [0]: Answer to op code = [PL_PROTOCOL_OPCODE_READ].
147 ......Pool thread [0]: Value = [18446744073709551615].
```

```
148 ......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
149 ...Pool thread [0] is closing IPC.
150 ...Pool thread [0] has unlocked itself.
151 Pool thread [0] is waiting for a PL API call to serve.
152 ...Pl port listener thread has received a request.
153 ...Pl port listener thread is searching a thread in the pool to serve the request.
154 ......Pl port listener thread is trying to lock pool thread [0].
155 ......Pl port listener thread has successfully locked pool thread [0].
156 ...Pl port listener thread has triggered pool thread [0].
157 ...Pl port listener thread is accepting connections.
158 Pool thread [0] is serving a PL API call.
159 ...Pool thread [0] has received...
160 ......Pool thread [0]: Bytes in full message: [26]d - [1a]h.
161 ......Pool thread [0]: Bytes in message (skipping size header): [22]d - [16]h.
162 ......Pool thread [0]: 16 00 00 00 03 90 4D A5 D1 26 A5 BD 40 85 47 6A F8 CD 07 DA 18
163 00 00 00 00 0D
164 ......Pool thread [0]: xx xx xx xx 03 90 4D A5 D1 26 A5 BD 40 85 47 6A F8 CD 07 DA 18
165 00 00 00 00 0D
166 ......Pool thread [0]: Op code = [PL_PROTOCOL_OPCODE_CLOSE].
167 ......Pool thread [0]: uuid = [d1a54d90-a526-40bd-8547-6af8cd07da18].
168 ......Pool thread [0]: pld = [0].
169 ......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
170 ...Pool thread [0] is sending...
171 ......Pool thread [0]: Bytes in full message: [9]d - [9]h.
172 ......Pool thread [0]: Bytes in message (skipping size header): [5]d - [5]h.
173 ......Pool thread [0]: 05 00 00 00 00 00 00 00 0D
174 ......Pool thread [0]: xx xx xx xx 00 00 00 00 0D
175 ......Pool thread [0]: Status = [PL_PROTOCOL_SUCCESS].
176 ......Pool thread [0]: Answer to op code = [PL_PROTOCOL_OPCODE_CLOSE].
177 ......Pool thread [0]: Last byte = [13] - [PL_PROTOCOL_EOR].
178 ...Pool thread [0] is closing IPC.
179 ...Pool thread [0] has unlocked itself.
180 Pool thread [0] is waiting for a PL API call to serve.
181 Received agent interrupt request from user [<CTRL>+<C>].
182 Signal handler is signaling pool thread [0].
183 Signal handler is signaling the ADMIN port listener thread.
184 Pool thread [0] was interrupted by user request.
185 ...Signal handler is setting-up IPC (ADMIN port).
186 Pool thread [0] has ended.
187 ...Signal handler is setting-up socket IPC data (ADMIN port).
188 ...Signal handler is resolving IPC address & port (ADMIN port).
189 ...Signal handler is attempting to connect to ADMIN listener thread (ADMIN port).
190 Admin port listener thread was interrupted by user request.
191 Admin port listener thread is tearing-down IPC.
192 Admin port listener thread has ended.
193 ...Signal handler is sending empty-message to ADMIN port listener thread (ADMIN port).
194 ...Signal handler is disconnecting from ADMIN port listener thread (ADMIN port).
195 ...Signal handler is tearing-down ADMIN IPC.
196 Signal handler is signaling the PL port listener thread.
```

```
...Signal handler is setting-up IPC (PL port).
...Signal handler is setting-up socket IPC data (PL port).
...Signal handler is resolving IPC address & port (PL port).
...Signal handler is attempting to connect to PL listener thread (PL port).
...Signal handler is sending empty-message to PL port listener thread (PL port).
Pl port listener thread was interrupted by user request.
...Signal handler is disconnecting from PL port listener thread (PL port).
Pl port listener thread is tearing-down IPC.
...Signal handler is tearing-down PL IPC.
Pl port listener thread has ended.
API Use Stats:
...pl_open:   [1] Call(s)
...pl_close:  [1] Call(s)
...pl_attach: [0] Call(s)
...pl_read:   [1] Call(s)
...pl_write:  [1] Call(s)
API Errors Stats:
...pl_open:   [0] Error(s)
...pl_close:  [0] Error(s)
...pl_attach: [0] Error(s)
...pl_read:   [0] Error(s)
...pl_write:  [0] Error(s)
Total error(s) count: [0]
Destroying synchronization objects.
De-allocating thread pool data.
De-initializing Windows socket system.
pl_agent has ended.
```

3.12 Network Configuration

When compiled in file system-less mode, the API uses two environment variables to specify the IPV4 *address* and *port number* in order to communicate with an agent.

3.12.1 IP Address

The IPV4 address environment variable is `PL_AGENT_ADDRESS`. If the variable does not exist, then `PL_DEFAULT_PL_AGENT_ADDRESS` (127.0.0.1) is used.

When the symbol `__PL_EXTRA_INPUT_CHECKS__` is defined, then the IPV4 address is checked to belong to one of the following classes:

- Class A: 000.000.000.000 to 127.255.255.255
- Class B: 128.000.000.000 to 191.255.255.255
- Class C: 192.000.000.000 to 223.255.255.255
- Class D: 224.000.000.000 to 239.255.255.255
- Class E: 240.000.000.000 to 255.255.255.255

3.12.2 Port Number

The port number environment variable is `PL_AGENT_PL_PORT`. If it does not exist, then `PL_DEFAULT_PL_AGENT_PL_PORT` (49253) is used.

When `__PL_EXTRA_INPUT_CHECKS__` symbol is defined, then the port number is checked to be between 1 and 65535.

NOTE

The configuration environment variables are checked each time a call to pl_open() or pl_attach() is issued. This allows running multiple agents on different addresses and/or ports, providing flexibility to dynamically load-balance a distributed system and enable room for scaling. Since the API guaranties that no data collision will happen if all PL data is aggregated on a single point, no special care has to be taken regarding where an agent can be started. The sole requirement is that the system hosting the PL sample agent has network access and a file system. An ad-hoc agent may wave this last requirement if it maintains the PL data in volatile memory, for example, and not permanently in a file system.

NOTE

The `productivity_link.h` defines the default environment variable names as:

- PL_DEFAULT_PL_AGENT_ADDRESS_ENVAR_NAME
- PL_DEFAULT_PL_PORT_ENVAR_NAME

NOTE

An instrumented application compiled with the `__PL_FILESYSTEM_LESS__` symbol defined will behave as if it was using the API compiled in file system-based mode if the following are true,

- *The sample agent is started on a system with an accessible file system.*
- *The sample agent was started without defining any configuration environment variables.*
- *The sample agent is started on the same system where the instrumented application runs.*

3.13 Using the API from Java*

The code below defines a Java* class (`ProductivityLink.java` in `iecsdk\src\core_api\interfaces`) shipped with the source code. This class is designed to simplify the use of the PL native functions stored in the dynamic libraries built from the source code (described below). This sample code can be replaced or amended as needed by the application developer.

```
1   //-------------------------------------------------------------------------
2   // Productivity Link JNI interface class
3   //-------------------------------------------------------------------------
4   import java.util.UUID;
5
6   public class ProductivityLink {
7
8      //-------------------------------------------------------------------------
9      // functions jni interfaces
10     //-------------------------------------------------------------------------
11     public native int pl_open(String pl_application_name, int pl_counters_count, String
12  pl_counters_names[], UUID puuid);
13     public native int pl_attach(String pl_config_file_name);
14     public native int pl_close(int pld);
15     public native int pl_write(int pld, Long counter, int counter_index);
16     public native int pl_read(int pld, Long counter, int counter_index);
17
18     //-------------------------------------------------------------------------
19     // enums
20     //-------------------------------------------------------------------------
21     public enum _pl_status {
22        PL_SUCCESS,
23        PL_FAILURE;
24     }
25
26     public enum _pl_failure {
27        PL_INVALID_DESCRIPTOR (0x10000000),
28        PL_BYPASSED,
29        PL_INVALID_PARAMETERS,
30        PL_SYNCHRONIZATION_FAILED,
31        PL_MISSING_DIRECTORY,
32        PL_NOT_A_DIRECTORY,
33        PL_DIRECTORY_CREATION_FAILED,
34        PL_DIRECTORY_ALREADY_EXISTS,
35        PL_PATH_NOT_FOUND,
36        PL_DESCRIPTOR_TABLE_FULL,
37        PL_DESCRIPTOR_TABLE_UNINITIALIZED,
38        PL_NON_GLOBAL_UUID_DESCRIPTOR,
39        PL_NON_GLOBAL_UUID_DESCRIPTOR_NO_ADDRESS,
40        PL_GLOBAL_UUID_DESCRIPTOR_CREATION_FAILED,
41        PL_GLOBAL_UUID_DESCRIPTOR_TO_STRING_FAILED,
```

```
42          PL_CRITICAL_FAILURE,
43          PL_CONFIG_FILE_GENERATION_FAILED,
44          PL_CONFIG_FILE_OPENING_FAILED,
45          PL_COUNTER_CREATION_FAILED,
46          PL_COUNTER_SEMAPHORE_CREATION_FAILED,
47          PL_COUNTER_ATTACH_FAILED,
48          PL_COUNTER_TO_STRING_FAILED,
49          PL_COUNTER_WRITE_FAILED,
50          PL_COUNTER_FILE_RESET_FILE_POINTER_FAILED,
51          PL_COUNTER_READ_FAILED,
52          PL_COUNTER_FILE_LOCK_FAILED,
53          PL_COUNTER_FILE_ALREADY_LOCKED,
54          PL_COUNTER_FILE_UNLOCK_FAILED,
55          PL_COUNTER_VALUE_OUT_OF_RANGE,
56          PL_NO_ERROR;
57
58          private int failure_code = 0;
59
60          private _pl_failure(int code) {
61             this.failure_code = code;
62          }
63
64          private _pl_failure() {
65             this.failure_code = 0;
66          }
67       }
68
69       //------------------------------------------------------------------------
70       // constants definitions
71       //------------------------------------------------------------------------
72       public static class _pl_constants
73       {
74          static final int PL_MAX_PRODUCTIVITY_LINKS = 10;
75          static final int PL_MAX_COUNTERS_PER_LINK = 250;
76          static final int PL_CONFIGURATION_FILE_APPLICATION_NAME_LINE = 1;
77          static final int PL_CONFIGURATION_FILE_UUID_STRING_LINE = 2;
78          static final int PL_CONFIGURATION_FILE_LOCATION_LINE = 3;
79          static final int PL_CONFIGURATION_FILE_COUNTERS_NUMBER_LINE = 4;
80       }
81
82       static {
83          System.loadLibrary("productivity_link_jni");
84       }
85    }
```

The sample code listed below shows how to use the Intel EC API from Java. The example creates a simple PL with four (4) counters and sets the values of two (2) of them (`ProductivityLinkDemo.java` in `iecsdk\src\samples\java_calling_sample` folder).

```
1   import java.util.UUID;
2   public class ProductivityLinkDemo {
3      public static void main(String[] args) {
4
5         int pld;
6         String application_name = "my_java_application";
7         String counter_names[] = {
8            "The Amazing A Counter",
9            "The not so bad B Counter",
10           "Counter C",
11           "Counter D"
12        };
13        UUID uuid = new UUID(0, 0);
14        Long val1 = new Long(987654321);
15        Long val2 = new Long(123456789);
16
17        //------------------------------------------------------------------
18        // create and open a PL
19        //------------------------------------------------------------------
20        ProductivityLink jpl = new ProductivityLink();
21        pld = jpl.pl_open(application_name, counter_names.length, counter_names, uuid);
22
23        //------------------------------------------------------------------
24        // write few counters
25        //------------------------------------------------------------------
26        jpl.pl_write(pld, val1, 0);
27        jpl.pl_write(pld, val2, 1);
28
29        //------------------------------------------------------------------
30        // close the PL
31        //------------------------------------------------------------------
32        jpl.pl_close(pld);
33     }
34  }
```

3.13.1 Running the Java Sample

The API is implemented as native code and needs to use an interface to interact with the Java Virtual Machine (JVM). Java uses the Java Native Interface (JNI) to do so (see http://java.sun.com/docs/books/jni/). You need to build the JNI dynamic library before running the Java sample provided with the Intel EC SDK.

Once the dynamic library with JNI support is built (`libproductivity_link_jni.so.1.0` or `productivity_link_java_dll`), the Java sample itself can be built.

Note that JNI is not the only possible interface, but is the one used by the API.

An important next step is to set the appropriate Java variables to include the dynamic library in the loader path, and the various class files generated into the class path. The listing below illustrates this process under Linux when you type the following commands.

```
cd /iecsdk/build/linux
make
```

```
Makefile Targets:
-----------------

all:.....................Build All Targets.
o:.......................Build PL Object.
a:.......................Build PL Static Library.
so:......................Build PL Shared Object.
j:.......................Build PL JNI Shared Object.
productivity_link:.......Build Dummy Application To Build PL.
productivity_link_helper:..Build Dummy Application To Build PL Helper.
consumer:................Build Porting Testing Applet's Consumer.
logger:..................Build Porting Testing Applet's Logger.
producer:................Build Porting Testing Applet's Producer.
esrv_client:.............Build ESRV Client Sample.
unit_tests:..............Build PL API Unit Tests.
benchmark:...............Build PL Benchmark (Not Available In 64bit Mode).
benchmark_bypass:........Build PL Benchmark Bypassed (Not Available In 64bit Mode).
java_sample:.............Build Java Sample Calling JNI Shared Object -- Depends On
Target (j).
linux_mem_info:..........Build Sample /proc/meminfo Data Collector.
linux_vmstat_info:.......Build Sample /proc/vmstat Data Collector.
linux_diskstat_info:..... Build Sample /proc/diskstats Data Collector.
linux_cpu_and_loadavg_info:.Build Sample /proc/loadavg, /proc/stats and cpu usage
(computed) Data Collector.
pl_csv_logger:...........Build The CSV Logger Companion Application.

clean:...................Delete All Build Files.
help:....................Print This Help Message.
```

```
29
30  Notes:
31  ------
32     Use The IECSDK_VERSION Symbol To Build Debug (debug) Or Release (release) Binaries.
33     Use The IECSDK_ADDRESSING Symbol To Build 32bit (32) Or 64bit (64) Binaries.
34     By Default, IECSDK_VERSION=debug and IECSDK_ADDRESSING=32 Are Used.
35
```

```
1  make clean
2  make IECSDK_ADDRESSING=64 java_sample
3  ll
```

```
1  total 75
2  -rwxrwxrwx 1 root 500 22157 Sep  9 13:56 Makefile
3  -rw-r--r-- 1 root 500   660 Sep 16 08:31 ProductivityLink$_pl_constants.class
4  -rw-r--r-- 1 root 500  3314 Sep 16 08:31 ProductivityLink$_pl_failure.class
5  -rw-r--r-- 1 root 500   976 Sep 16 08:31 ProductivityLink$_pl_status.class
6  -rw-r--r-- 1 root 500   768 Sep 16 08:31 ProductivityLink.class
7  -rw-r--r-- 1 root 500  1248 Sep 16 08:31 ProductivityLink.h
8  -rw-r--r-- 1 root 500   843 Sep 16 08:31 ProductivityLinkDemo.class
9  -rw-r--r-- 1 root 500  1207 Sep 16 08:31 ProductivityLink__pl_constants.h
10 -rw-r--r-- 1 root 500   292 Sep 16 08:31 ProductivityLink__pl_failure.h
11 -rw-r--r-- 1 root 500   289 Sep 16 08:31 ProductivityLink__pl_status.h
12 lrwxrwxrwx 1 root 500    31 Sep 16 08:31 libproductivity_link_jni.so ->
13 libproductivity_link_jni.so.1.0
14 lrwxrwxrwx 1 root 500    31 Sep 16 08:31 libproductivity_link_jni.so.1 ->
15 libproductivity_link_jni.so.1.0
16 -rwxr-xr-x 1 root 500 19864 Sep 16 08:31 libproductivity_link_jni.so.1.0
17 -rw-r--r-- 1 root 500 15228 Sep 16 08:31 productivity_link_jni.o
```

```
export LD_LIBRARY_PATH=.
java ProductivityLinkDemo
ll /opt/productivity_link/my_java_application_ce48b428-a9a1-4e3a-a60f-1e8bce2481bf/
```

```
total 12
-rwxr-xr-x 1 root root   0 Sep 16 08:49 Counter C
-rwxr-xr-x 1 root root   0 Sep 16 08:49 Counter D
-rwxr-xr-x 1 root root  11 Sep 16 08:49 The Amazing A Counter
-rwxr-xr-x 1 root root  11 Sep 16 08:49 The not so bad B Counter
-rwxr-xr-x 1 root root 816 Sep 16 08:49 pl_config.ini

```

NOTE

The example above is run on a 64-bit version of Linux and a 64-bit Java Virtual Machine. To avoid the error message `wrong ELF class: ELFCLASS32 (Possible cause: architecture word width mismatch)`*, it is important to build the 64-bit version of the JNI dynamic library (achieved by using* `IECSDK_ADDRESSING=64` *build option).*

NOTE

Some older compilers may require exporting MALLOC_CHECK_=0 to avoid the following error message:

```
*** glibc detected *** java: double free or corruption (fasttop):....
```

3.14 Using the API from .NET* in C#

Similarly to the JNI interface in Java, .NET* provides the *InteropServices* assembly to interface managed and unmanaged code. The code below defines a C# class (`ProductivityLink.cs`) shipped with the source code to simplify the use of the PL native functions.

The listing below shows the use of PL counters from C# code. This code sample assumes that this is a Windows environment, the DLL is named `productivity_link.dll`, and that DLL is visible to the application's binary at runtime. This code is provided with the SDK (`ProductivityLinkDemo.cs` in the `iecsdk\src\samples\csharp_calling_sample` folder).

```
1   using System;
2   using System.Collections.Generic;
3   using System.Runtime.InteropServices;
4
5   namespace ProductivityLinkDemo {
6
7      class Program {
8
9         //---------------------------------------------------------------------
10        // the entry point
11        //---------------------------------------------------------------------
12        [STAThread]
13        static void Main(string[] args) {
14
15           int pld;
16           string application_name = "my_C#_application";
17           string [] counter_names = {
18              "The Amazing A Counter",
19              "The not so bad B Counter",
20              "Counter C",
21              "Counter D"
22           };
23           Guid uuid = Guid.NewGuid();
24           ulong val1 = 987654321;
25           ulong val2 = 123456789;
26
27           //---------------------------------------------------------------------
28           // create and open a PL
29           //---------------------------------------------------------------------
30           pld = ProductivityLink.pl_open(application_name, counter_names.Length,
31  counter_names, ref uuid);
32
33           //---------------------------------------------------------------------
34           // write few counters
35           //---------------------------------------------------------------------
36           ProductivityLink.pl_write(pld, ref val1, 0);
```

```
37          ProductivityLink.pl_write(pld, ref val2, 1);
38
39          //---------------------------------------------------------------
40          // close the PL
41          //---------------------------------------------------------------
42          ProductivityLink.pl_close(pld);
43       }
44    }
45
46    public class ProductivityLink {
47
48       //---------------------------------------------------------------
49       // functions InteropServices interfaces
50       //---------------------------------------------------------------
51       [DllImport("productivity_link.dll", CharSet = CharSet.Ansi, EntryPoint =
52 "pl_open", ExactSpelling = false, CallingConvention = CallingConvention.Cdecl)]
53       public static extern int pl_open(string pl_application_name, int
54 pl_counters_count, string[] pl_counters_names, ref Guid puuid);
55       [DllImport("productivity_link.dll", CharSet = CharSet.Ansi, EntryPoint =
56 "pl_attach", ExactSpelling = false, CallingConvention = CallingConvention.Cdecl)]
57       public static extern int pl_attach(string pl_config_file_name);
58       [DllImport("productivity_link.dll", EntryPoint = "pl_close", ExactSpelling =
59 false, CallingConvention = CallingConvention.Cdecl)]
60       public static extern int pl_close(int pld);
61       [DllImport("productivity_link.dll", EntryPoint = "pl_read", ExactSpelling =
62 false, CallingConvention = CallingConvention.Cdecl)]
63       public static extern int pl_read(int pld, ref ulong counter, int counter_index);
64       [DllImport("productivity_link.dll", EntryPoint = "pl_write", ExactSpelling =
65 false, CallingConvention = CallingConvention.Cdecl)]
66       public static extern int pl_write(int pld, ref ulong counter, int
67 counter_index);
68    }}
```

3.14.1 Running the C# Sample

The API is implemented as native code and needs to use an interface. To run the C# sample, first build the dynamic library (`productivity_link.dll`). Once the dynamic library (`libproductivity_link.so.1.0` or `productivity_link.dll`) is built, build the C# sample itself.

All these steps are performed from the solution (`iesdk.sln` in `iecsdk\build\windows\iesdk` folder) shipped with the SDK. The listing below illustrates this process:

Rebuild the `productivity_link_dll` project.

```
1>------ Rebuild All started: Project: productivity_link_dll, Configuration: Release
Win32 ------
1>Deleting intermediate and output files for project 'productivity_link_dll',
configuration 'Release|Win32'
1>Compiling...
1>productivity_link.c
1>//------------------------------------------------------------------------------
1>// PL Build configuration report.
1>//------------------------------------------------------------------------------
1>NOTE: Building using _UNICODE.
1>NOTE: Building using UNICODE.
1>NOTE: Building using __PL_WINDOWS__.
1>NOTE: Building using _WINDLL.
1>NOTE: Building using _USRDLL.
1>NOTE: Building using __PL_WINDOWS_DLL_EXPORTS__.
1>NOTE: Building using __PL_GENERATE_INI__.
1>NOTE: Building using __PL_GENERATE_INI_VERSION_TAGGING__.
1>NOTE: Building using __PL_GENERATE_INI_BUILD_TAGGING__.
1>NOTE: Building using __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__.
1>NOTE: Building using __PL_BLOCKING_COUNTER_FILE_LOCK__.
1>Linking...
1>   Creating library D:\iecsdk\build\windows\iesdk\Release\productivity_link.lib and
object D:\iecsdk\build\windows\iesdk\Release\productivity_link.exp
1>Generating code
1>Finished generating code
1>Embedding manifest...
1>Performing Post-Build Event...
1>        1 file(s) copied.
1>Build log was saved at
"file://d:\iecsdk\build\windows\iesdk\productivity_link_dll\Release\BuildLog.htm"
1>productivity_link_dll - 0 error(s), 0 warning(s)
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

Upon completion, copy the `productivity_link_dll` into the run folder or in a folder in the system path (`iecsdk\build\windows\iesdk\csharp_calling_sample\bin\Release folder` for example).

Next, rebuild the `csharp_calling_sample` project.

```
------ Rebuild All started: Project: csharp_calling_sample, Configuration: Release Any
CPU ------
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Csc.exe /noconfig /nowarn:1701,1702
/errorreport:prompt /warn:4 /define:TRACE
/reference:C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Data.dll
/reference:C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.dll
/reference:C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Windows.Forms.dll
/reference:C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Xml.dll /debug:pdbonly
/optimize+ /out:obj\Release\csharp_calling_sample.exe /target:exe ProductivityLink.cs
ProductivityLinkDemo.cs Properties\AssemblyInfo.cs

Compile complete -- 0 errors, 0 warnings
csharp_calling_sample ->
D:\iecsdk\build\windows\iesdk\csharp_calling_sample\bin\Release\csharp_calling_sample.
exe
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========

```

```
dir c:\productivity_link\my_C#_application_af1d3db5-8559-407b-9503-3b8fdf0c3dd4
```

```
 Volume in drive C has no label.
 Volume Serial Number is 5CDF-F0B9

 Directory of c:\productivity_link\my_C#_application_af1d3db5-8559-407b-9503-
3b8fdf0c3dd4

09/16/2009  09:16 AM    <DIR>          .
09/16/2009  09:16 AM    <DIR>          ..
09/16/2009  09:16 AM                 0 Counter C
09/16/2009  09:16 AM                 0 Counter D
09/16/2009  09:16 AM               822 pl_config.ini
09/16/2009  09:16 AM                11 The Amazing A Counter
09/16/2009  09:16 AM                11 The not so bad B Counter
               5 File(s)            844 bytes
               2 Dir(s)  36,097,593,344 bytes free

```

3.15 Using the API from Objective-C

Objective-C is a development language under MacOS X. It is also a superset of the C language. Therefore, it is possible to use the API as-is from an Objective-C program. However, it may be useful to use the sample Objective-C wrapper class shipped with the SDK (`ProductivityLink.m`).

This section describes operations to execute under MacOS X. This is the only operating system for which the SDK supports the Objective-C class.

The listing below shows the use of PL counters from Objective-C code. This code is provided with the SDK (`ProductivityLinkDemo.m` in the `iecsdk/src/samples/objectivec_calling_sample` folder).

```
1   #include <assert.h>
2   #import <Foundation/NSObject.h>
3   #import "ProductivityLink.h"
4
5   int main() {
6
7      int ret = PL_FAILURE;
8      int pld = PL_INVALID_DESCRIPTOR;
9      unsigned int counters_count = 4;
10     const char *application_name = "my_Objective-C_application";
11     const char *counters_names[] = {
12        "The Amazing A Counter",
13        "The not so bad B Counter",
14        "Counter C",
15        "Counter D"
16     };
17     uuid_t uuid;
18     unsigned long long val1 = 987654321;
19     unsigned long long val2 = 123456789;
20
21     //------------------------------------------------------------------------
22     // create, allocate and initialize productivity link object
23     //------------------------------------------------------------------------
24     ProductivityLink *productivity_link = [
25        [ProductivityLink alloc]
26        init
27     ];
28
29     //------------------------------------------------------------------------
30     // create and open a PL
31     //------------------------------------------------------------------------
32     pld = [
33        productivity_link
34        PlOpen:
35        (char *)application_name andcounters_count:
36        counters_count andcounters_names:
```

```
37          counters_names andpuuid:
38          &uuid
39      ];
40      assert(pld != PL_INVALID_DESCRIPTOR);
41
42      //------------------------------------------------------------------------
43      // write few counters
44      //------------------------------------------------------------------------
45      ret = [
46          productivity_link
47          PlWrite:
48          pld andpvalue:
49          &val1 andoffset:
50          0
51      ];
52      assert(ret == PL_SUCCESS);
53      ret = [
54          productivity_link
55          PlWrite:
56          pld andpvalue:
57          &val2 andoffset:
58          1
59      ];
60      assert(ret == PL_SUCCESS);
61
62      //------------------------------------------------------------------------
63      // close the PL
64      //------------------------------------------------------------------------
65      ret = [
66          productivity_link
67          PlClose:
68          pld
69      ];
70      assert(ret == PL_SUCCESS);
71
72      //------------------------------------------------------------------------
73      // free memory
74      //------------------------------------------------------------------------
75      [productivity_link release];
76      return(PL_SUCCESS);
77  }
```

3.15.1 Running the Objective-C Sample

Build and run the Objective-C sample (`make objectivec_sample`, then `./objectivec_sample`). The listing below illustrates this process:

```
/bin/cp \
                /orssg-data/Development/iecsdk/src/core_api/productivity_link.c \
                /orssg-data/Development/iecsdk/src/core_api/productivity_link.m
/usr/bin/gcc \
                -D__PL_MACOSX__ -D__PL_GENERATE_INI__ -
D__PL_GENERATE_INI_VERSION_TAGGING__ -D__PL_GENERATE_INI_BUILD_TAGGING__ -
D__PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ -D__PL_BLOCKING_COUNTER_FILE_LOCK__ -
D__PL_EXTRA_INPUT_CHECKS__ -m32  -O2 -msse -Wall -fPIC -std=gnu99 -D_SVID_SOURCE -
D_REENTRANT -D_LIBC_REENTRANT \
                -framework Cocoa \
                -I/orssg-data/Development/iecsdk/src/core_api \
                -I/orssg-data/Development/iecsdk/src/core_api/interfaces \
                /orssg-
data/Development/iecsdk/src/core_api/interfaces/ProductivityLink.m \
                /orssg-
data/Development/iecsdk/src/samples/objectivec_calling_sample/ProductivityLinkDemo.m \
                /orssg-data/Development/iecsdk/src/core_api/productivity_link.m \
                -o objectivec_sample \
                -lpthread -ldl
/bin/rm -f \
                /orssg-data/Development/iecsdk/src/core_api/productivity_link.m

```

```
ll /opt/productivity_link/my_Objective-C_application_9ED2A7FB-CF5A-40B6-B1C0-
C7B63E582DE3/
```

```
 total 24
drwxr-xr-x  7 root  admin  238 Dec  2 08:37 .
drwxr-xr-x  3 root  admin  102 Dec  2 08:37 ..
-rwxr-xr-x  1 root  admin    0 Dec  2 08:37 Counter C
-rwxr-xr-x  1 root  admin    0 Dec  2 08:37 Counter D
-rwxr-xr-x  1 root  admin   11 Dec  2 08:37 The Amazing A Counter
-rwxr-xr-x  1 root  admin   11 Dec  2 08:37 The not so bad B Counter
-rwxr-xr-x  1 root  admin  844 Dec  2 08:37 pl_config.ini

```

3.16 API Unit Tests

The SDK is shipped with a set of API-level unit tests to check API functionality. It is recommended to run these tests if you modify any of the API files. The tests can also be used when development policy requires unit testing, and for organizations that decide to integrate the API source code into their application's source tree.

The tests use a simple and limited proprietary testing framework. However, it is fairly easy to extract the test cases and implement them in a different testing framework if necessary.

In this release of the SDK, there are 66 API unit tests. Each test has a status and an error code. Both must pass to consider the test as successful. All 66 tests must be successful to pass the API functional testing.

The listing at the end of this section shows a typical successful output of the API unit tests.

NOTE

Define the __PL_UNIT_TESTS__ symbol when compiling the unit tests. The normal behavior of the API, without this symbol defined, sets the system's last error only if an error occurs. Defining the symbol forces the API functions to set the system's last error to PL_NO_ERROR on success.

3.16.1 Reading the Test Output

Here is an example of the output list column headers. These are described in Table 4.

```
=============================================================================================================
[ T#  ] [  STATUS  ] [  ERROR   ] : { FUNCTION, CONDITION (VALUE) --> STATUS (VALUE), ERROR CODE, [EXTRA DATA] }
=============================================================================================================
```

Table 4. API Unit Test Output Field Descriptions

Column Header	Description
T#	Thi is the test number. This number is reproduced in the `unit_tests_framework.c` file (`main` function) as a comment. Use this number to trace the failing test function indicated.
STATUS	This is the test status (`STATUS`) and can be either `PASS` or `FAIL`. This is not the `PL_STATUS` returned by the tested API function, but the pass/fail of the unit test. For example, a unit test expected to fail is reported as `OK` if the tested API function returns `PL_FAILURE`.
ERROR	This is the error status (`ERROR`) and can be either `PASS` or `FAIL`. This is not the `PL_ERROR` set by the tested API function, but the pass/fail of the unit test error reporting mechanism. For example, a test expected to succeed is reported as `PASS` if the tested API function sets the system's last error code to `PL_NO_ERROR`. To pass, a unit test must always report `OK` for both the `STATUS` and `ERROR` columns.
Test Summary	This summarizes the unit test using the following convention: `{ FUNCTION, CONDITION (VALUE) --> STATUS (VALUE), ERROR CODE, [EXTRA DATA] }`. • `FUNCTION` is the tested API function name. • `CONDITION` describes the test object ("what is tested"). • `VALUE` is the tested object's value used by the unit test. Following the `-->` sign, is the expected result for the unit test. • `STATUS` is the expected status in the tested condition. • `VALUE` provides the expected `PL_SATATUS`. • `EXTRA DATA` is a debug parameter.

For example, unit test #017 checks that if `pl_close` is called when the PL table is not yet initialized (no prior `pl_open()` call for example), it returns `PL_FAILURE` status and sets the system's error code to `PL_DESCRIPTOR_TABLE_UNINITIALIZED` it.

```
================================================================================================================
[ T#  ] [  STATUS  ] [  ERROR   ] : { FUNCTION, CONDITION (VALUE) --> STATUS (VALUE), ERROR CODE, [EXTRA DATA] }
================================================================================================================
[ 001 ] [   PASS   ] [   PASS   ] : { pl_open, pl_counters_name (NULL) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 002 ] [   PASS   ] [   PASS   ] : { pl_open, puuid (NULL) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 003 ] [   PASS   ] [   PASS   ] : { pl_open, pl_counters_count (LOWER_THAN_0) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 004 ] [   PASS   ] [   PASS   ] : { pl_open, pl_counters_count (HIGHER_THAN_PL_MAX_COUNTERS_PER_LINK) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 005 ] [   PASS   ] [   PASS   ] : { pl_open, pl_counters_count (EQUAL_TO_PL_MAX_COUNTERS_PER_LINK) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
[ 006 ] [   PASS   ] [   PASS   ] : { pl_open, pl_counters_count (BETWEEN_0_AND_PL_MAX_COUNTERS_PER_LINK) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
[ 007 ] [   PASS   ] [   PASS   ] : { pl_open, application name (PRE_DEFINED_APPLICATION_NAME) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
[ 008 ] [   PASS   ] [   PASS   ] : { pl_open, application name (PRE_DEFINED_ANONYMOUS_APPLICATION_NAME) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
[ 009 ] [   PASS   ] [   PASS   ] : { pl_open, counter(s) names (PRE_DEFINED_NAMES) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
[ 010 ] [   PASS   ] [   PASS   ] : { pl_open, counter(s) names (PRE_DEFINED_ANONYMOUS_NAMES) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
[ 011 ] [   PASS   ] [   PASS   ] : { pl_open, pld (PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
        [   PASS   ] [   PASS   ] : { pl_open, pld (PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [1] }
        [   PASS   ] [   PASS   ] : { pl_open, pld (PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [2] }
        [   PASS   ] [   PASS   ] : { pl_open, pld (PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [3] }
        [   PASS   ] [   PASS   ] : { pl_open, pld (PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [4] }
        [   PASS   ] [   PASS   ] : { pl_open, pld (PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [5] }
        [   PASS   ] [   PASS   ] : { pl_open, pld (PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [6] }
        [   PASS   ] [   PASS   ] : { pl_open, pld (PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [7] }
        [   PASS   ] [   PASS   ] : { pl_open, pld (PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [8] }
        [   PASS   ] [   PASS   ] : { pl_open, pld (PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [9] }
[ 012 ] [   PASS   ] [   PASS   ] : { pl_open, pld (PL_MAX_PRODUCTIVITY_LINKS + 1) --> FAILURE (PL_FAILURE), PL_DESCRIPTOR_TABLE_FULL, [10] }
[ 013 ] [   PASS   ] [   PASS   ] : { pl_open, pl_application_name (illegal char) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 014 ] [   PASS   ] [   PASS   ] : { pl_open, pl_counter_name (illegal char) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 015 ] [   PASS   ] [   PASS   ] : { pl_open, pl_application_name (greater than PL_MAX_PATH) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 016 ] [   PASS   ] [   PASS   ] : { pl_open, pl_counter_name (greater than PL_MAX_PATH) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
----------------------------------------------------------------------------------------------------------------
[ 017 ] [   PASS   ] [   PASS   ] : { pl_close, pl_table_initialized (FALSE) --> FAILURE (PL_FAILURE), PL_DESCRIPTOR_TABLE_UNINITIALIZED, [0] }
[ 018 ] [   PASS   ] [   PASS   ] : { pl_close, pld (PL_INVALID_DESCRIPTOR) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 019 ] [   PASS   ] [   PASS   ] : { pl_close, pld (LOWER_THAN_0_PLD) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 020 ] [   PASS   ] [   PASS   ] : { pl_close, pld (HIGHER_THAN_PL_MAX_PRODUCTIVITY_LINKS_PLD) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 021 ] [   PASS   ] [   PASS   ] : { pl_close, pld (BETWEEN_0_AND_PL_MAX_PRODUCTIVITY_LINKS_PLD) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
----------------------------------------------------------------------------------------------------------------
```

```
[ 022 ] [   PASS   ] [   PASS   ] : { pl_attach, pld (NULL) --> FAILURE (PL_INVALID_DESCRIPTOR), PL_INVALID_PARAMETERS, [0] }
[ 023 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config (DOESN'T EXIST) --> FAILURE (PL_INVALID_DESCRIPTOR), PL_CONFIG_FILE_OPENING_FAILED, [0] }
[ 024 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config (DOES EXIST) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
[ 025 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file_name_length (greater_than_PL_MAX_PATH) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 026 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file_name (pl_folder_not_present) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 027 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file_name (pl_folder_name_incorrect) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 028 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file_name (not present) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 029 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file_name (UUID with three dashes) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 030 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file_name (UUID with illegal character) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 031 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file (counters count with illegal character) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 032 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file (counters count greater than PL_MAX_UNSIGNED_INTEGER_STRING) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 033 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file (counters name with incorrect PL folder) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 034 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file (counters name with incorrect path separator) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 035 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file (counters name with incorrect application name) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 036 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file (counters name with application name invalid characters) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 037 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file (counters name with incorrect UUID) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 038 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file (counters name with invalid UUID format) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 039 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file (counters count incorrect) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 040 ] [   PASS   ] [   PASS   ] : { pl_attach, pl_config_file (from Windows) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
-----------------------------------------------------------------------------------------------------------
[ 041 ] [   PASS   ] [   PASS   ] : { pl_read, pl_table_initialized (FALSE) --> FAILURE (PL_FAILURE), PL_DESCRIPTOR_TABLE_UNINITIALIZED, [0] }
[ 042 ] [   PASS   ] [   PASS   ] : { pl_read, pld (PL_INVALID_DESCRIPTOR) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 043 ] [   PASS   ] [   PASS   ] : { pl_read, pld (LOWER_THAN_0) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 044 ] [   PASS   ] [   PASS   ] : { pl_read, pld (HIGHER_THAN_PL_MAX_PRODUCTIVITY_LINKS) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 045 ] [   PASS   ] [   PASS   ] : { pl_read, p (NULL) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 046 ] [   PASS   ] [   PASS   ] : { pl_read, counter (LOWER_THAN_0) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 047 ] [   PASS   ] [   PASS   ] : { pl_read, counter (HIGHER_THAN_PL_MAX_COUNTERS_PER_LINK) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 048 ] [   PASS   ] [   PASS   ] : { pl_read, counter (HIGHER_THAN_ACTUAL_COUNTERS_IN_PL) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 049 ] [   PASS   ] [   PASS   ] : { pl_read, pld (BETWEEN_0_AND_PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
        [   PASS   ] [   PASS   ] : { pl_read, pld (BETWEEN_0_AND_PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [18446744073709551615] }
        [   PASS   ] [   PASS   ] : { pl_read, pld (BETWEEN_0_AND_PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [10] }
        [   PASS   ] [   PASS   ] : { pl_read, pld (BETWEEN_0_AND_PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [100] }
[ 050 ] [   PASS   ] [   PASS   ] : { pl_read, *p (PL_MAX_COUNTER_VALUE) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [18446744073709551615] }
[ 051 ] [   PASS   ] [   PASS   ] : { pl_read, *p (0 (a.k.a. PL_MIN_COUNTER_VALUE) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
[ 052 ] [   PASS   ] [   PASS   ] : { pl_read, counter data (CORRUPT DATA) --> FAILURE (PL_FAILURE), PL_COUNTER_VALUE_OUT_OF_RANGE, [0] }
```

```
[ 053 ] [   PASS   ] [   PASS   ] : { pl_read, counter data (OVERFLOW DATA) --> FAILURE (PL_FAILURE), PL_COUNTER_VALUE_OUT_OF_RANGE, [0] }
-------------------------------------------------------------------------------------------------------------
[ 054 ] [   PASS   ] [   PASS   ] : { pl_write, pl_table_initialized (FALSE) --> FAILURE (PL_FAILURE), PL_DESCRIPTOR_TABLE_UNINITIALIZED, [0] }
[ 055 ] [   PASS   ] [   PASS   ] : { pl_write, pld (PL_INVALID_DESCRIPTOR) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 056 ] [   PASS   ] [   PASS   ] : { pl_write, pld (LOWER_THAN_0) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 057 ] [   PASS   ] [   PASS   ] : { pl_write, pld (HIGHER_THAN_PL_MAX_PRODUCTIVITY_LINKS) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 058 ] [   PASS   ] [   PASS   ] : { pl_write, p (NULL) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 059 ] [   PASS   ] [   PASS   ] : { pl_write, counter (LOWER_THAN_0) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 060 ] [   PASS   ] [   PASS   ] : { pl_write, counter (HIGHER_THAN_PL_MAX_COUNTERS_PER_LINK) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 061 ] [   PASS   ] [   PASS   ] : { pl_write, counter (HIGHER_THAN_ACTUAL_COUNTERS_IN_PL) --> FAILURE (PL_FAILURE), PL_INVALID_PARAMETERS, [0] }
[ 062 ] [   PASS   ] [   PASS   ] : { pl_write, pld (BETWEEN_0_AND_PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
        [   PASS   ] [   PASS   ] : { pl_write, pld (BETWEEN_0_AND_PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [18446744073709551615] }
        [   PASS   ] [   PASS   ] : { pl_write, pld (BETWEEN_0_AND_PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [10] }
        [   PASS   ] [   PASS   ] : { pl_write, pld (BETWEEN_0_AND_PL_MAX_PRODUCTIVITY_LINKS) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [100] }
[ 063 ] [   PASS   ] [   PASS   ] : { pl_write, *p (PL_MAX_COUNTER_VALUE) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [18446744073709551615] }
[ 064 ] [   PASS   ] [   PASS   ] : { pl_write, *p (0 (a.k.a. PL_MIN_COUNTER_VALUE) --> SUCCESS (PL_SUCCESS), PL_NO_ERROR, [0] }
[ 065 ] [   PASS   ] [   PASS   ] : { pl_write, counter (CACHED-CACHE HIT) --> SUCCESS (PL_SUCCESS), PL_COUNTER_WRITE_CACHE_HIT, [0] }
        [   PASS   ] [   PASS   ] : { pl_write, counter (CACHED-CACHE HIT) --> SUCCESS (PL_SUCCESS), PL_COUNTER_WRITE_CACHE_HIT, [18446744073709551615] }
        [   PASS   ] [   PASS   ] : { pl_write, counter (CACHED-CACHE HIT) --> SUCCESS (PL_SUCCESS), PL_COUNTER_WRITE_CACHE_HIT, [10] }
        [   PASS   ] [   PASS   ] : { pl_write, counter (CACHED-CACHE HIT) --> SUCCESS (PL_SUCCESS), PL_COUNTER_WRITE_CACHE_HIT, [100] }
[ 066 ] [   PASS   ] [   PASS   ] : { pl_write, counter (NON CACHED-CACHE MISS) --> SUCCESS (PL_SUCCESS), PL_COUNTER_WRITE_CACHE_MISS, [0] }
        [   PASS   ] [   PASS   ] : { pl_write, counter (NON CACHED-CACHE MISS) --> SUCCESS (PL_SUCCESS), PL_COUNTER_WRITE_CACHE_MISS, [18446744073709551615] }
        [   PASS   ] [   PASS   ] : { pl_write, counter (NON CACHED-CACHE MISS) --> SUCCESS (PL_SUCCESS), PL_COUNTER_WRITE_CACHE_MISS, [10] }
        [   PASS   ] [   PASS   ] : { pl_write, counter (NON CACHED-CACHE MISS) --> SUCCESS (PL_SUCCESS), PL_COUNTER_WRITE_CACHE_MISS, [100] }
-------------------------------------------------------------------------------------------------------------
```

NOTE

By default, MacOS X limits the number of opened files per process to 256. To wave this limitation – and successfully run the API unit tests – use the `ulimit` *command before running the API unit tests. For example, to set the number of opened files per process to 2000, type the* `ulimit –n 2000` *command. The* `ulimit` *command may be needed to run Intel EC-instrumented applications using many counters. The* `ulimit –a –H` *command can be used to check system settings.*

NOTE

Operating System (OS) API failures are not covered in these unit tests. The assumption is that the OS APIs are validated by the OS vendor and are supposed to succeed. This doesn't mean of course that the API ignores OS API errors. They are just not covered in these unit tests.

NOTE

Be sure that you have access rights to the PL_FOLDER for the system on which you are running the unit tests. Deletion rights are required for the PL_FOLDER.

3.17 API Micro-benchmarks

Two micro-benchmarks (`benchmark` and `benchmark_bypass`) are shipped with the SDK to estimate the overhead introduced by the use of the `pl_read()` and `pl_write()` API functions. These micro-benchmarks provide *relative* measures of performance, not *absolute* measurements.

`benchmark` measures the average number of CPU cycles consumed by non-cached `pl_write` and `pl_read` API functions. The listing at the end of this section shows a typical output of `benchmark`.

`benchmark_bypass` measures the average number of CPU cycles consumed by the bypassed `pl_write` and `pl_read` API functions (code compiled with the `__PL_BYPASS__` symbol defined).

Table 5 summarizes results obtained on several Intel development systems. Note that the tested systems are vastly different (processors, frequency, memory, etc.) and therefore this data **cannot be used** to compare performance between operating systems. The sole role of this data is to show an example of data collected.

Table 5: Sample micro-benchmark results

Operating System	pl_write	pl_read
Microsoft Windows* XP version 2002 SP3	15,670.2	10,737.4
Linux* 2.6.16.60-0.21-smp	22,893.2	10,918.0
Darwin* 9.5.0 Darwin Kernel Version 9.5.0	35,645.7	32,411.2

NOTE

These micro-benchmarks are designed to work in 32-bit environments only.

```
1  ************************************************************
2  * This micro-benchmark uses RDTSC / _IA64_REG_AR_ITC to    *
3  * evaluate the processor cycles required to carry-out the  *
4  * pl_write() and pl_read() Productivity Link API calls     *
5  * (     1000000 write and read operations are performed    *
6  * in sequence).                                           *
7  *                                                         *
8  * This method is imperfect as it will catch cycles        *
9  * consumed by interrupt routines.  Therefore, consider     *
10 * numbers as indications, not exact values.               *
11 * When running the benchmark, stop all other applications. *
12 *                                                         *
13 * When compiling the benchmark, deactivate all compiler   *
14 * optimizations.  Run in single processor configuration.  *
15 * Deactivate all frequency and voltage adjustment         *
16 * features before running this micro-benchmark.           *
17 ************************************************************
18
19 Opening Test PL:
20 Measuring Base Data       : ********** (     1375353 Cycles)
21 Measuring pl_write() Data : ********** (  2224668267 Cycles)
22 Measuring pl_read() Data  : ********** (  1101083940 Cycles)
23 Closing Test PL.
24
25 Test Report:
26 =============
27
28 Average CPU cycles per pl_write() call :              22256.
29 Average CPU cycles per pl_read() call  :              11033.
30
31 Notes:
32 ======
33  *  A 2.0 GHz processor with two cores processes ~4 billion
34 instructions per second (~2 billion per core).   Allocation
35 of system CPU cycles under the following conditions:
36 1000 pl_write() calls per second:                   ~0.556%
37 1000 pl_read() calls per second:                    ~0.276%
38
39  *  1.0 GHz cycle = 1.0 nanosecond = 1.0 x 10-9 second.
40                10 nanoseconds   =   1.0 x 10-5 millisecond.
41               100 nanoseconds   =       0.0001 millisecond.
42             1,000 nanoseconds   =        0.001 millisecond.
43            10,000 nanoseconds   =         0.01 millisecond.
44           100,000 nanoseconds    =          0.1 millisecond.
45         1,000,000 nanoseconds    =            1 millisecond.
46
47 ************************************************************
48 *               End of micro-benchmark                     *
49 ************************************************************
```

3.18 Error Code Values

Table 6 below provides a listing of the SDK error codes, including their decimal and hexadecimal values. These error codes are defined in the source code provided with the SDK.

Table 6: Error code values

Error Codes	Decimal	Hexadecimal
PL_INVALID_DESCRIPTOR	268435456	10000000
PL_BYPASSED[a]	268435457	10000001
PL_INVALID_PARAMETERS	268435458	10000002
PL_SYNCHRONIZATION_FAILED	268435459	10000003
PL_MISSING_DIRECTORY	268435460	10000004
PL_NOT_A_DIRECTORY	268435461	10000005
PL_DIRECTORY_CREATION_FAILED	268435462	10000006
PL_DIRECTORY_ALREADY_EXISTS	268435463	10000007
PL_PATH_NOT_FOUND	268435464	10000008
PL_DESCRIPTOR_TABLE_FULL	268435465	10000009
PL_DESCRIPTOR_TABLE_UNINITIALIZED	268435466	1000000a
PL_NON_GLOBAL_UUID_DESCRIPTOR	268435467	1000000b
PL_NON_GLOBAL_UUID_DESCRIPTOR_NO_ADDRESS	268435468	1000000c
PL_GLOBAL_UUID_DESCRIPTOR_CREATION_FAILED	268435469	1000000d
PL_GLOBAL_UUID_DESCRIPTOR_TO_STRING_FAILED	268435470	1000000e
PL_CRITICAL_FAILURE	268435471	1000000f
PL_CONFIG_FILE_GENERATION_FAILED	268435472	10000010
PL_CONFIG_FILE_OPENING_FAILED	268435473	10000011
PL_COUNTER_CREATION_FAILED	268435474	10000012
PL_COUNTER_SEMAPHORE_CREATION_FAILED	268435475	10000013
PL_COUNTER_ATTACH_FAILED	268435476	10000014
PL_COUNTER_TO_STRING_FAILED	268435477	10000015
PL_COUNTER_WRITE_FAILED	268435478	10000016
PL_COUNTER_FILE_RESET_FILE_POINTER_FAILED	268435479	10000017
PL_COUNTER_READ_FAILED	268435480	10000018
PL_COUNTER_FILE_LOCK_FAILED	268435481	10000019
PL_COUNTER_FILE_ALREADY_LOCKED	268435482	1000001a
PL_COUNTER_FILE_UNLOCK_FAILED	268435483	1000001b
PL_COUNTER_VALUE_OUT_OF_RANGE	268435484	1000001c
PL_OUT_OF_MEMORY	268435485	1000001d
PL_OUT_OF_BUFFER_SPACE	268435486	1000001e
PL_BLOCKING_PL_READ_INSTANCE_CREATION_FAILED	268435487	1000001f
PL_BLOCKING_PL_READ_INSTANCE_DESTRUCTION_FAILED	268435488	10000020
PL_BLOCKING_PL_READ_HANDLE_CREATION_FAILED	268435489	10000021

Error Codes	Decimal	Hexadecimal
PL_BLOCKING_PL_READ_HANDLE_DESTRUCTION_FAILED	268435490	10000022
PL_BLOCKING_PL_READ_HANDLE_RENEWING_FAILED	268435491	10000023
PL_BLOCKING_PL_READ_WAITING_NOTIFICATION_FAILED	268435492	10000024
PL_FILESYSTEM_LESS_REMOTE_CRITICAL_FAILURE	268435493	10000025
PL_FILESYSTEM_LESS_INITIALIZATION_FAILED	268435494	10000026
PL_FILESYSTEM_LESS_NETWORK_ADDRESS_RESOLUTION_FAILED	268435495	10000027
PL_FILESYSTEM_LESS_SOCKET_FAILED	268435496	10000028
PL_FILESYSTEM_LESS_CLOSE_SOCKET_FAILED	268435497	10000029
PL_FILESYSTEM_LESS_CONNECTION_FAILED	268435498	1000002a
PL_FILESYSTEM_LESS_SEND_FAILED	268435499	1000002b
PL_FILESYSTEM_LESS_RECV_FAILED	268435500	1000002c
PL_FILESYSTEM_LESS_INVALID_IPV4_ADDRESS	268435501	1000002d
PL_FILESYSTEM_LESS_INVALID_PORT	268435502	1000002e
PL_COUNTER_WRITE_CACHE_HIT[a]	268435503	1000002f
PL_COUNTER_WRITE_CACHE_MISS[a]	268435504	10000030
PL_NO_ERROR[a]	268435505	10000031

[a] Non-error codes

This page intentionally blank.

4 SDK Installation

4.1 PL Storage Locations

The location where PL data is maintained is different on different types of systems. Table 7 lists locations of different OSs.

Table 7: PL storage locations

Target Operating System	Preprocessor Defines	PL_FOLDER
Windows*	__PL_WINDOWS__	C:\productivity_link
Linux*	__PL_LINUX__	/opt/productivity_link
Solaris*	__PL_SOLARIS__	/opt/productivity_link
MacOS X*	__PL_MACOSX__	/opt/productivity_link
MeeGo*	__PL_LINUX__	/opt/productivity_link

Future revisions of the SDK may provide alternate methods to store PL data. Therefore, applications must avoid direct references to the PL_FOLDER location, with the following exception:

- During application installation, the PL_FOLDER directory as indicated above must be created by the user or system administrator.

In the exception case above, the assumption for the PL_FOLDER value should be clearly identified in any installation utility source code to facilitate modifications later if necessary.

⚠ CAUTION

Before running an instrumented application on a given system, the PL_FOLDER must be created, as specified in Table 7. This is best done during application installation.

NOTE

The user should not place files of any type in the PL_FOLDER. Do not store the SDK tools and source code in this location.

4.2 Installing the SDK

The SDK is shipped as a compressed file (zip). To install the SDK, do the following:

10. Copy the compressed file to the installation location.
11. Decompress the installation file. Section 4.3 lists the SDK tree structure.
12. On the system on which the instrumented code will be run and tested, create the PL_FOLDER.

Refer to Table 7 to find the proper folder name for each OS.

NOTE

Security should be managed as usual in production environments. Administrators should set-up the appropriate Access Control Lists (ACL) to allow and deny PL_FOLDER access to users and applications, except where specific access rights are needed. If remote PL counter data collection is desired, extra steps may be required to map users and user-rights between various systems. Instrumented applications need write access to the PL_FOLDER.

NOTE

PL_FOLDER is the symbolic name of the folder storing all the PLs created by instrumented applications. The actual name of the folder is given in Table 7.

4.3 Folder structure

The SDK has the folder structure listed below. Following the directory structure is a description of the major folders.

```
\iecsdk
├──bin
│   ├──companion_applications
│   │   └──pl_gui_monitor
│   │       └──windows
│   │       │   ├──X64
│   │       │   └──X86
│   │       └──linux
│   │       │   ├──X64
│   │       │   └──X86
│   │       └──macosx
│   │       │   ├──X64
│   │       │   └──X86
│   │       └──solaris
│   │           ├──X64
│   │           └──X86
│   ├──energy_server
│   │   ├──linux
│   │   │   ├──X64
│   │   │   └──X86
│   │   ├──macosx
│   │   │   ├──X64
│   │   │   └──X86
│   │   ├──meego
│   │   │   └──X86
│   │   ├──solaris
│   │   │   ├──X64
│   │   │   └──X86
│   │   └──windows
│   │       ├──X64
│   │       └──X86
│   ├──pl_scripting_tools
│   │   ├──linux
│   │   │   ├──X64
│   │   │   └──X86
│   │   ├──macosx
│   │   │   ├──X64
│   │   │   └──X86
│   │   ├──meego
│   │   │   └──X86
│   │   ├──solaris
│   │   │   ├──X64
│   │   │   └──X86
│   │   └──windows
│   │       ├──X64
│   │       └──X86
│   ├──temperature_server
│   │   ├──linux
│   │   │   ├──X64
│   │   │   └──X86
│   │   ├──macosx
│   │   │   ├──X64
│   │   │   └──X86
│   │   ├──meego
│   │   │   └──X86
│   │   ├──solaris
│   │   │   ├──X64
│   │   │   └──X86
│   │   └──windows
```


X64
X86
windows_interoperability_tools
windows
X64
X86
build
linux
macosx
meego
solaris
windows
iecsdk
benchmark
benchmark_bypass
consumer
core_api_unit_tests
cpu_use_histogram
csharp_calling_sample
debug
iesdk
java_calling_sample
logger
pl_csv_logger
producer
productivity_link
productivity_link_dll
productivity_link_helper
productivity_link_java_dll
sample_logger
threading
doc
license
src
companion_applications
pl_csv_logger
core_api
benchmark
interfaces
porting_test_applets
common
consumer
logger
producer
unit_tests
core_api_unit_tests
energy_server
helper_api
samples
cluster_energy_efficiency
cpu_use_histogram
csharp_calling_sample
energy
esrv_client
java_calling_sample
linux_system_infos
mti_sample
objectivec_calling_sample
pl2ganglia
sample_logger
start_esrv
threading
uuid_variant_sample
temperature_server
utils
device_driver_kit
build
linux
macosx
meego
solaris
windows
iecddk
energy_meter_driver
iecddk
release
temperature_meter_driver
doc
src
energy_meter_driver
temperature_meter_driver

4.3.1 Bin

This folder contains the release binaries of the SDK. Binaries stored in this branch are the ones marked as B (as binary) in

Table 2 (distribution column). This folder branches into the following binary groups:

- Companion applications
- Energy and temperature servers
- Scripting tools
- Windows interoperability tools

Each branch may have folders to structure binaries. However, at the end, the binaries are regrouped by supported operating systems (Linux, MacOS X, Solaris 10, and Windows). Each OS folder has 32 and 64-bit terminal folders containing the corresponding binary (if supported) (see

Table 2).

4.3.2 Build

This folder contains the Intel® EC building tools. As described in Section 0, a Microsoft* Visual Studio* 2005 solution and several Makefiles are provided. This folder is organized by supported operating systems (Linux, MacOS X, Solaris 10, and Windows). Each non-Windows OS folder contains a single Makefile. The `windows` folder contains an `iecsdk` subfolder which contains all the project folders and the `iesdk.sln` solution file.

Binaries generated by the Makefiles are stored in the same folder. Binaries generated by the solution are stored as follows:

- Debug 32-bit in `iecsdk\build\windows\iesdk\debug`
- Release 32-bit in `iecsdk\build\windows\iesdk\release`
- Debug 64-bit in `iecsdk\build\windows\iesdk\x64\debug`
- Release 64-bit in `iecsdk\build\windows\iesdk\x64\release`

The csharp_calling_sample binaries are respectively stored in `iecsdk\build\windows\iesdk\csharp_calling_sample\bin\Debug` and `iecsdk\build\windows\iesdk\csharp_calling_sample\bin\Release`.

4.3.3 Doc

This folder contains the SDK manual pdf files. There are three manuals:

- *Intel® Energy Checker SDK Companion Applications User Guide*
- *Intel® Energy Checker SDK Device Driver Kit User Guide*
- *Intel® Energy Checker SDK User Guide* (this manual)

4.3.4 License

This folder contains the Intel® Energy Checker SDK license agreement.

4.3.5 Src

This folder contains the SDK source code. Sources stored in this branch are the ones marked as S (as source) in

Table 2 (distribution column). This folder branches into source groups:

- Companion applications
- Energy and temperature servers
- Core_api
- Helper_api
- Samples

Each branch may have folders to structure source files.

4.3.6 Utils

This folder contains the SDK ESRV and TSRV device driver kit. The folder segregates the build tools and the source code required to build a support library for power and energy plus temperature and relative humidity measurement devices. Please refer to the *Intel® Energy Checker Device Driver Kit User Guide* for details.

This page intentionally blank.

5 Code Building

This chapter describes the processes for the following:

- Integrating and building applications using the Intel Energy Checker API.
- Building various dynamic link libraries or shared libraries.
- Building other SDK components whose source code is provided in the SDK.

The SDK is shipped with a solution for Microsoft Visual Studio* 2005 and a set of Makefiles (one per supported non-Windows* OS). Table 8 lists the build file locations for each supported OS.

Table 8: Build files locations

Operating System	SDK Location
Windows*	`iecsdk\build\windows\iesdk\iesdk.sln`
Linux*	`iecsdk\build\linux\Makefile`
Meego*	`Iecsdk\build\meego\Makefile`
Solaris* 10	`iecsdk\build\solaris\Makefile`
MacOS* X	`iecsdk\build\macosx\Makefile`

5.1 Visual Studio* Solution

The `iesdk.sln` solution (in folder `iecsdk\build\windows\iesdk`) includes the following projects (targets):

- benchmark
- benchmark_bypass
- cluster_energy_efficiency
- consumer
- core_api_unit_tests
- cpu_use_histogram
- csharp_calling_sample
- energy
- esrv_sample
- iecsdk.ncb
- iecsdk.sln
- java_calling_sample
- logger
- mti_sample
- pl2ganglia
- pl_agent
- pl_csv_logger

- producer
- productivity_link
- productivity_link_dll
- productivity_link_helper
- productivity_link_java_dll
- sample_logger
- threading
- uuid_variant_sample

5.1.1 Configure the Solution

Developers should not need to modify the configurations of the solution's projects'. The sole configuration that might need to be changed is the pre-build section for the productivity_link_java_dll project.

The listing below shows the pre-build step element to be modified (line 4) in bold text. Figure 14 shows the location of the pre-build step settings in Microsoft Visual Studio 2005. Select `All Configurations` and `All Platforms` in the *Configuration and Platform* drop-down lists to apply the change once.

```
1  javac "..\..\..\..\src\core_api\interfaces\ProductivityLink.java"
2  del "..\..\..\..\src\core_api\interfaces\ProductivityLink.h" 2>nul
3  javah -jni  -d "..\..\..\..\src\core_api\interfaces" -classpath
4  "..\..\..\..\src\core_api\interfaces;C:\Program
5  Files\Java\jre1.6.0_03lib\ext\QTJava.zip" ProductivityLink
```

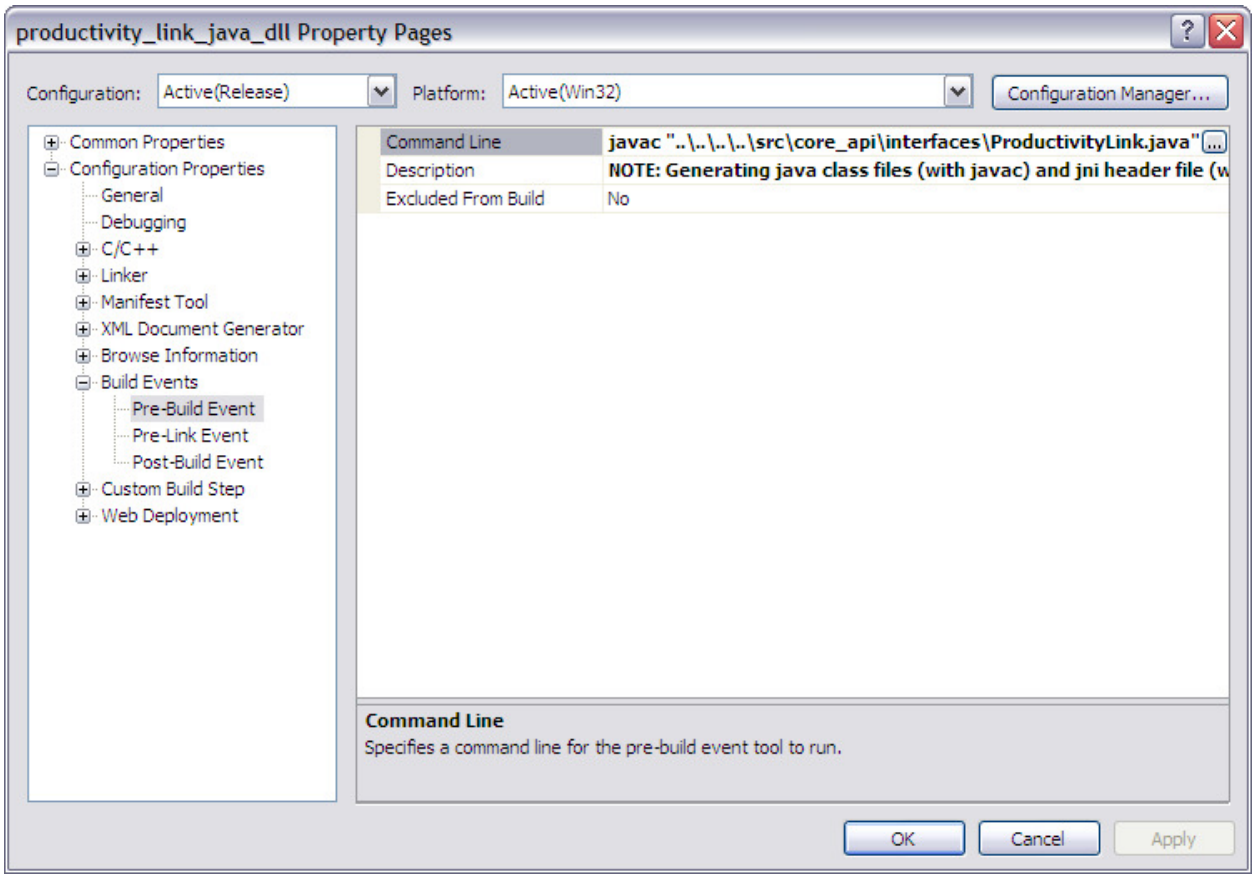

Figure 14: Pre-build configuration of the productivity_link_java_dll project.

5.2 Makefile (non-Windows Environments)

The following listing shows the available targets of each Makefile (all supported non-Windows OSes). To obtain this list, simply type `make` followed by return. To build a specific target, type `make <target>` followed by return. The target list might differ between various non-Windows environments.

By default, the Makefile is set to build 32-bit binaries. To change this behavior, set the `IECSDK_VERSION` variable (`debug` to build a debug version and `release` to build a release version).

Similarly, to build a 64-bit version, set the `IECSDK_ADDRESSING` variable (`64` for 64-bit and `32` for 32-bit).

For example, to build a 64-bit debug version of pl_csv_logger, type `make ADDRESSING=64 VERSION=debug pl_csv_logger`.

```
1
2   Makefile Targets:
3   -----------------
4
5   all:.....................Build All Targets.
6   o:.......................Build PL Object.
7   a:.......................Build PL Static Library.
8   so:......................Build PL Shared Object.
9   j:.......................Build PL JNI Shared Object.
10  productivity_link:.......Build Dummy Application To Build PL.
11  productivity_link_helper:..Build Dummy Application To Build PL Helper.
12  consumer:................Build Porting Testing Applet's Consumer.
13  logger:..................Build Porting Testing Applet's Logger.
14  producer:................Build Porting Testing Applet's Producer.
15  esrv_client:.............Build ESRV Client Sample.
16  unit_tests:..............Build PL API Unit Tests.
17  benchmark:...............Build PL Benchmark (Not Available In 64bit Mode).
18  benchmark_bypass:........Build PL Benchmark Bypassed (Not Available In 64bit Mode).
19  java_sample:.............Build Java Sample Calling JNI Shared Object -- Depends On
20  Target (j).
21  linux_mem_info:..........Build Sample /proc/meminfo Data Collector.
22  linux_vmstat_info:.......Build Sample /proc/vmstat Data Collector.
23  linux_diskstat_info:..... Build Sample /proc/diskstats Data Collector.
24  linux_cpu_and_loadavg_info:.Build Sample /proc/loadavg, /proc/stats and cpu usage
25  (computed) Data Collector.
26  pl_csv_logger:...........Build The CSV Logger Companion Application.
27
28  clean:...................Delete All Build Files.
29  help:....................Print This Help Message.
30
31  Notes:
32  ------
33     Use The IECSDK_VERSION Symbol To Build Debug (debug) Or Release (release) Binaries.
34     Use The IECSDK_ADDRESSING Symbol To Build 32bit (32) Or 64bit (64) Binaries.
35     By Default, IECSDK_VERSION=debug and IECSDK_ADDRESSING=32 Are Used.
36
```

5.2.1 Configure the Makefile

Developers may need to configure the Makefile to reflect their specific build system. The listing below is an extract of the Linux Makefile. Configuration-wise, each Makefile is equal. The first configuration should take place at line 4 by updating the `IECSDK_ROOT` variable to match the location where the SDK files are copied. The second configuration should take place between lines 31 and 37.

Update the `JAVA_PATH`, `JAVA_INCLUDE_PATH` and `JAVA_LINUX_INCLUDE_PATH` variables to reflect the JDK installation on the developer's system. Optionally, variables between lines 42 to 64 can be set to match special tools required during the build process.

```
1  #-------------------------------------------------------------------------------
2  # IEC SDK directory structure and useful defines.
3  #-------------------------------------------------------------------------------
4  IECSDK_ROOT=/orssg-data/Development/iecsdk
5  IECSDK_BIN_ROOT=$(IECSDK_ROOT)/bin
6  IECSDK_DEP_ROOT=$(IECSDK_ROOT)/dep
7  IECSDK_SRC_ROOT=$(IECSDK_ROOT)/src
8  IECSDK_DEP_UTILS=$(IECSDK_ROOT)/utils
9
10 IECSDK_CORE_API=$(IECSDK_SRC_ROOT)/core_api
11 IECSDK_HELPER_API=$(IECSDK_SRC_ROOT)/helper_api
12 IECSDK_SAMPLES=$(IECSDK_SRC_ROOT)/samples
13 IECSDK_CORE_API_PORTING=$(IECSDK_CORE_API)/porting_test_applets
14 IECSDK_CORE_API_UNIT_TESTS=$(IECSDK_CORE_API)/unit_tests/core_api_unit_tests
15 IECSDK_CORE_API_BENCHMARK=$(IECSDK_CORE_API)/benchmark
16 IECSDK_COMPANION=$(IECSDK_SRC_ROOT)/companion_applications
17 IECSDK_ESRV=$(IECSDK_SRC_ROOT)/energy_server
18
19 #-------------------------------------------------------------------------------
20 # IEC SDK configuration.
21 #-------------------------------------------------------------------------------
22 #IECSDK_VERSION=debug
23 IECSDK_VERSION=release
24 IECSDK_ADDRESSING=32
25 #IECSDK_ADDRESSING=64
26
27 #-------------------------------------------------------------------------------
28 # IEC SDK JAVA directory structure.
29 #-------------------------------------------------------------------------------
30 ifeq ($(IECSDK_ADDRESSING), 32)
31    JAVA_PATH=/usr/java/jdk1.6.0_13/bin
32    JAVA_INCLUDE_PATH=/usr/java/jdk1.6.0_13/include
33    JAVA_LINUX_INCLUDE_PATH=/usr/java/jdk1.6.0_13/include/linux
34 else
35    JAVA_PATH=/usr/java/jdk1.6.0_13/bin
36    JAVA_INCLUDE_PATH=/usr/java/jdk1.6.0_13/include
37    JAVA_LINUX_INCLUDE_PATH=/usr/java/jdk1.6.0_13/include/linux
```

```
38 endif
39
40 #-------------------------------------------------------------------------
41 # IEC SDK tools.
42 #-------------------------------------------------------------------------
43 IECSDK_CLEAR=/usr/bin/clear
44 IECSDK_CC=/usr/bin/gcc
45 IECSDK_CPP=/usr/bin/g++
46 IECSDK_LD=/usr/bin/ld
47 IECSDK_LDD=/usr/bin/ldd
48 IECSDK_NM=/usr/bin/nm
49 IECSDK_LN=/bin/ln
50 IECSDK_LL=/bin/ls -al
51 IECSDK_OBJDUMP=/usr/bin/objdump
52 IECSDK_STRIP=/usr/bin/strip
53 IECSDK_RM_F=/bin/rm -f
54 IECSDK_CP=/bin/cp
55 IECSDK_MV=/bin/mv
56 IECSDK_MKDIR=/bin/mkdir -p
57 IECSDK_TAR=/bin/tar -cvf
58 IECSDK_GZIP=/bin/gzip
59 IECSDK_CHAMOD=/bin/chmod
60 IECSDK_MAKE=/usr/bin/make
61 IECSDK_AR=/usr/bin/ar
62 IECSDK_ECHO=/bin/echo
63 IECSDK_GREP=/bin/grep
64 IECSDK_JAVAC=$(JAVA_PATH)/javac
65 IECSDK_JAVAH=$(JAVA_PATH)/javah
66
```

5.3 Compiling

To compile and use the Intel Energy Checker API code, add the two files (`productivity_link.c` and `productivity_link.h`) to a project and re-build it. The code has been tested with the Intel® C/C++ compiler 11.1.040, Microsoft Visual Studio 2005, GCC 4.1.2 on Linux, GCC 4.0.1 on MacOS X, and GCC 3.4.3 on Solaris 10. It should work with any C/C++ compiler.

NOTE

See Table 3 for a list of mandatory compilation symbols.

NOTE

Refer to the project files and Makefiles for building details.

To call the API from another language/environment, use the C linkage interface as directed by programming language being used. For example, see Section 0 and Section 0 for information on how to use the API from Java and C# .NET.

NOTE

If the calling language requires a dynamic library with stdcall linkage interface, then update the linker option in the project's settings.

This page intentionally blank.

6 Scripting Tools

Not all users have access to the source code of their applications. This is especially true in production environments. Still, it is possible to export and import counters for many applications, provided that there is a predictable way to derive that information.

The SDK provides four scripting tools to access Productivity Link counters:

- pl_open
- pl_write
- pl_read
- pl_desc

NOTE

There are no scripting equivalents to the `pl_close()` *and* `pl_attach()` *API functions, since the scripting tools automatically incorporate those features as necessary.*

These scripting tools can be used by scripts or batch files. The data exported by these batch files via counters can come from various sources. For example, log files of running or terminated applications can be culled by helper utilities and the data can be exported in PL format. Databases maintained by the application or the managing middleware are also good candidates. Even existing counters such as the native Windows counters or the Linux `/proc` pseudo-file system can be used. In these cases, scripts can read the original counter and expose them using the scripting tools, as shown in Figure 15.

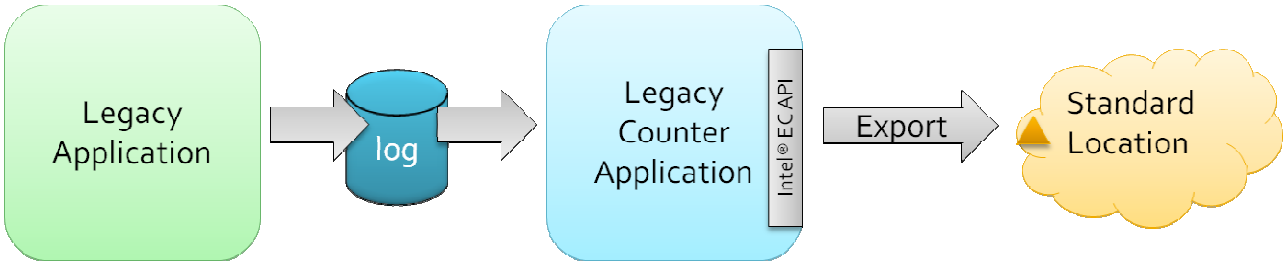

Figure 15: Legacy counter application

NOTE

In addition to managing PLs and counters from a batch program, the scripting tools allow administrators to query any existing PL, especially with the pl_desc tool. This can be very useful in debugging applications.

6.1 pl_open

pl_open creates a Productivity Link as described in Section 1. The pl_open scripting tool is a wrapper to the `pl_open()` API function, so it requires essentially the same information:

- Name of the application.
- Number of counters.
- Name of each counter.

Contrary to a program, pl_open doesn't return a PL descriptor, but instead, prints out the guid created for the PL. The calling application or script file should keep this guid for further interaction with the opened PL.

The listing below provides the help message (from `pl_open --help`).

```
Creates a Productivity Link.

Usage: pl_open <application name> <max_counters> <<counter 1 name>
               <counter 2 name> ... <counter max_counters name>>

   Use quotes (") around names that incorporate spaces.
   To specify an anonymous application name, use NULL.
   To specify an anonymous counter name, use NULL.
   pl_open writes the guid to stdout.
      The guid is a globally unique identifier for the PL.
   Max_counters is a decimal (base 10) number.
   Errors are returned as non-zero exit codes.
   Use [--version] to display version information.
   Use [-h|--help] to display this help message.

```

6.2 pl_read

pl_read reads a value from a Productivity Link counter as described in Section 2.2. The pl_read scripting tool is a wrapper to the `pl_read()` API function, so it requires essentially the same information:

- Name of the application.
- GUID.
- Counter index to read.

The index is zero-based, just like the API.

The listing below provides the help message (from `pl_read --help`).

```
Read a value from a Productivity Link counter.

Usage: pl_read <application name> <guid> <index>

   Use quotes (") around names that incorporate spaces.
   To specify the anonymous application name, use NULL.
   Index is a decimal (base 10) number.
     Indexing is zero based.
   The value of the counter is printed to stdout.
   Errors are returned as non-zero exit codes.
   Use [--version] to display version information.
   Use [-h|--help] to display this help message.

```

6.3 pl_write

pl_write writes a value into a Productivity Link counter as described in Section 1. The pl_write scripting tool is a wrapper to the `pl_write()` API function, so it requires similar information:

- Name of the application.
- GUID returned by the pl_open tool.
- Counter index to read.

The index is zero-based, just like the API.

The listing below provides the help message (from `pl_write --help`).

```
Write a value into a Productivity Link counter.

Usage: pl_write <application name> <guid> <index> <value>

   Use quotes (") around names that incorporate spaces.
   To specify the anonymous application name, use NULL.
   Index and value are decimal (base 10) numbers.
     Indexing is zero based.
   Errors are returned as non-zero exit codes.
   Use [--version] to display version information.
   Use [-h|--help] to display this help message.
```

NOTE

Because pl_write can modify counters owned by another application, it is recommended to restrict access to the PL folder directory (see Table 7) to a subset of users and/or processes.

6.4 pl_desc

pl_desc displays Productivity Link and/or counters descriptions. pl_desc has no direct equivalent in the PL API, but it provides several services, including:

- The ability to list all the PLs visible from the system
- The ability to list all the PLs visible from the system associated with a particular application
- The ability to list the descriptions for all the counters in a given PL

The output of this tool varies depending on the user-supplied input. The index is zero-based as in the API.

The listing below provides the help message (from `pl_desc --help`).

```
Display Productivity Link and / or counter(s) description(s).

Usage: pl_desc <application name> <guid> <index>

   Use quotes (") around names that incorporate spaces.
   To specify an anonymous application name, use NULL.
   If the guid is NULL, then these index values have special meaning:
       -1: List the # of PLs associated to an application name.
       -2: List all the PLs associated to an application name.
       -3: List all the PLs.
   If the guid is not NULL, these index values have special meaning:
       -1: List the # of counter(s) of the specified PL.
       -2: List all the counter(s) of the specified PL.
       Otherwise, list the counter specified in the specified PL.
   Index and value are decimal (base 10) numbers.
       Indexing is zero based.
   The information listed is printed to stdout.
   Errors are returned as non-zero exit codes.
   Use [--version] to display version information.
   Use [-h|--help] to display this help message.

```

6.5 Scripting Sample

The sample shell script code below illustrates how the scripting tools can be used from within a Linux shell script to retrieve PL counters exported by another application. This example uses one of the Linux utilities described in Section 0. This example retrieves CPU utilization information and counts the number of readings in 4 bins: 0-25%, 26-50%, 51-75%, and >75%.

```
#!/bin/bash

#-------------------------------------------------------------------------
# <CTRL>+<C> Processing:
# clean up temp files and exit
#-------------------------------------------------------------------------
trap "rm -f \
   ./data.source \
   ./cpu_pl.out \
   ./cpu_pl_raw.out \
   ./decimals \
   ./gawk.script \
   ./get_cpu_load \
   ./get_decimals \
   ./tmp ; clear; exit" SIGHUP SIGINT SIGTERM

clear

#-------------------------------------------------------------------------
# Setup variables to count current CPU usage and bins
# We define four bins:
# bin0: 0-25, bin1: 26-50, bin2: 51-75 and bin3: 76-100
# NOTE:
# Two first counters of linux_cpu_and_loadavg_info are (see pl_config.ini):
# /CPU usage (percentage)
# /CPU usage (percentage).decimals
#-------------------------------------------------------------------------
declare -i cpu_load=0
declare -i cpu_load_decimals=0
declare -i scaling=0
declare -i cpu_load_bin0=0
declare -i cpu_load_bin1=0
declare -i cpu_load_bin2=0
declare -i cpu_load_bin3=0
declare -i length=0

#-------------------------------------------------------------------------
# Create the gawk script to collect the GUID for our PL instance to read
# gawk script file is saved in gawk.script file
#-------------------------------------------------------------------------
```

```
41  echo "{ if(\$1==\"Using\") { print(substr(\$3, 2, length(\$3) - 2)); } }" >
42  ./gawk.script
43  echo "./linux_cpu_and_loadavg_info | tee ./cpu_pl_raw.out 1>/dev/null" > ./data.source
44  chmod +x ./data.source
45
46  #-------------------------------------------------------------------------
47  # Start the linux_cpu_and_loadavg_info SDK sample code
48  #-------------------------------------------------------------------------
49  /bin/bash -c "./data.source" &
50  sleep 1
51
52  #-------------------------------------------------------------------------
53  # Collect the GUID for our PL instance created by this instance of
54  # linux_cpu_and_loadavg_info
55  #-------------------------------------------------------------------------
56  gawk -f ./gawk.script ./cpu_pl_raw.out > ./cpu_pl.out
57
58  #-------------------------------------------------------------------------
59  # Export several variables required to read data from the PL maintained by
60  # linux_cpu_and_loadavg_info
61  #-------------------------------------------------------------------------
62  MyPL=`cat ./cpu_pl.out`
63  MyApplication="Linux CPU and Load Info"
64  cmd_decimals="./pl_read \"$MyApplication\" $MyPL 1"
65  cmd_cpu_load="./pl_read \"$MyApplication\" $MyPL 0"
66  echo $cmd_decimals > ./get_decimals
67  chmod +x ./get_decimals
68  echo $cmd_cpu_load > ./get_cpu_load
69  chmod +x ./get_cpu_load
70
71  #-------------------------------------------------------------------------
72  # Display information to stdout
73  #-------------------------------------------------------------------------
74  echo "*********************************************************"
75  echo "Monitor CPU load and distribute sample count in four bins."
76  echo "Type <CTRL>+<C> to interrupt monitoring."
77  echo "*********************************************************"
78  echo Application = [$MyApplication]
79  echo PL GUID = [$MyPL]
80  echo Decimals Command = [$cmd_decimals]
81  echo CPU Load Command = [$cmd_cpu_load]
82
83  #-------------------------------------------------------------------------
84  # Read the decimals value for the CPU load
85  #-------------------------------------------------------------------------
86  cpu_load_decimals=`./get_decimals`
87  echo Decimals = [$cpu_load_decimals]
88  scaling=10**$cpu_load_decimals
89  echo Scaling = [$scaling]
```

```
90
91    #-------------------------------------------------------------------------
92    # Loop until interrupted and display cpu use and bin counts
93    #-------------------------------------------------------------------------
94    while [ 1 ]; do
95
96       #-------------------------------------------------------------------------
97       # Read CPU load
98       #-------------------------------------------------------------------------
99       cpu_load=`./get_cpu_load`/$scaling
100
101      #-------------------------------------------------------------------------
102      # Assign the sample to a bin
103      #-------------------------------------------------------------------------
104      if [ $cpu_load -lt 26 ]; then
105         cpu_load_bin0+=1
106      else
107         if [ $cpu_load -lt 52 ]; then
108            cpu_load_bin1+=1
109         else
110            if [ $cpu_load -lt 77 ]; then
111               cpu_load_bin2+=1
112            else
113              cpu_load_bin3+=1
114            fi
115         fi
116      fi
117
118      #-------------------------------------------------------------------------
119      # Build the statistics file
120      #-------------------------------------------------------------------------
121      echo -e "CPU Load = [$cpu_load], #Sample: Bin 0 = [$cpu_load_bin0], Bin 1 =
122   [$cpu_load_bin1], Bin 2 = [$cpu_load_bin2], Bin 3 = [$cpu_load_bin3]\c" > ./tmp
123
124      #-------------------------------------------------------------------------
125      # Display statistics file and erase the line
126      #-------------------------------------------------------------------------
127      cat ./tmp
128      length=`cat ./tmp | wc -m`
129      for (( c=1; c<=$length; c++ ))
130      do
131         echo -e "\b\c"
132      done
133      sleep 1
134   done
```

The sample script above assumes that the PL scripting tools and the Linux system info tools are located in the current working directory. If the PL scripting tools are located in another directory, either fully qualify the filenames or eliminate the leading "./" and ensure the $PATH variable points to the directory where those tools are located (lines 43, 65, and 66 in the code listing above).

The output from the code sample above is shown in Figure 16 below.

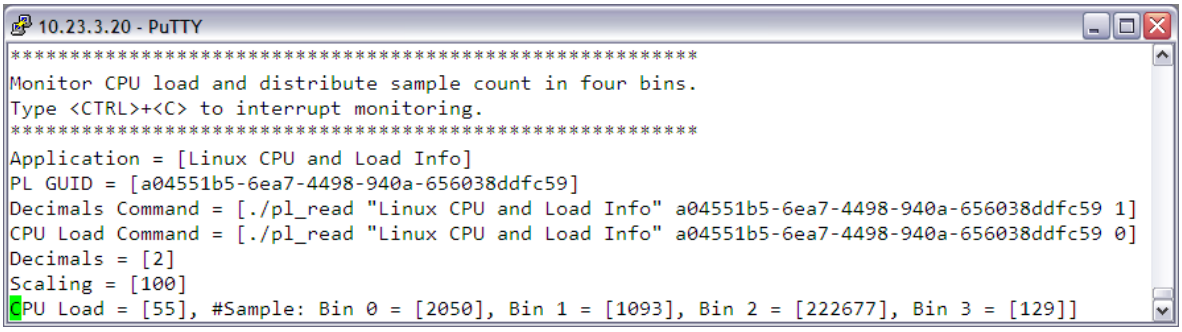

Figure 16: Scripting sample output

This page intentionally blank.

7 Interoperability Tools for Windows*

The Intel Energy Checker SDK's interoperability tools for Windows provide conversion facilities between Windows-native counters and Intel EC counters. These tools enable the user to dynamically convert Windows native counters into Intel EC counters and to convert Intel EC counters into native Windows counters (see Figure 17).

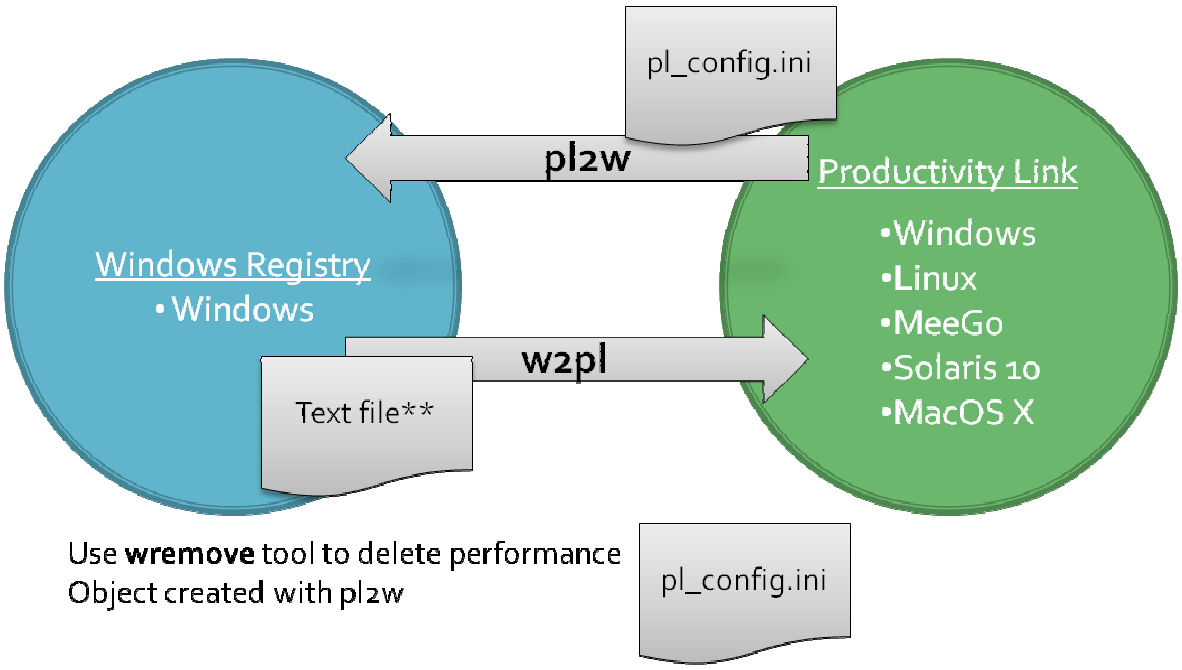

Figure 17: Windows interoperability tools

**w2pl optionally uses a text file containing the application name on its first line and a couplet for each counter to convert (with the Windows counter name on the first line of each couplet and the Intel EC counter name on the second line of each couplet).

The Intel Energy Checker SDK provides four interoperability tools to convert Productivity Link counters to or from Windows:

- pl2w
- w2pl
- wselect
- wremove

For example, there are several figures in this document with POV-Ray* counters monitored from the Windows Perfmon* utility; these were all created with POV-Ray PL counters dynamically converted by pl2w.

7.1 pl2w Tool

pl2w converts all the counters in a Productivity Link into a set of native Windows counters. The conversion frequency defaults to one (1) second between cycles, but this can be modified using the `--pause` switch.

The listing below provides the help message (from `pl2w --help`).

```
Convert a set of Productivity Link counters into native Windows* counters.

Usage: pl2w [--config <file name>] [--pause <t>]

   Use quotes (") around names that incorporate spaces.
   file name is a pl_config.ini file name.
      If omitted, a counter selection GUI is displayed.
      Note: the GUI may require few seconds to display.
   Use [--pause] to specify the time interval between conversions.
      For example, --pause 5 for a five second interval.
      By default, counter conversion takes place at 1 Hz.
   Errors are returned as non-zero exit codes.
   Note: pl2w requires the .NET Framework to execute.
   Note: first time application insertion may require some time to register.
   Use [--version] to display version information.
   Use [-h|--help] to display this help message.
   * Third-party trademarks are the property of their respective owners.

```

NOTE

The first time a set of PL counters is converted to native Windows counters, the registration of the performance object associated with the PL may take some time. Having more counters in the set will also require more time. Subsequent conversions of that same information are faster.

NOTE

Each time pl2w is used with a new PL (not instances of the same PL), it is registered as a performance object associated in the Windows registry. The wremove utility can delete outdated performance objects.

NOTE

pl2w requires the .NET framework to run.

7.2 w2pl Tool

w2pl converts a single or a set of native Windows counter(s) into Productivity Link counter(s). The conversion frequency defaults to one (1) sample per second, though this can be modified be modified using the `--pause` switch.

The listing below provides the help message (from `w2pl --help`).

```
Convert a native Windows* counter into a Productivity Link counter.

Usage: w2pl <Windows counter name> <application name> <PL counter name> [--pause <t>]
Or     w2pl <application name> <PL counter name>
Or     w2pl [--config <configuration file>]

   Use quotes (") around names that incorporate spaces.
   To specify an anonymous application name, use NULL.
   To specify an anonymous counter name, use NULL.
   Windows counter name is a valid native Windows counter name.
      For example, "\\Processor(_Total)\% Processor Time"
      or, "\\SERVER\Processor(_Total)\% C3 Time"
      If omitted, a counter selection GUI is displayed.
      Note: the GUI may require a few seconds to display.
   PL counter name is the name of the converted Windows counter.
      For example, "Total Processor Time (in %)"
   Use [--pause] to specify the time interval between conversions.
      For example, --pause 5 for a five second interval.
      By default, counter conversion takes place at 1 Hz.
   w2pl writes the guid to stdout.
      The guid is a globally unique identifier for the PL.

   The configuration file is the name of a text file containing:
      The application name on the first line.
      One or more 2-line entries for each Windows counter to convert:
         One line containing the Windows counter name
         One line containing the PL counter name
      Note: do not use quotes (") around names that incorporate spaces.
      Note: you may use wselect to select Windows counter names with a GUI
         to ease the Windows counters' name input.  Counters are dumped to
         stdout so you can use this output as the base of the configuration file.
   Errors are returned as non-zero exit codes.
   Counter conversion takes place at 1 Hz.
   Use [--version] to display version information.
   Use [-h|--help] to display this help message.
   * Third-party trademarks are the property of their respective owners.

   Sample configuration file:
   --------------------------
   Laptop
   \\LAPTOP\BatteryStatus(ACPI\PNP0C0A\0_0)\ChargeRate
```

```
42    Charge Rate
43    \\LAPTOP\BatteryStatus(ACPI\PNP0C0A\0_0)\DischargeRate
44    Discharge Rate
45    \\LAPTOP\BatteryStatus(ACPI\PNP0C0A\0_0)\RemainingCapacity
46    Remaining Capacity
47    \\LAPTOP\BatteryStatus(ACPI\PNP0C0A\0_0)\Voltage
48    Voltage
49    \\LAPTOP\ProcessorPerformance(Processor_Number_0)\Power Consumption
50    Power Consumption (Processor 0)
51    \\LAPTOP\ProcessorPerformance(Processor_Number_1)\Power Consumption
52    Power Consumption (Processor 1)
53    \\LAPTOP\ProcessorPerformance(Processor_Number_0)\Processor Frequency
54    Processor Frequency (Processor 0)
55    \\LAPTOP\ProcessorPerformance(Processor_Number_1)\Processor Frequency
56    Processor Frequency (Processor 1)
57    \\LAPTOP\ProcessorPerformance(Processor_Number_0)\% of Maximum Frequency
58    % Maximum Frequency (Processor 0)
59    \\LAPTOP\ProcessorPerformance(Processor_Number_1)\% of Maximum Frequency
60    % Maximum Frequency (Processor 1)
61
```

The sample configuration file shown in lines 42-62 above is part of the help text for w2pl.

7.3 wselect Tool

Native Windows counters have a complex syntax that is error prone if manually entered. **wselect** helps craft the configuration file to be used with w2pl, by listing information from the counters in the proper syntax. This information can then be used to build your configuration file..

To execute `wselect`, enter wselect in a command line. A sample screenshot of the wselect utility is shown in Figure 18 below.

The listing below shows the wselect help message (from `wselect --help`).

```
Prints to stdout a set of GUI selected native Windows* counters.

Usage: wselect

   Registry entries selected in the GUI are printed to stdout.
   Use [--version] to display version information.
   Use [-h|--help] to display this help message.
   Errors are returned as non-zero exit codes.
   * Third-party trademarks are the property of their respective owners.

```

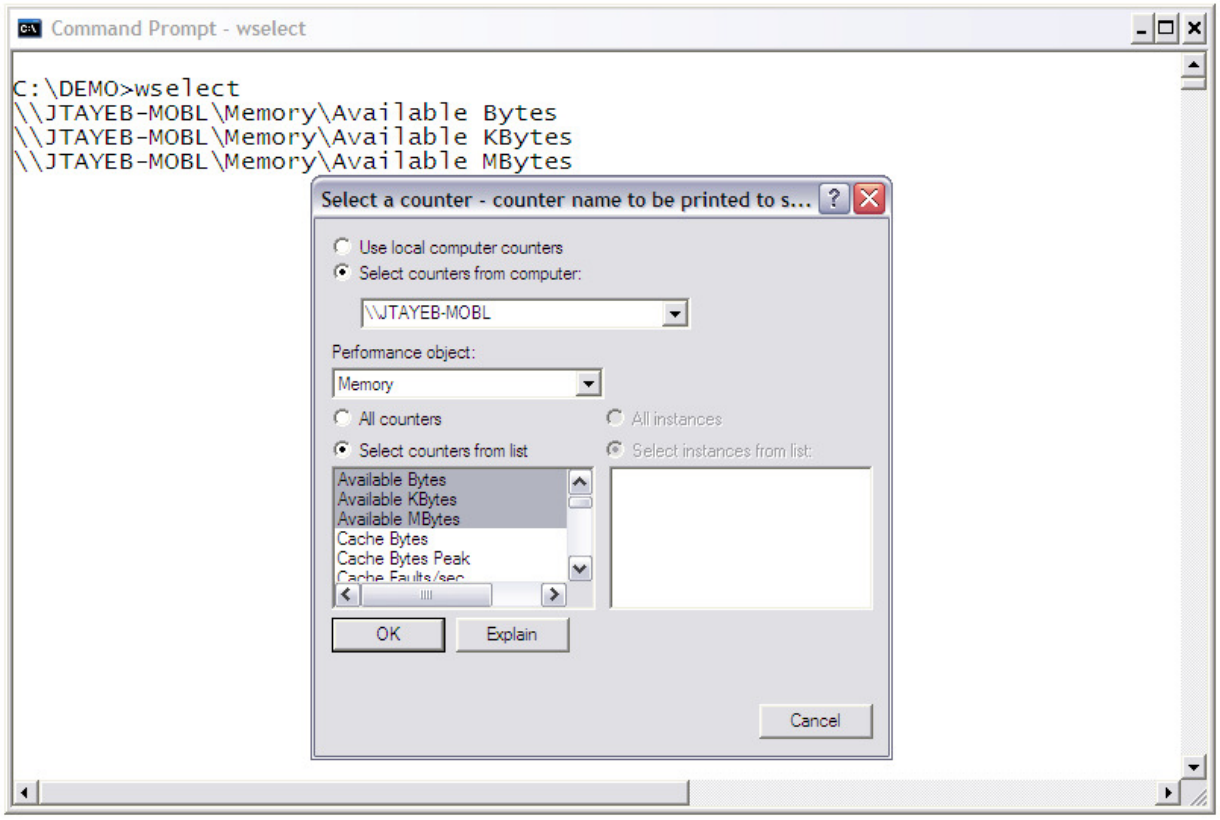

Figure 18: wselect counter selector

wselect prints to stdout the desired data from each counter that is selected in the GUI. Clicking Cancel terminates the utility. The output of wselect can be used to create a configuration file.

The listing below shows the output of wselect, demonstrating how to capture the amount of physical memory available in a PL with three counters.

```
1  \\JTAYEB-MOBL\Memory\Available Bytes
2  \\JTAYEB-MOBL\Memory\Available KBytes
3  \\JTAYEB-MOBL\Memory\Available MBytes
```

In your configuration file, use an editor to combine the wselect output and the data required by w2pl for the conversion. In the example below, the manually added data is shown in bold blue.

```
1  Available Memory
2  \\JTAYEB-MOBL\Memory\Available Bytes
3  Available Memory (in Bytes)
4  \\JTAYEB-MOBL\Memory\Available KBytes
5  Available Memory (in KBytes)
6  \\JTAYEB-MOBL\Memory\Available MBytes
7  Available Memory (in MBytes)
```

Figure 19 and Figure 20 show the combined use of wselect and w2pl.

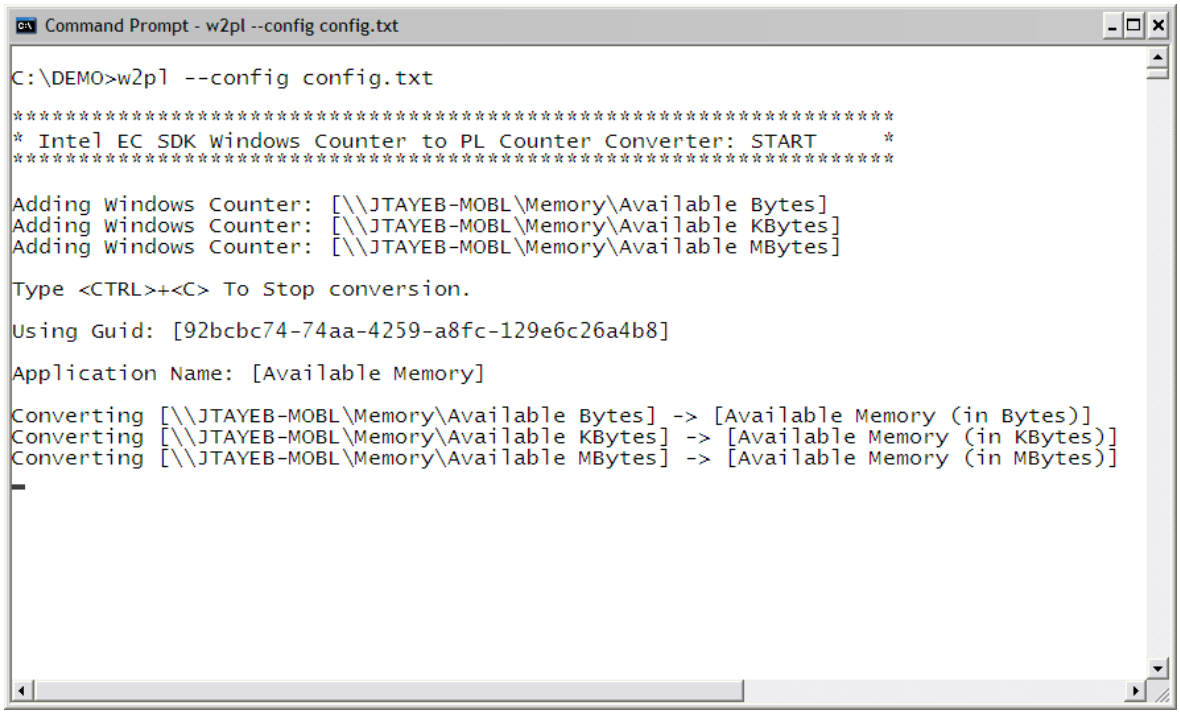

Figure 19: w2pl converting counters from Windows

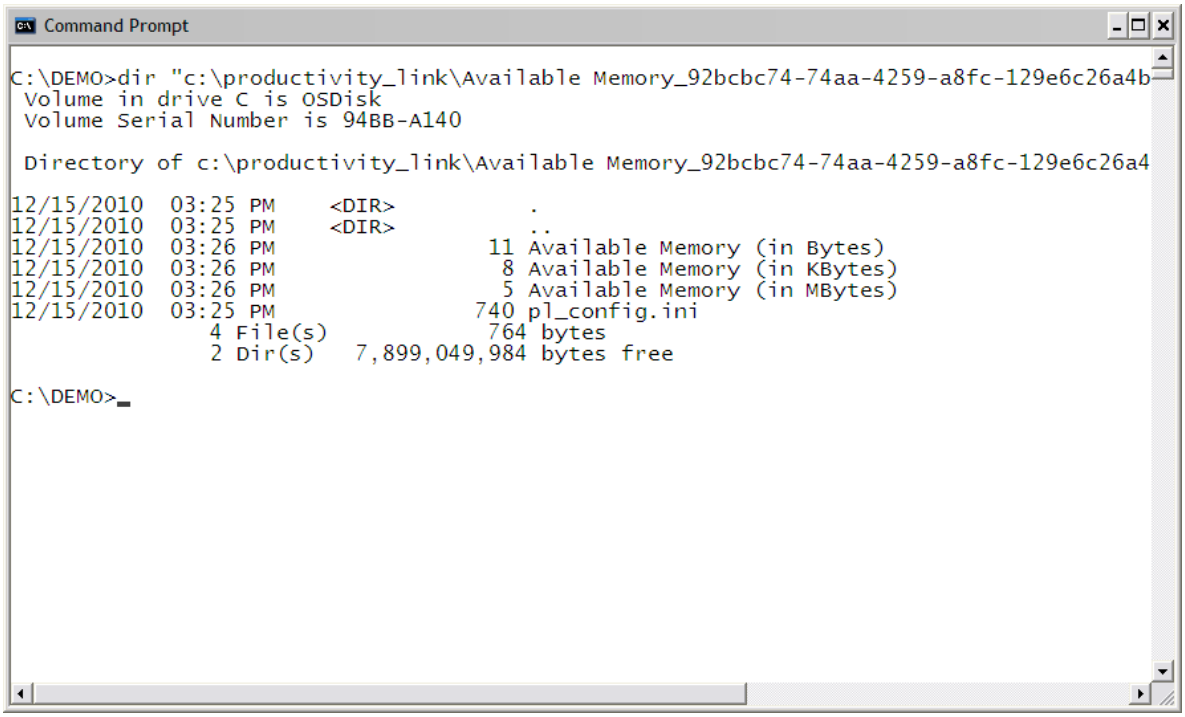

Figure 20: Converted counters

7.4 wremove Tool

wremove helps keep the Windows-based system's registry clean. When using pl2w with a new PL, a new performance object is added into the Windows registry. This performance object will stay there until it is removed. To do so, use wremove and a `pl_config.ini` file. wremove uses the content of the `pl_config.ini` file to remove the performance object from the system's registry.

The listing below provides the wremove help message (from `wremove --help`).

```
Remove a set of Productivity Link counters from Windows* registry.

Usage: wremove [--config <file name>]

   Use quotes (") around names that incorporate spaces.
   file name is a pl_config.ini file name.
      If omitted, a file selection GUI is displayed.
   Errors are returned as non-zero exit codes.
   Note: wremove requires the .NET Framework to execute.
   Use [--version] to display version information.
   Use [-h|--help] to display this help message.
   * Third-party trademarks are the property of their respective owners.

```

NOTE

wremove requires the .NET framework to run.

NOTE

It is necessary to have access to the `pl_config.ini` file used during the registration with wremove. If the original `pl_config.ini` file cannot be found, the w2pl tool can be used to re-create the `pl_config.ini` file.

8 Interoperability Tool for Ganglia*

The Intel Energy Checker SDK's interoperability tool for Ganglia* allows the conversion of native PL counters into Ganglia counter metrics. Using the pl2ganglia interoperability tool of the SDK allows Ganglia users to collect and aggregate PL data in their existing monitoring infrastructure.

8.1 Ganglia Background

The Ganglia monitoring system is *a scalable distributed monitoring system for high-performance computing systems such as clusters and grids*. Refer to the Ganglia web page for further details (http://ganglia.sourceforge.net/).

Ganglia uses a minimum of two levels of data aggregation.

- The first aggregation level is performed at the node level, using an instance of the Ganglia Monitoring daemon, gmond.
- The second aggregation level, at the cluster or grid level, uses instances of the Ganglia Metadata daemon, gmetad.

A working Ganglia system should have the following elements:

- Data collectors (one per node)
  - The gmond daemon
  - A configuration file /etc/gmond.conf
- Data consolidators (one per cluster)
  - The gmetad daemon
  - A configuration file /etc/gmetad.conf
- Database
  - Ganglia uses rrdtool*
- Web GUI tools
  - A web server with PHP support (like Apache*)

Figure 21 shows a typical Ganglia output. The view shows details of the node running ESRV and the graphics representing the counters.

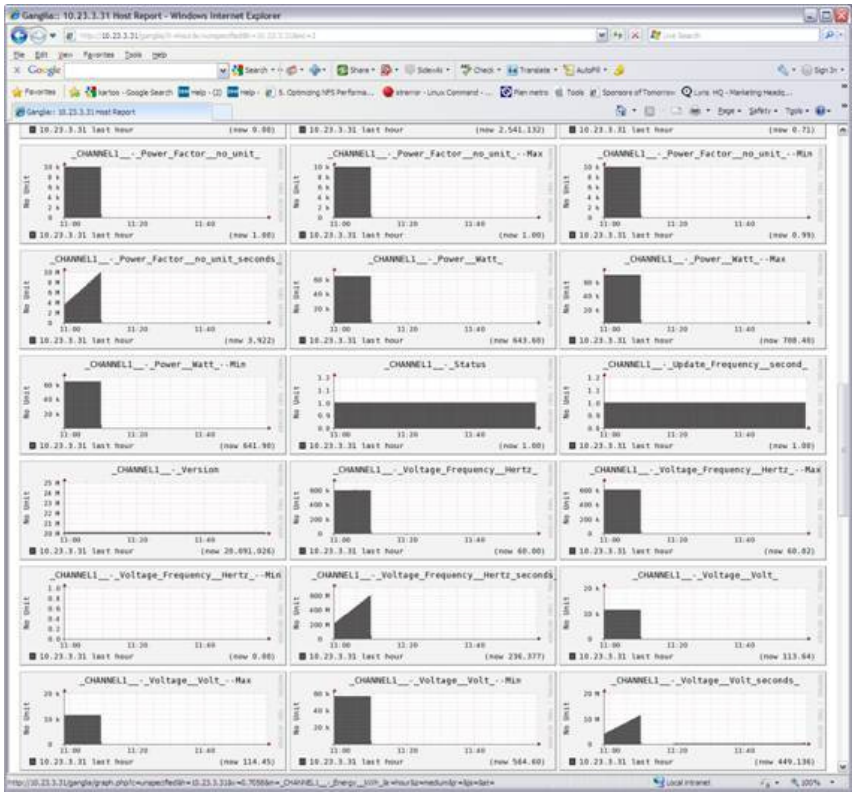

Figure 21: ESRV PL data reported in the Ganglia Web Frontend

8.2 Pl2ganglia Tool

pl2ganglia is shipped with the SDK as a source code, so it is also a sample. This sample only communicates with gmond to pass PL counter data to the aggregation mechanism of Ganglia.

As a sample, the source code can be studied to learn additional techniques to collect and process the PL data and how to interface with an existing monitoring infrastructure.

In this section, the tool will be described, not the sample. See Section 10 to read about pl2ganglia as a sample code and how to build the binary from the source code.

The listing below shows the pl2ganglia help message (from `pl2ganglia --help`).

```
Start pl2ganglia and send PL counter data to local gmond instance.

Usage:  pl2ganglia --application <name> --guid <GUID> ...
        [--pause <n>] [--process] [--decorate] ...
        [--filter] [--force_use_gmetric] [--force_to_double] ...
        [--list_counters | --counters <string>] ...
        [--gmond_config <gmond.conf>]
Or      pl2ganglia [--version]
```

```
Or      pl2ganglia [--help]

Note: ellipsis (...) marks a line continuation.

   pl2ganglia converts the counters of a given PL into Ganglia*
metrics.  These metrics are sent to gmond at a regular time interval
until it is interrupted by the user.
   The metric name is the name of the PL counter.  If name decoration
is requested (--decorate), then the PL application's GUID is appended
to the metric name separated by an underscore (_) character.

   --application with <name> sets the name of the application which
     generates(ed) the PL to convert.
   --guid with <GUID> sets the GUID to specify the application's
     PL instance.
     Note:
       For details on PL, GUID, instances, counters, etc., read
       the Intel(r) Energy Checker SDK User Guide.
   --pause with <n> sets the pause time between each metric update.
     n is expressed in seconds.  n is equal to 60 by default (1 minute).
   --process forces additional analysis on counter values with prescribed
     suffixes.  Use pl2ganglia --process --help for more information on
     the suffixes.
   --decorate appends the PL GUID to each counter name for the Ganglia
     metric names.
   --filter performs the following actions on the Ganglia metric names:
     1 Clips the name to PL2GANGLIA_COUNTER_MAX_LENGTH characters.
     2 Replace space ( ) character with underscode (_) character.
   --force_use_gmetric forces the use of gmetric to send metric update.
     to gmond.  By default, pl2ganglia communicates directly with gmond.
     Note that on Windows*, this option is enforced and the gmetric
     command is displayed for informational purpose only.
   --force_to_double forces native, non-processes PL counters into.
     double metric.  This option allows pl2ganglia to work with versions
     of Ganglia not supporting ull data types.  Caution, data will be
     converted into double and will be lost when a PL counter's value
     becomes too big to be represented by a double on your system.
   --list_counters lists the specified PL's counters after applying
     counter processing if requested (--process).  Each counter is
     numbered from one.  Use this information with --counter option.
   --counters with <string> selects which counters in the PL will be
     converted.  By default all counters in the PL are converted.
     The counter string represents counter numbers and/or counter ranges
     separated by space(s).  A counter range is specified by the first and
     the last (inclusive) counter numbers separated by a dash (-) character.
     For example "1 2 5-10".  Use the --list_counters to identify the
     counter numbers to use.
   --gmond_config with <gmond.conf> sets the configuration file name to
     use.  By default it is set to /etc/ganglia/gmond.conf.
```

```
   --version prints version information.
   --help prints this help message.

Example:
   ./pl2ganglia --application test --guid 8c04f223-425b-42e8-9605-83d...
   05e15d4f9 --boost --decorate --filter --gmond_conf...
   /usr/etc/gmond.conf --process

pl2ganglia continues to send data once per minute to gmond (or
whatever time interval is selected with the --pause option) until it.
is interrupted by the user (<CTRL>+<C>).

* Third-party trademarks are the property of their respective owners.

```

8.3 Reporting Counters

You can use pl2ganglia to report the counters of a PL without running the tool, whether or not the counters are processed. The information provided can be used to select, out of all the available counters in the PL, which counter(s) (again, processed or not) will be sent to gmond.

The listing below shows the output from the following command, which reports the processed counters from an esrv running on the indicated guid.

```
./pl2ganglia --list_counters --application esrv --guid 0a85b319-7fe3-4a01-beb5-
59bd3de7256a --process
```

If the `--process` option is omitted, all of the PL counters will be listed as-is.

```
--------------------------------------------------------------------
 @@@@@  @         @@    @@@@    @@   @    @  @@@@  @      @@@@@    @@
 @    @ @        @  @   @    @  @  @  @@   @ @    @ @        @     @  @
 @    @ @           @   @       @    @ @ @  @ @       @        @    @    @
 @@@@@  @          @    @       @    @ @ @  @ @       @        @    @    @
 @      @         @     @   @@@ @@@@@@ @  @ @ @  @@@ @        @    @@@@@@
 @      @        @      @    @ @    @ @   @@ @    @ @        @    @    @
 @      @@@@@@  @@@@   @@@@  @    @ @    @  @@@@  @@@@@@ @@@@@  @    @
--------------------------------------------------------------------
pl2ganglia: version 2010.04.30
using PL helper version 2009.05.18
using PL version 2010.12.15(L)

[Tue Nov  9 07:15:09 2010]-Using Ganglia* - and gmetric if requested by user - version
3.1.7
* Third-party trademarks are the property of their respective owners.
[Tue Nov  9 07:15:09 2010]----------------------------------------------------------------
-----------
[Tue Nov  9 07:15:09 2010]-...Application Name: [esrv]
[Tue Nov  9 07:15:09 2010]-...Application Instance UUID: [0a85b319-7fe3-4a01-beb5-
59bd3de7256a]
[Tue Nov  9 07:15:09 2010]-...Unprocessed Counters Count: [63]
[Tue Nov  9 07:15:09 2010]----------------------------------------------------------------
-----------
[Tue Nov  9 07:15:09 2010]-...Counter [1]: [[CHANNEL1] - Energy (Joule)]
[Tue Nov  9 07:15:09 2010]-...Counter [2]: [[CHANNEL1] - Energy (kWh)]
[Tue Nov  9 07:15:09 2010]-...Counter [3]: [[CHANNEL1] - Energy Overflows (no unit)]
[Tue Nov  9 07:15:09 2010]-...Counter [4]: [[CHANNEL1] - Update Frequency (second)]
[Tue Nov  9 07:15:09 2010]-...Counter [5]: [[CHANNEL1] - Power (Watt)]
[Tue Nov  9 07:15:09 2010]-...Counter [6]: [[CHANNEL1] - Power (Watt)--Max]
[Tue Nov  9 07:15:09 2010]-...Counter [7]: [[CHANNEL1] - Power (Watt)--Min]
[Tue Nov  9 07:15:09 2010]-...Counter [8]: [[CHANNEL1] - Current (Ampere)]
[Tue Nov  9 07:15:09 2010]-...Counter [9]: [[CHANNEL1] - Current (Ampere)--Max]
[Tue Nov  9 07:15:09 2010]-...Counter [10]: [[CHANNEL1] - Current (Ampere)--Min]
[Tue Nov  9 07:15:09 2010]-...Counter [11]: [[CHANNEL1] - Current (Ampere seconds)]
[Tue Nov  9 07:15:09 2010]-...Counter [12]: [[CHANNEL1] - Voltage (Volt)]
[Tue Nov  9 07:15:09 2010]-...Counter [13]: [[CHANNEL1] - Voltage (Volt)--Max]
[Tue Nov  9 07:15:09 2010]-...Counter [14]: [[CHANNEL1] - Voltage (Volt)--Min]
[Tue Nov  9 07:15:09 2010]-...Counter [15]: [[CHANNEL1] - Voltage (Volt seconds)]
[Tue Nov  9 07:15:09 2010]-...Counter [16]: [[CHANNEL1] - Power Factor (no unit)]
[Tue Nov  9 07:15:09 2010]-...Counter [17]: [[CHANNEL1] - Power Factor (no unit)--Max]
[Tue Nov  9 07:15:09 2010]-...Counter [18]: [[CHANNEL1] - Power Factor (no unit)--Min]
[Tue Nov  9 07:15:09 2010]-...Counter [19]: [[CHANNEL1] - Power Factor (no unit
seconds)]
[Tue Nov  9 07:15:09 2010]-...Counter [20]: [[CHANNEL1] - Current Frequency (Hertz)]
[Tue Nov  9 07:15:09 2010]-...Counter [21]: [[CHANNEL1] - Current Frequency (Hertz)--
Max]
```

```
[Tue Nov  9 07:15:09 2010]-...Counter [22]: [[CHANNEL1] - Current Frequency (Hertz)--
Min]
[Tue Nov  9 07:15:09 2010]-...Counter [23]: [[CHANNEL1] - Current Frequency (Hertz
seconds)]
[Tue Nov  9 07:15:09 2010]-...Counter [24]: [[CHANNEL1] - Voltage Frequency (Hertz)]
[Tue Nov  9 07:15:09 2010]-...Counter [25]: [[CHANNEL1] - Voltage Frequency (Hertz)--
Max]
[Tue Nov  9 07:15:09 2010]-...Counter [26]: [[CHANNEL1] - Voltage Frequency (Hertz)--
Min]
[Tue Nov  9 07:15:09 2010]-...Counter [27]: [[CHANNEL1] - Voltage Frequency (Hertz
seconds)]
[Tue Nov  9 07:15:09 2010]-...Counter [28]: [[CHANNEL1] - Channel(s)]
[Tue Nov  9 07:15:09 2010]-...Counter [29]: [[CHANNEL1] - Status]
[Tue Nov  9 07:15:09 2010]-...Counter [30]: [[CHANNEL1] - Version]
[Tue Nov  9 07:15:09 2010]------------------------------------------------------------
-----------
[Tue Nov  9 07:15:09 2010]-Done.
```

Once you have the list (and number) of the counters, you can specify a sub-set of counters of interest.

For example, out of the 30 processed counters in the previous example, if only the following five counters are of interest

- [CHANNEL1] - Energy (kWh)
- [CHANNEL1] - Power (Watt)
- [CHANNEL1] - Current (Ampere)
- [CHANNEL1] - Current (Ampere)-Max
- [CHANNEL1] - Status)

the following command may be used:

```
./pl2ganglia --application esrv --guid 0a85b319-7fe3-4a01-beb5-59bd3de7256a --process
--counters "2 5 8-9 29" --force_use_gmetric
```

This will deliver an output similar to the following.

```
1  ------------------------------------------------------------------------
2   @@@@@  @         @@    @@@@    @@   @    @  @@@@  @      @@@@@    @@
3   @    @ @        @  @  @    @  @  @  @@   @ @    @ @        @     @  @
4   @    @ @           @  @       @    @ @ @  @ @       @        @     @    @
5   @@@@@  @          @   @       @    @ @ @  @ @       @        @     @    @
6   @      @         @    @  @@@ @@@@@@ @  @ @ @  @@@ @        @     @@@@@@
7   @      @        @     @    @ @    @ @   @@ @    @ @        @     @    @
8   @      @@@@@@  @@@@   @@@@  @    @ @    @  @@@@  @@@@@@ @@@@@  @    @
9   ------------------------------------------------------------------------
10  pl2ganglia: version 2010.04.30
11  using PL helper version 2009.05.18
12  using PL version 2010.12.15(L)
13
14  [Tue Nov  9 07:26:42 2010]-Using Ganglia* - and gmetric if requested by user - version
15  3.1.7
16  * Third-party trademarks are the property of their respective owners.
17  [Tue Nov  9 07:26:42 2010]----------------------------------------------------------
18  -----------
19  [Tue Nov  9 07:26:42 2010]-...Application Name: [esrv]
20  [Tue Nov  9 07:26:42 2010]-...Application Instance UUID: [0a85b319-7fe3-4a01-beb5-
21  59bd3de7256a]
22  [Tue Nov  9 07:26:42 2010]-...Unprocessed Counters Count: [63]
23  [Tue Nov  9 07:26:42 2010]----------------------------------------------------------
24  -----------
25  [Tue Nov  9 07:26:42 2010]-...Processed Counters Count: [30]
26  [Tue Nov  9 07:26:42 2010]-...Pause Interval (in ms): [60000]
27  [Tue Nov  9 07:26:42 2010]-...Counter Processing: [YES]
28  [Tue Nov  9 07:26:42 2010]-...Counter Name Decoration: [NO]
29  [Tue Nov  9 07:26:42 2010]-...Counter Name Filtering: [NO]
30  [Tue Nov  9 07:26:42 2010]-...Forced To Double: [NO]
31  [Tue Nov  9 07:26:42 2010]-...Forced To Use gmetric: [YES]
32  [Tue Nov  9 07:26:42 2010]-...Priority Boost: [NO]
33  [Tue Nov  9 07:26:42 2010]-...Partial Counter Convertion: [YES]
34  [Tue Nov  9 07:26:42 2010]-......Selected Counters Count: [5]
35  [Tue Nov  9 07:26:42 2010]-......Selected Counters: ["2 5 8-9 29"]
36  [Tue Nov  9 07:26:42 2010]----------------------------------------------------------
37  -----------
38  [Tue Nov  9 07:26:42 2010]-Installed Interrupt Signal Handler.
39  Samples:        [8]
```

8.4 Effects of Data Type Mismatches

There is a risk of data loss when converting PL counters into Ganglia counter metrics. This is because the current release of Ganglia uses 32-bit data types, while the Intel Energy Checker SDK uses 64-bit data types.

When such data truncation happens, a message is displayed, and the value of `PL2GANGLIA_UINT_OVERFLOW_VALUE` is injected into the Ganglia counter mertric.

The listing below illustrates output when the following command is issued to force the status counter to PL_MAX_VALUE:

```
1  pl_write esrv 7fbf78e4-37b1-4235-bc5a-5d258db8b09f 61 18446744073709551615
```

and then the following command requests data from five counters:

```
1  ./pl2ganglia --application esrv --guid 7fbf78e4-37b1-4235-bc5a-5d258db8b09f --process
2  --counters "2 5 8-9 29" --force_use_gmetric --pause 1
```

If all counters can be represented as a double, then the `--force_to_double` option can be used.

```
-----------------------------------------------------------------------
 @@@@@  @         @@     @@@@     @@    @     @  @@@@  @       @@@@@     @@
 @     @ @        @  @   @     @   @  @  @@    @ @     @ @         @      @  @
 @     @ @           @   @       @     @ @ @   @ @       @         @     @     @
 @@@@@  @          @    @       @     @ @ @   @ @       @         @     @     @
 @       @         @     @   @@@ @@@@@@ @   @ @ @   @@@ @         @     @@@@@@
 @       @        @      @     @ @     @ @    @@ @     @ @         @     @     @
 @       @@@@@@  @@@@    @@@@  @     @ @     @  @@@@  @@@@@@ @@@@@  @     @
-----------------------------------------------------------------------
pl2ganglia: version 2010.04.30
using PL helper version 2009.05.18
using PL version 2010.12.15(L)

[Tue Nov  9 07:54:20 2010]-Using Ganglia* - and gmetric if requested by user - version
3.1.7
* Third-party trademarks are the property of their respective owners.
[Tue Nov  9 07:54:20 2010]----------------------------------------------------------
-----------
[Tue Nov  9 07:54:20 2010]-...Application Name: [esrv]
[Tue Nov  9 07:54:20 2010]-...Application Instance UUID: [7fbf78e4-37b1-4235-bc5a-
5d258db8b09f]
[Tue Nov  9 07:54:20 2010]-...Unprocessed Counters Count: [63]
[Tue Nov  9 07:54:20 2010]----------------------------------------------------------
-----------
[Tue Nov  9 07:54:20 2010]-...Processed Counters Count: [30]
[Tue Nov  9 07:54:20 2010]-...Pause Interval (in ms): [1000]
[Tue Nov  9 07:54:20 2010]-...Counter Processing: [YES]
[Tue Nov  9 07:54:20 2010]-...Counter Name Decoration: [NO]
[Tue Nov  9 07:54:20 2010]-...Counter Name Filtering: [NO]
[Tue Nov  9 07:54:20 2010]-...Forced To Double: [NO]
[Tue Nov  9 07:54:20 2010]-...Forced To Use gmetric: [YES]
[Tue Nov  9 07:54:20 2010]-...Priority Boost: [NO]
[Tue Nov  9 07:54:20 2010]-...Partial Counter Convertion: [YES]
[Tue Nov  9 07:54:20 2010]-......Selected Counters Count: [5]
[Tue Nov  9 07:54:20 2010]-......Selected Counters: ["2 5 8-9 29"]
[Tue Nov  9 07:54:20 2010]----------------------------------------------------------
-----------
[Tue Nov  9 07:54:20 2010]-Installed Interrupt Signal Handler.
[Tue Nov  9 07:55:12 2010]-...WARNING: counter [61] overflow!
[Tue Nov  9 07:55:13 2010]-...WARNING: counter [61] overflow!
[Tue Nov  9 07:55:14 2010]-...WARNING: counter [61] overflow!
[Tue Nov  9 07:55:15 2010]-...WARNING: counter [61] overflow!
[Tue Nov  9 07:55:16 2010]-...WARNING: counter [61] overflow!
```

NOTE

In the example above, counter 61 is reported, but it is actually counter 29. When a counter overflow is signaled, the absolute offset of the counter in the PL is reported. This difference is due to the processing of the counters that consume multiple counters as part of one real value.

NOTE

pl2ganglia is a standalone process. However, Ganglia allows the creation of DSO modules for gmond. It is possible to create such modules using the algorithms used in pl2ganglia.

9 Using ESRV and TSRV Data

The Intel Energy Checker SDK is shipped with two utilities to measure and report power and environmental information of monitored system(s).

- Energy Server (ESRV) monitors and reports power and energy consumption.
- Temperature Server (TSRV) monitors and reports temperature and humidity conditions.

The data from these utilities are exposed using the Intel EC API. Any application using the API can access this data for monitoring, reporting, and control and management usages.

The SDK provides information on how to configure and operate the ESRV and TSRV.

9.1 Creating Energy-aware, Energy-efficient Software

Through the Intel EC API and ESRV, software can become aware of the power and energy consumption of monitored systems. This is the first step toward energy-aware software that can optimize operations for energy efficiency.

This section explains how ESRV counters can be used in an application to compute and report energy efficiency metrics. Independent Software Vendors (ISVs) can use this as a model for their application.

Incorporating ESRV capability into their applications, ISVs can provide energy efficiency metrics in their software by correlating the amount of useful work performed with the amount of energy consumed, such as the energy used by a graphics rendering engine to render a measured number of pixels over a period of time. Such correlations can lead to more efficient software operations, which benefit data centers in their quest to contain and cut power costs while creating more efficient data centers.

Similarly, it's often useful to track performance and power consumption relative to temperature and humidity conditions. Applications can utilize TSRV counters for this capability.

9.2 Getting Started with ESRV

Before presenting how ESRV and TSRV data can be used programatically, this section will list the steps required to run ESRV and to collect a first batch of data manually. The steps will be described for Windows and Linux operating systems and will cover two scenarios. This mini-tutorial applies to a single system setup. The two scenarios are:

- The user does not have a power meter (yet).
- The user has a Yokogawa* WT210 power analyzer (with a serial interface).

The use of other power meters is similar. See the *Intel® Energy Checker SDK Device Driver Kit User Guide* for details on supported power meters and the available options for each of them.

NOTE

Some steps may require adaptation based on the user's system configuration. However, if the prerequisits listed below are met, then all the steps should be reproductible as-is.

NOTE

Besides the Extech 380801 power meter, which ESRV supports natively, all other power meters require a support library. In this case, substitute the `--device` option for `--library` options. The later usage model requires specifying the name of the library after the option.

The following are prerequisites for the setup:

- For all scenarios:
  - The PL_FOLDER is created and the user can read and write the PL_FOLDER. See Section 4.1 for details on the PL_FOLDER.
  - The folders where ESRV, pl_csv_logger, and pl_gui_monitor are copied are in the system path. Refer to the *Intel Energy Checker SDK Companion Applications User Guide* for details on pl_csv_logger and pl_gui_monitor.
- For scenario 1:
  - The CPU-indexed simulated device ESRV support library is located in a folder which is in the system path. The following libraries must be used:
    - cpu_indexed_simulated_device.dll for Windows
    - cpu_indexed_simulated_device.so.1.0 for Linux
      - Users may have to setup symbolic links for the library and to update the system's library paths.
- For scenario 2:
  - The Yokogawa WT210 is connected to serial port 1. The device end of record (EOR) is set to LF. See the *Yokogawa WT210 User Guide* for setup details or the short description of the WT210 setup provided in Section 0. Communications should be as follows:
    - COM 1 for Windows. The port and the device communication settings match and are set to: baud=9600, parity=n, data=8, stop=1, xon=n, dtr=y, and rts=n.
    - /dev/ttyS1 for Linux

9.2.1 Start ESRV

9.2.1.1. Windows, Scenario 1

For this example, the support module simulates data using the CPU utilization percentage as returned by the OS.

Type the following command in a shell:

```
esrv --start --library esrv_cpu_indexed_simulated_device.dll
```

A printout like the one shown below is displayed. The samples count should increase by one every second. Version data may differ and the UUID will always differ. At this stage, ESRV is running. By default, the simulated power readings are based on a fixed power draw of 60.0 watts plus a variable power draw of 40.0 watts, of which 25.0 percent is indexed to the CPU utilization percentage. See the section dedicated to the ESRV CPU indexed simulated device in the *Intel® Energy Checker SDK Device Driver Kit User Guide* for details.

```
*******************************************
* IECSDK Sample Energy Server: START       *
*******************************************

Sample Energy Monitor: esrv version 2009.10.26 - using PL version 2010.12.15(W)

Server Name:    [esrv]
Using Guid:     [2089f820-d787-43c3-a96b-56b56d948706]
Samples:        [5]
```

9.2.1.2. Linux, Scenario 1

For this example, the support module simulates data using the CPU utilization percentage as returned by the OS.

Type the following command in a shell:

```
./esrv --start --library ./esrv_cpu_indexed_simulated_device.so
```

A printout like the one shown below is displayed. The samples count should increase by one every second. Version data may differ and the UUID will always differ. At this stage, ESRV is running. By default, the simulated power readings are based on a fixed power draw of 60.0 watts plus a variable power draw of 40.0 watts, of which 25.0 percent is indexed to the CPU utilization percentage. See the section dedicated to the ESRV CPU indexed simulated device in the *Intel® Energy Checker SDK Device Driver Kit User Guide* for details.

```
*****************************************
* IECSDK Sample Energy Server: START        *
*****************************************

Sample Energy Monitor: esrv version 2009.10.26 - using PL version 2010.12.15(L)

Server Name:    [esrv]
Using Guid:     [38e57a18-77be-4a60-9fc8-920eb98787ef]
Samples:        [4]
```

NOTE

In the command shown above, ESRV and the library are supposed to be in the same location, and no assumptions are made regarding the path and library path environment variables. The command is issued from the folder containing ESRV and the library.

9.2.1.3. Windows, Scenario 2

This example illustrates using an external measurement device (the WT210), which is natively supported in the SDK.

Type the following command in a shell:

```
esrv --start --device y210
```

A printout like the one shown below is displayed. The samples count should increment by one every second. Version data may differ and the UUID will always differ. See the section dedicated to the Yokogawa WT210 power meter in the *Intel® Energy Checker SDK Device Driver Kit User Guide* for details.

```
*****************************************
* IECSDK Sample Energy Server: START        *
*****************************************

Sample Energy Monitor: esrv version 2009.10.26 - using PL version 2010.12.15(W)

Server Name:    [esrv]
Using Guid:     [83b93ec0-5316-4177-a4eb-25992f33687a]
Samples:        [5]

```

9.2.1.4. Linux, Scenario 2

This example illustrates using an external measurement device (the WT210), which is natively supported in the SDK.

Type the following command in a shell:

```
./esrv --start --device y210
```

A printout like the one shown below is displayed. The samples count should increment by one every second. Version data may differ and the UUID will always differ. See the section dedicated to the Yokogawa WT210 power meter in the *Intel® Energy Checker SDK Device Driver Kit User Guide* for details.

```
*****************************************
* IECSDK Sample Energy Server: START        *
*****************************************

Sample Energy Monitor: esrv version 2009.10.26 - using PL version 2010.12.15(L)

Server Name:    [esrv]
Using Guid:     [34c9435e-b7b0-4426-90a1-f06adb645b37]
Samples:        [5]
```

On all supported operating systems, if the `--diagnostic` option is specified in the command line, additional information is displayed by ESRV during startup. This information may be useful to debug situations where ESRV fails to start.

The listing below uses the `--diagnostic` option. It shows the output generated by ESRV under Linux where it cannot communicate with the device (in this case the device was simply not connected to the system). This is just an example, and debug messages may vary based on the issue.

```
1  ******************************************
2  * IECSDK Sample Energy Server: START       *
3  ******************************************
4
5  Sample Energy Monitor: esrv version 2009.10.26 - using PL version 2010.12.15(L)
6
7  [Fri Nov  5 09:53:45 2010]-DIAG: Loading And Configuring Device. [SERVER]
8  [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering init_yokogawa_wt210_extra_data
9  Function. [SERVER]
10 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering delete_yokogawa_wt210_extra_data
11 Function. [SERVER]
12 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering open_yokogawa_wt210 Function. [SERVER]
13 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering close_yokogawa_wt210 Function.
14 [SERVER]
15 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering parse_yokogawa_wt210_option_string
16 Function. [SERVER]
17 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering
18 read_yokogawa_wt210_read_all_measurements Function. [SERVER]
19 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering read_yokogawa_wt210_power Function.
20 [SERVER]
21 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering read_yokogawa_wt210_energy Function.
22 [SERVER]
23 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering read_yokogawa_wt210_current Function.
24 [SERVER]
25 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering read_yokogawa_wt210_voltage Function.
26 [SERVER]
27 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering read_yokogawa_wt210_power_factor
28 Function. [SERVER]
29 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering read_yokogawa_wt210_voltage_frequency
30 Function. [SERVER]
31 [Fri Nov  5 09:53:45 2010]-DIAG: ...Registering read_yokogawa_wt210_current_frequency
32 Function. [SERVER]
33 [Fri Nov  5 09:53:45 2010]-DIAG: Setup Thread Synchronization Data. [SERVER]
34 [Fri Nov  5 09:53:45 2010]-DIAG: Initializing Device Extra Data - First Call. [SERVER]
35 [Fri Nov  5 09:53:45 2010]-DIAG: Setting interface function pointers to Serial.
36 [SERVER]
37 [Fri Nov  5 09:53:45 2010]-DIAG: Pre-Parsing Default Serial Interface Options.
38 [SERVER]
39 [Fri Nov  5 09:53:45 2010]-DIAG: ...Default Options: [com=1 baud=9600 parity=n data=8
40 stop=1 xon=n dtr=y rts=n]. [SERVER]
41 [Fri Nov  5 09:53:45 2010]-DIAG: No Interface Options To Parse. [SERVER]
42 [Fri Nov  5 09:53:45 2010]-DIAG: No Device Options String To Parse. [SERVER]
43 [Fri Nov  5 09:53:45 2010]-DIAG: Initializing Device Extra Data - Second Call.
44 [SERVER]
45 [Fri Nov  5 09:53:45 2010]-DIAG: Opening Interface. [SERVER]
46 [Fri Nov  5 09:53:45 2010]-DIAG: ...Opening Serial Port. [SERVER]
47 [Fri Nov  5 09:53:45 2010]-DIAG: ...Configuring Serial Port. [SERVER]
48 [Fri Nov  5 09:53:45 2010]-DIAG: ...Purging Serial Port. [SERVER]
```

```
[Fri Nov  5 09:53:45 2010]-DIAG: Opening Device. [SERVER]
[Fri Nov  5 09:53:45 2010]-DIAG: ...VA1 Item Was Requested But Will Not Be Served.
[Y210]
[Fri Nov  5 09:53:45 2010]-DIAG: ...VAR1 Item Was Requested But Will Not Be Served.
[Y210]
[Fri Nov  5 09:53:45 2010]-DIAG: ...DEGR1 Item Was Requested But Will Not Be Served.
[Y210]
[Fri Nov  5 09:53:45 2010]-DIAG: ...WHP1 Item Was Requested But Will Not Be Served.
[Y210]
[Fri Nov  5 09:53:45 2010]-DIAG: ...WHM1 Item Was Requested But Will Not Be Served.
[Y210]
[Fri Nov  5 09:53:45 2010]-DIAG: ...AH1 Item Was Requested But Will Not Be Served.
[Y210]
[Fri Nov  5 09:53:45 2010]-DIAG: ...AHP1 Item Was Requested But Will Not Be Served.
[Y210]
[Fri Nov  5 09:53:45 2010]-DIAG: ...AHM1 Item Was Requested But Will Not Be Served.
[Y210]
[Fri Nov  5 09:53:45 2010]-DIAG: ...TIME Item Was Requested But Will Not Be Served.
[Y210]
[Fri Nov  5 09:53:47 2010]-DIAG: ...Retry Read. [SERIAL READ]
[Fri Nov  5 09:53:51 2010]-DIAG: ...Retry Read. [SERIAL READ]
[Fri Nov  5 09:53:55 2010]-DIAG: ...Retry Read. [SERIAL READ]
[Fri Nov  5 09:53:59 2010]-DIAG: ...Retry Read. [SERIAL READ]
[Fri Nov  5 09:53:59 2010]-DIAG: ...Read Fatal Error. [SERIAL READ]
[Fri Nov  5 09:53:59 2010]-DIAG: Closing Interface. [SERVER]
[Fri Nov  5 09:53:59 2010]-DIAG: ...Closing Serial Port. [SERVER]
[Fri Nov  5 09:53:59 2010]-DIAG: Freeing Device Extra Data. [SERVER]
[Fri Nov  5 09:53:59 2010]-DIAG: Delete Thread Synchronization Event. [SERVER]
[Fri Nov  5 09:53:59 2010]-DIAG: Shutting Down Inter-Process Communication. [COMMON]


*****************************************
* IECSDK Sample Energy Server: STOP         *
*****************************************
[Fri Nov  5 09:53:59 2010]-DIAG: Stopping Energy Server. [COMMON].
```

9.2.2 Monitor the ESRV PL Data

One quick and simple way to monitor any PL data is to use the pl_gui_monitor companion application shipped with the SDK. This tool is shipped as a binary. See the *Intel® Energy Checker SDK Companion Applications User Guide* for details on this tool.

NOTE

To stop the monitor at any time, hit <CTRL>+<C> in the shell window, or simply close the monitor's graphic window. Monitoring can be restarted later on, with no effect to the on-going measurements.

9.2.2.1. Windows, All Scenarios

This example illustrates how to start the monitor when a configuration file name is not supplied.

Type the following command in a shell:

```
1  pl_gui_monitor
```

If no fully qualified PL configuration file name is provided on the command line, an Open dialog box is displayed to interactively select the PL(s) to monitor (Figure 22). You can select more than one to monitor multiple PLs.

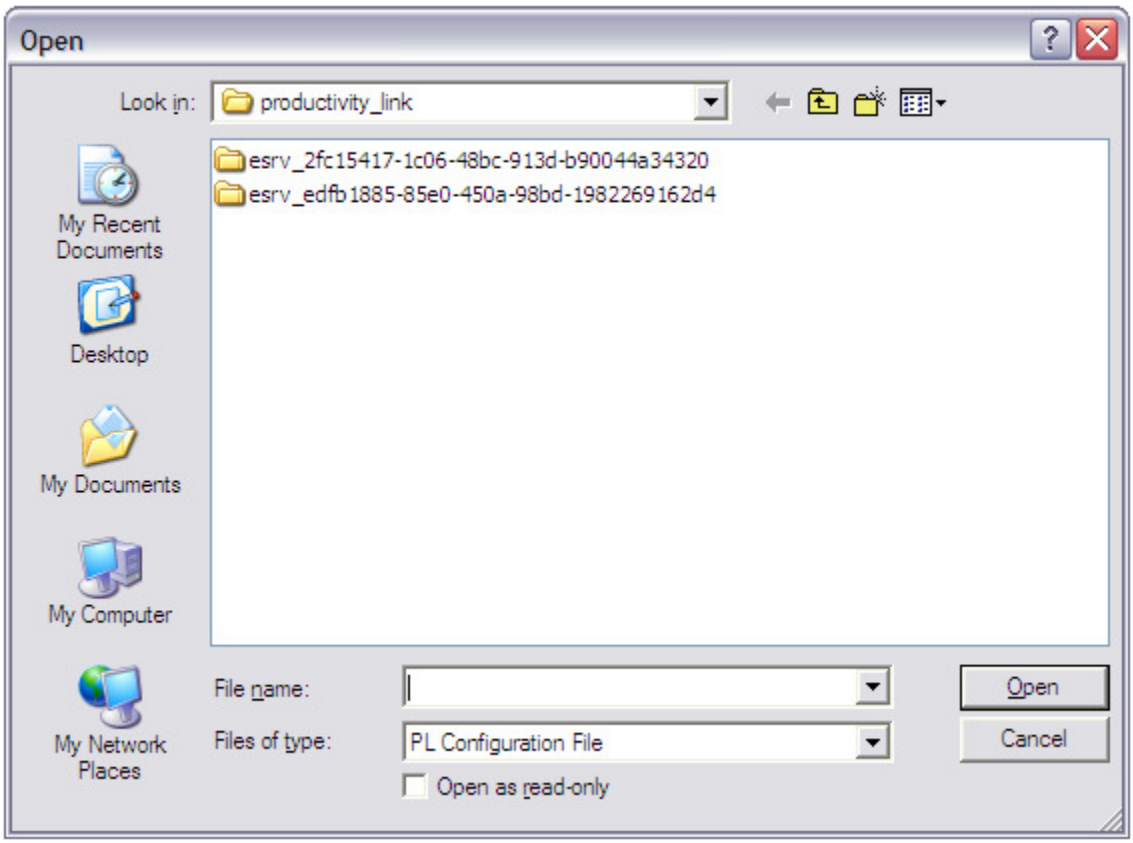

Figure 22: Select the PL to monitor and click on Cancel to proceed.

Double-click on the PL folder you want to monitor and double click on the pl_config.ini file listed in it. The Open dialog box will be repeatedly displayed until the Cancel button is clicked. This allows for the selection of multiple PLs to monitor. However, in this tutorial simply select the pl_config.ini file of the ESRV instance just started and click on the Cancel button to start the monitoring.

Figure 22 depicts a situation where a second ESRV PL exists. It exists because it was actually communicating with an external power analyzer, measuring the power and energy consumed by a remote system. This situation will happen – with even more PLs – as ESRV instances are started and stopped. The PLs are persistent.

Note that if a PL becomes useless, simply delete its folder.

NOTE

It is also possible to specify on the command line the fully qualified PL configuration file name. The syntax is the same as Linux, with the exception of the PL_FOLDER name (see Section 4.1).

Figure 23 shows the display that youn should see.

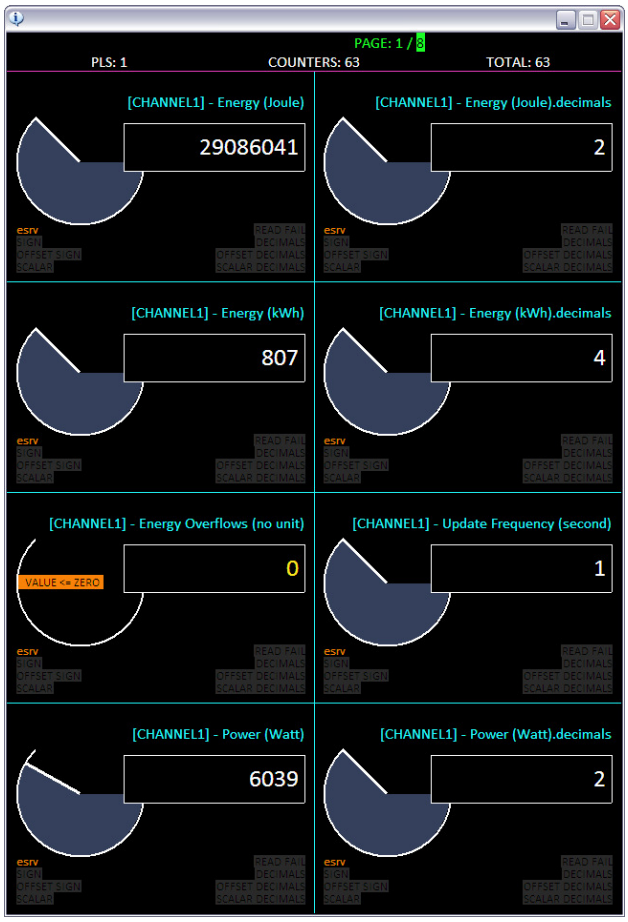

Figure 23: Example of a PL monitored as-is.

Values are displayed in their brut, non-processed, format. The power readings are not directly usable. To get usable readings, use the --process option in the command line to direct the monitor to compute real values using the suffix counters. See Section 0 for details on suffix counters.

Additional options can be used to make the readings more usable. In particular, the following options are useful:

- `--format` to direct the monitor to format counters values so they are easier to read. See the *Intel® Energy Checker SDK Companion Applications User Guide* for details.
- `--gdiplus --transparency 30 --top`. These options use GDI+ for smoother display, to display the monitor graphic window partially transparent so the user can see what is behing the window, and to position the monitor window on top of all other windows.

Figure 24 shows the display with these additional options. The command line is:

```
1   pl_gui_monitor --process --format --gdiplus --transparency 30 --top
```

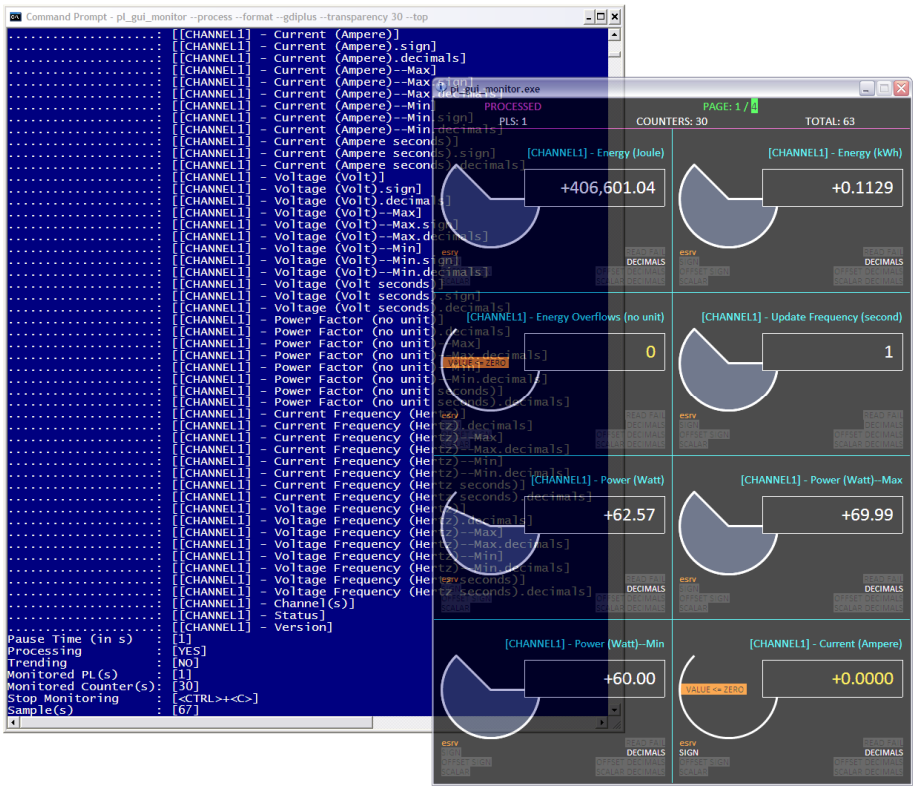

Figure 24: Monitor display with enhanced options.

9.2.2.2. Linux, All Scenarios

Under Linux, you must include a fully qualified PL configuration file name in the command line. The command below starts pl_gui_monitor to monitor the PL counters generated by the ESRV instance started in the previous section.

This example assumes that the UUID of this instance of ESRV is f18ee848-736d-4bc0-8c6f-576f6e096997. The UUID is printed out at ESRV startup. Review the SDK's various samples to see how this UUID can be shared among applications progamatically.

This section only focuses on a manual exercise; therefore it is up to the user to provide this information.

Type the following command in a shell:

```
./pl_gui_monitor /opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/pl_config.ini
```

Additional options can be used to make the readings more usable. In particular, the following options are useful:

- `--process` to direct the monitor to use the suffix counters to compute and display real data values. See Section 0 for details on suffix counters.
- `--format` to direct the monitor to format counters values so they are easier to read. See the *Intel® Energy Checker SDK Companion Applications User Guide* for details.

With these additional options, the command line is:

```
./pl_gui_monitor --process --format /opt/productivity_link/esrv_f18ee848-736d-4bc0-
8c6f-576f6e096997/pl_config.ini
```

NOTE

The GDI/GDI+-based Windows version of the monitor has many more graphical options than the X11 based Linux version. Refer to the Intel® Energy Checker SDK Companion User Guide for details on the available options.

9.2.3 Record the ESRV PL Data

One quick and simple way to record any PL data is to use the pl_csv_logger companion application shipped with the SDK as a source code. See the *Intel® Energy Checker SDK Companion Applications User Guide* for details.

NOTE

To stop the logger at any time, hit <CTRL>+<C> in the shell window. Logging can be restarted later on, with no effect to the on-going measurements.

NOTE

Since the pl_csv_logger is shipped as a set of source code, the user will have to build the logger prior to using it. This will be demonstrated later on.

9.2.3.1. Windows, All Scenarios

Open the `iecsdk\build\windows\iecsdk\iecsdk.sln` Microsoft Visual Studio 2005 Solution. Right-click on the pl_csv_logger project and select build in the drop-down menu. Figure 25 shows the typical output of the build process. Refer to Section 5.1 for details on the build process.

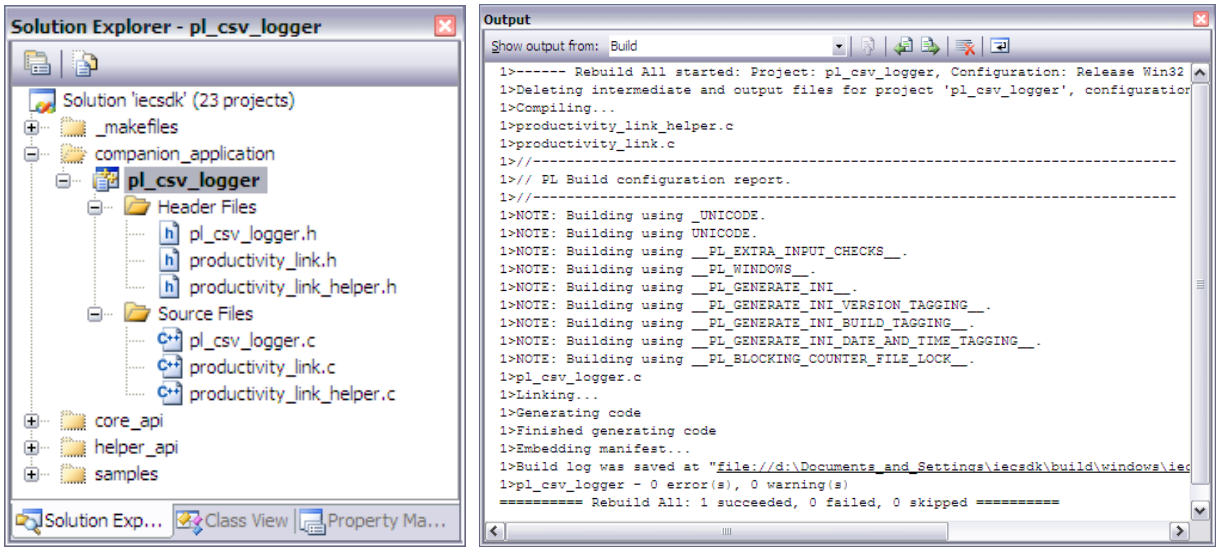

Figure 25: Logger build result in Visual Studio 2005.

Type the following command in a shell:

```
1  pl_csv_logger /opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
2  576f6e096997/pl_config.ini
```

A listing similar to the one partially reproduced in the Linux section below (Section 9.2.3.2) is printed to the screen. The end of this output shows the actual data as it is collected (first three samples shown).

Additional options can be used to make the readings more usable. In particular, the following options are useful:

- `--process` to direct the logger to use the suffix counters to compute real data. See Section 0 for details on suffix counters.
- `--output <f>` to direct the logger to save the data in a comma separated format in a text file (named `f`). See the *Intel® Energy Checker SDK Companion Applications User Guide* for details.

With these additional options, the command line is:

```
pl_csv_logger /opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/pl_config.ini --process --output ./esrv_log.csv
```

Once the logger is interrupted (using <CTRL>+<C> in the shell window), the content of the output file can be reviewed in a spreadsheet application for further analysis. Figure 26 shows a simple plotting of the data in Microsoft Excel of the first 100 samples.

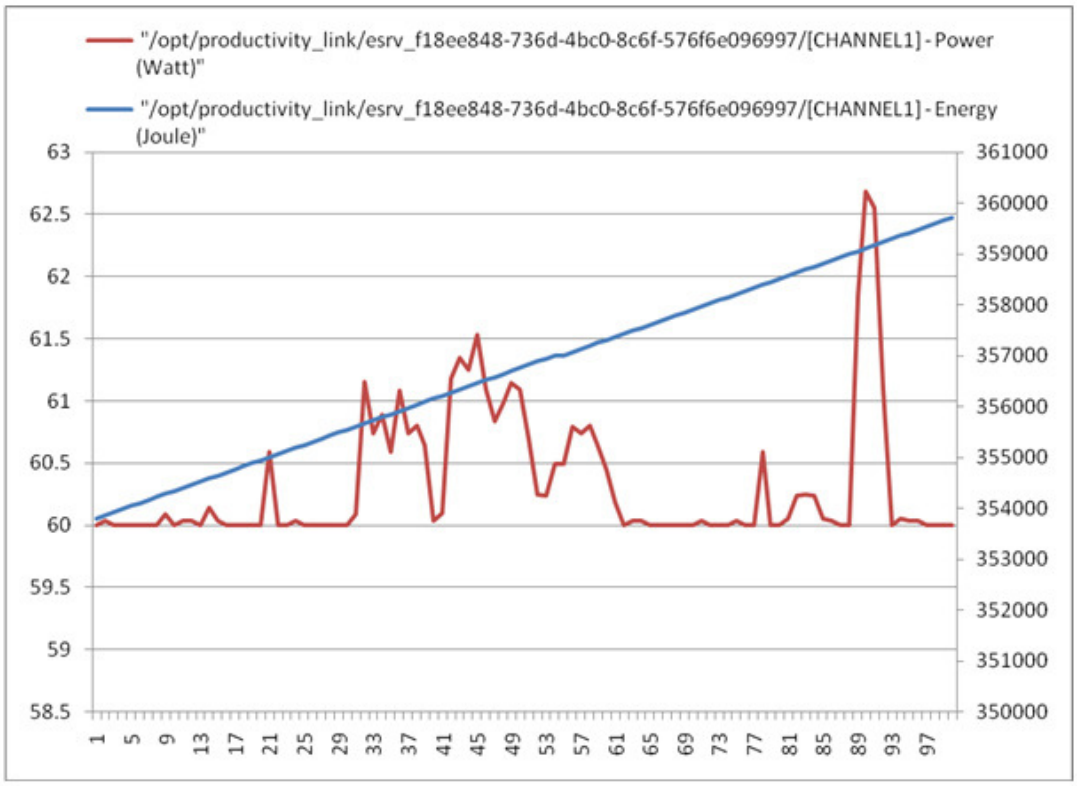

Figure 26: Plot of power and energy data collected using the logger.

9.2.3.2. Linux, All Scenarios

To build ESRV under Linux, change to directory `iecsdk/build/linux`. Type

```
make pl_csv_logger.
```

The following listing shows the typical build result. Refer to Section 5.2 for details on the build process and Makefile configuration.

```
/usr/bin/gcc \
        -D__PL_LINUX__ -D__PL_GENERATE_INI__ -D__PL_GENERATE_INI_VERSION_TAGGING__ -
D__PL_GENERATE_INI_BUILD_TAGGING__ -D__PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ -
D__PL_BLOCKING_COUNTER_FILE_LOCK__ -D__PL_EXTRA_INPUT_CHECKS__ -m64  -O2 -msse -Wall -
fPIC -std=gnu99 -D_SVID_SOURCE -D_REENTRANT -D_LIBC_REENTRANT -pthread \
        -I/orssg-data/Development/iecsdk/src/core_api \
        -I/orssg-data/Development/iecsdk/src/helper_api \
        -I/orssg-data/Development/iecsdk/src/companion_applications/pl_csv_logger \
        /orssg-
data/Development/iecsdk/src/companion_applications/pl_csv_logger/pl_csv_logger.c \
        /orssg-data/Development/iecsdk/src/core_api/productivity_link.c \
        /orssg-data/Development/iecsdk/src/helper_api/productivity_link_helper.c \
        -o pl_csv_logger \
        -lpthread -luuid -ldl \
        -lm
```

Type the following command in a shell:

```
./pl_csv_logger /opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/pl_config.ini
```

A listing similar to the one partially reproduced below is printed to the screen. The end of this output shows the actual data as it is collected (first three samples shown).

```
  @@@@   @@@@  @    @        @       @@@@   @@@@   @@@@  @@@@@@ @@@@@
 @    @ @    @ @    @        @       @    @ @    @ @    @ @      @    @
 @      @      @    @        @       @    @ @      @      @      @    @
 @       @@@@  @    @        @       @    @ @      @      @@@@@@ @@@@@
 @           @  @   @        @       @    @ @  @@@ @  @@@ @      @  @
 @    @      @   @ @         @       @    @ @    @ @    @ @      @   @
  @@@@  @@@@@     @          @@@@@@  @@@@   @@@@   @@@@  @@@@@@ @    @


Using PL.........: [/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/pl_config.ini]
Application......: [esrv]
Counter(s).......: [...]
.................: [/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Energy (Joule)]

        <------------------------ MANY LINES OMMITED ------------------------>

.................: [/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Version]
Pause Time       : [1]
Processing       : [NO]
Logged PL(s)     : [1]
Logged Counter(s): [63]
Saving Log Into  : [stdout]
"Time Stamp", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Energy (Joule)", "/opt/productivity_link/esrv_f18ee848-736d-
4bc0-8c6f-576f6e096997/[CHANNEL1] - Energy (Joule).decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Energy
(kWh)", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] -
Energy (kWh).decimals", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Energy Overflows (no unit)",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Update
Frequency (second)", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Power (Watt)", "/opt/productivity_link/esrv_f18ee848-736d-
4bc0-8c6f-576f6e096997/[CHANNEL1] - Power (Watt).decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Power
(Watt)--Max", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Power (Watt)--Max.decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Power
(Watt)--Min", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Power (Watt)--Min.decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Current
(Ampere)", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Current (Ampere).sign",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Current
(Ampere).decimals", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Current (Ampere)--Max",
```

```
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Current
(Ampere)--Max.sign", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Current (Ampere)--Max.decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Current
(Ampere)--Min", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Current (Ampere)--Min.sign",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Current
(Ampere)--Min.decimals", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Current (Ampere seconds)",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Current
(Ampere seconds).sign", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Current (Ampere seconds).decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Voltage
(Volt)", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1]
- Voltage (Volt).sign", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Voltage (Volt).decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Voltage
(Volt)--Max", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Voltage (Volt)--Max.sign",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Voltage
(Volt)--Max.decimals", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Voltage (Volt)--Min", "/opt/productivity_link/esrv_f18ee848-
736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Voltage (Volt)--Min.sign",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Voltage
(Volt)--Min.decimals", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Voltage (Volt seconds)",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Voltage
(Volt seconds).sign", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Voltage (Volt seconds).decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Power
Factor (no unit)", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Power Factor (no unit).decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Power
Factor (no unit)--Max", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Power Factor (no unit)--Max.decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Power
Factor (no unit)--Min", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Power Factor (no unit)--Min.decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Power
Factor (no unit seconds)", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Power Factor (no unit seconds).decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Current
Frequency (Hertz)", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Current Frequency (Hertz).decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Current
Frequency (Hertz)--Max", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Current Frequency (Hertz)--Max.decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Current
Frequency (Hertz)--Min", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
```

```
576f6e096997/[CHANNEL1] - Current Frequency (Hertz)--Min.decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Current
Frequency (Hertz seconds)", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Current Frequency (Hertz seconds).decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Voltage
Frequency (Hertz)", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Voltage Frequency (Hertz).decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Voltage
Frequency (Hertz)--Max", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Voltage Frequency (Hertz)--Max.decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Voltage
Frequency (Hertz)--Min", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Voltage Frequency (Hertz)--Min.decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] - Voltage
Frequency (Hertz seconds)", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Voltage Frequency (Hertz seconds).decimals",
"/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-576f6e096997/[CHANNEL1] -
Channel(s)", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/[CHANNEL1] - Status", "/opt/productivity_link/esrv_f18ee848-736d-4bc0-
8c6f-576f6e096997/[CHANNEL1] - Version", "Sample #"
To Stop Logging  : [<CRTL>+<C>]
Fri Nov  5 11:39:54 2010, 33532177, 2, 931, 4, 0, 1, 6000, 2, 6519, 2, 6000, 2, 0, 0,
4, 0, 0, 4, 0, 0, 4, 0, 1, 4, 0, 0, 2, 0, 0, 2, 0, 0, 2, 0, 1, 2, 0, 4, 0, 4, 0, 4, 0,
4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 1, 1, 20091026, 1
Fri Nov  5 11:39:55 2010, 33538186, 2, 931, 4, 0, 1, 6009, 2, 6519, 2, 6000, 2, 0, 0,
4, 0, 0, 4, 0, 0, 4, 0, 1, 4, 0, 0, 2, 0, 0, 2, 0, 0, 2, 0, 1, 2, 0, 4, 0, 4, 0, 4, 0,
4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 1, 1, 20091026, 2
Fri Nov  5 11:39:56 2010, 33544186, 2, 931, 4, 0, 1, 6000, 2, 6519, 2, 6000, 2, 0, 0,
4, 0, 0, 4, 0, 0, 4, 0, 1, 4, 0, 0, 2, 0, 0, 2, 0, 0, 2, 0, 1, 2, 0, 4, 0, 4, 0, 4, 0,
4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 0, 4, 1, 1, 20091026, 3
```

Additional options can be used to make the readings more usable. In particular, the following options are useful:

- `--process` to direct the logger to use the suffix counters to compute real data. See Section 0 for details on suffix counters.
- `--output <f>` to direct the logger to save the data in a comma separated format in a text file (named `f`). See the *Intel® Energy Checker SDK Companion Applications User Guide* for details.

With these additional options, the command line is:

```
./pl_csv_logger /opt/productivity_link/esrv_f18ee848-736d-4bc0-8c6f-
576f6e096997/pl_config.ini --process --output ./esrv_log.csv
```

Once the logger is interrupted (hit <CTRL>+<C> in the shell window), the content of the output file can be reviewed in a spreadsheet application for further analysis. Figure 26 on page 164 shows a simple plot of the data in Microsoft Excel of the first 100 samples.

9.2.4 Stop ESRV

Two options are available to stop an ESRV instance.

Type <CTRL>+<C> in the shell window.

Use the `--stop` ESRV command.

The command below assumes that the UUID of the ESRV instance to stop is f18ee848-736d-4bc0-8c6f-576f6e096997.

Figure 27 shows the message displayed by the target ESRV in its shell. The illustrations were captured under Linux. Simply ignore the leading `./` in the command line under Windows. Type the following command in a shell:

```
./esrv --stop f18ee848-736d-4bc0-8c6f-576f6e096997
```

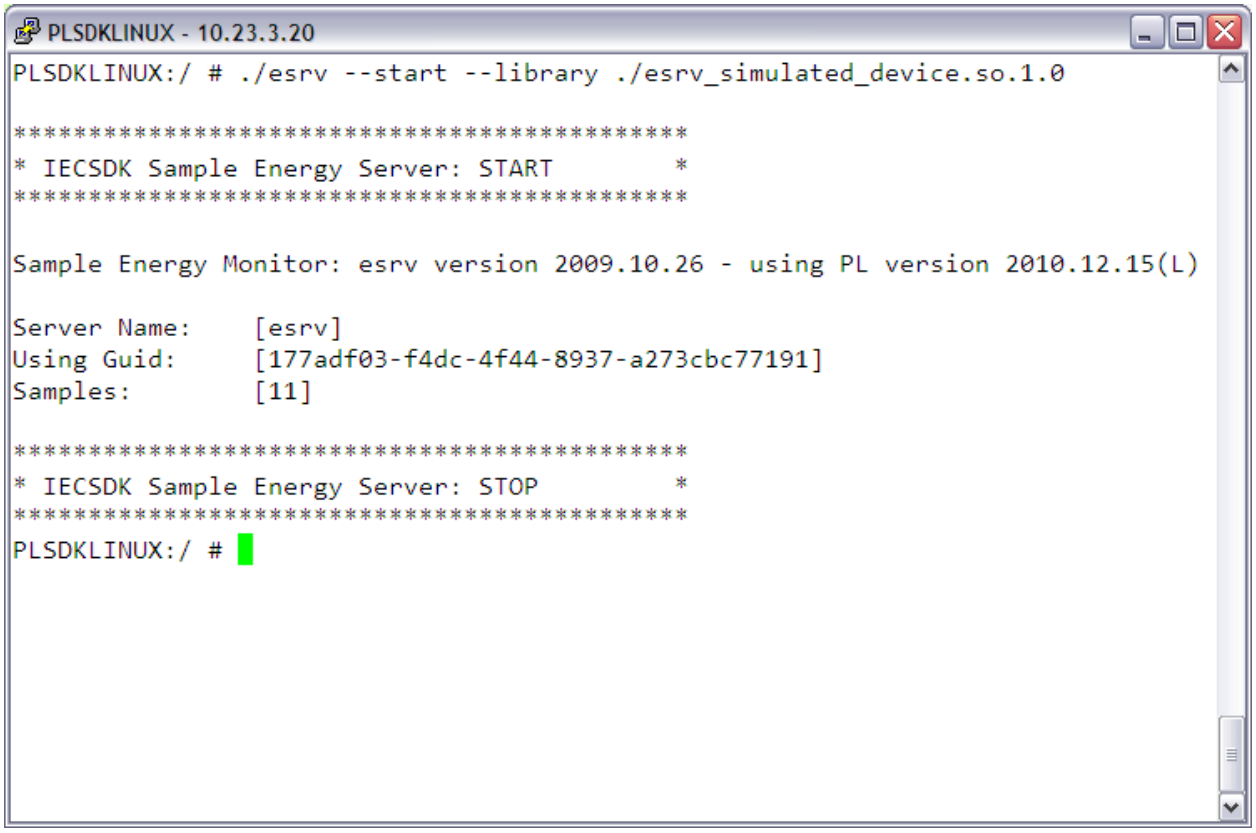

Figure 27: Stop message of ESRV.

9.3 Setting-up the WT210

This section will illustrate, step-by-step, how to install and configure the WT210 power reader to match the configuration expected by this tutorial. However, this is **NOT** a replacement of the Yokogawa user guide.

9.3.1 Cables and Connections

***** CAUTION *****

ALWAYS SWITCH OFF AND THEN DISCONNECT THE DEVICE FROM POWER BEFORE PERFORMING ANY CHANGES TO CONNECTIONS

To use the Yokogawa WT210 equiped with a serial interface, three connections must be set. Figure 28 shows the back of the device.

- Zone 1 (outlined in red) connects the power strip used to power the devices under test. For safety reasons, follow the instruction of the user guide to correctly make the connection. . A screw driver is required to remove the plastic protection cover.
- Zone 2 (outlined in green) is the serial interface. It is a femal DB25 port. It is recommended to have a DB25 male to DB9 femal cable for standard serial connections, or a serial-to-USB converter for USB connections.
- Zone 3 (outlined in purple) is the power socket of the WT210.

Two electric outlets are required for a functional setup. The first one will be used to power the device itself (Zone 3). The second socket will be used to power the power strip (attached to Zone 1).

Figure 28: WT210 connectivity.

Connect the components as shown in Figure 29.

Serial connectivity is shown on Zones 1 and 2 (green outlines). A serial-to-USB converter is used to connect the WT210 to a laptop (without standard serial interface).

Zone 6 (purple outine) shows the device's power plug (the device side of the cable is not called out in the figure).

Zone 3 (red outline on the power strip) is the power stip ***used to measure the power drawn and energy consumed by the system(s) under test***.

Zone 5 is the power plug of the power strip.

Zone 4 (red outline on the device) is the connection of the power strip to the device.

Plug the system(s) under test in the power strip (Zone 3).

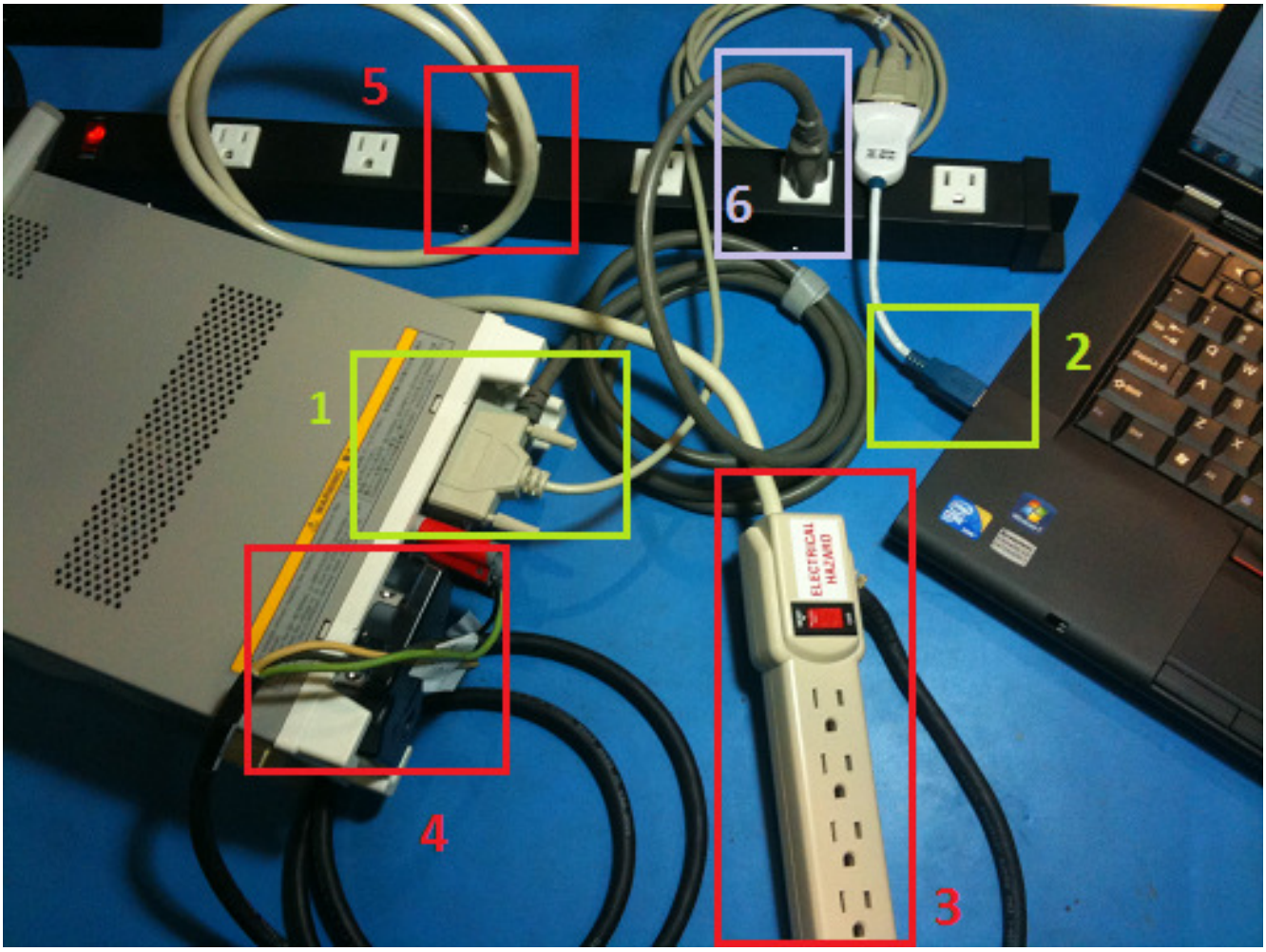

Figure 29: Overall connectivity.

9.3.2 Communication Settings

Once the WT210 connectivity is setup, the communication settings have to be configured.

Under non-Windows operating systems, no settings are required on the machine running ESRV.

Under Windows, use the Device Manager and select the serial port used to connect to the WT210.

Figure 30 shows the setting for the setup used to illustrate this mini tutorial. The settings shown in Figure 30 match the ESRV default settings.

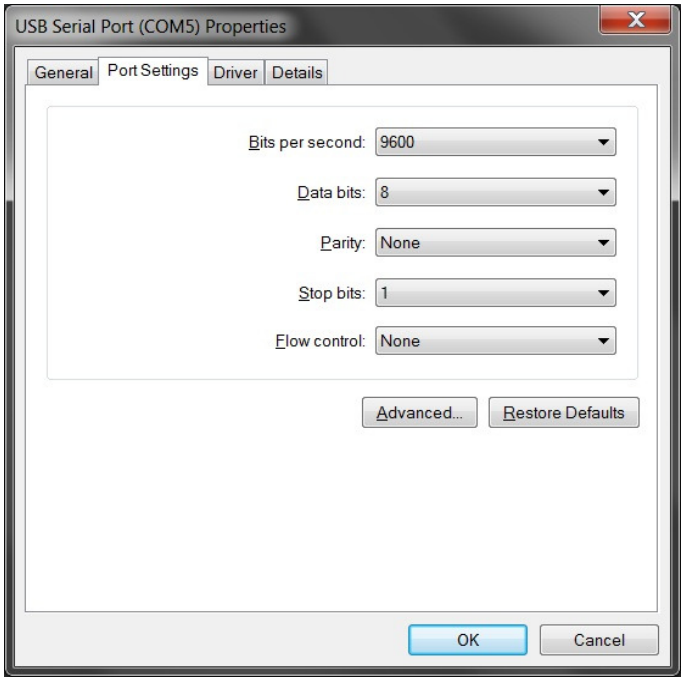

Figure 30: WT210 and controller serial settings must match.

The same settings have to be done on the WT210 side.

Switching the device into setting mode by pressing the LOCAL button on the device's panel (Figure 31).

Zone A of the display prints 232C and 488.2 blinks in zone C (Figure 32).

If this is not the case, press the UP / DOWN buttons to select 488.2.

Press ENTER.

hAnd followed by a flashing digit is displayed in zone A.

Since no handshake is used, press the UP / DOWN buttons (marked with symbols, not the words) to select hAnd0.

Press ENTER.

For followed by a flashing digit is displayed in zone B.

Since no flow control is used, press the UP / DOWN buttons to select hAnd0.

Press ENTER.

b followed by a flashing number is displayed in zone C.

Press the UP / DOWN buttons to select 9600 (Figure 33)

Press ENTER.

232C and dElin are displayed respectively in zone A and B. Characters are blinking in zone C (Figure 34).

Press ENTER.

The setup is complete.

NOTE

The UP and DOWN buttons are not labeled as up and down, but are marked with an up arrow and down arrow.

NOTE

The WT210 has three display zones named (from top down) A, B, and C. ESRV reprograms the display zones as zone A = Volatge, zone B = power, and zone C = energy.

Figure 31: Enter WT210's setting mode.

Figure 32: Typical display of a WT210 equiped with a serial interface.

Figure 33: Select the highest seep: 9600 baud.

Figure 34: Select LF.

9.3.3 Available ESRV Counters

To facilitate the use of ESRV counters using the API, a header file is provided in the SDK (`pub_esrv_counters.h` file in the `\iecsdk\src\energy_server` folder). This file should be included in applications that interface to ESRV. `pub_esrv_counters.h` (partially shown below) contains counter index definitions to be use with `pl_read()` API function calls.

```
1   //--------------------------------------------------------------------------
2   // counters definitions.
3   //--------------------------------------------------------------------------
4   typedef enum _esrv_counters_base_indexes {
5
6      //-----------------------------------------------------------------------
7      // mandatory counters
8      //-----------------------------------------------------------------------
9      ESRV_COUNTER_ENERGY_JOULES_INDEX = 0,
10     ESRV_COUNTER_ENERGY_JOULES_DECIMALS_INDEX,
11     ESRV_COUNTER_ENERGY_KWH_INDEX,
12     ESRV_COUNTER_ENERGY_KWH_DECIMALS_INDEX,
13     ESRV_COUNTER_ENERGY_OVERFLOWS_INDEX,
14     ESRV_COUNTER_UPDATE_FREQUENCY_INDEX,
15     ESRV_COUNTER_POWER_INDEX,
16     ESRV_COUNTER_POWER_DECIMALS_INDEX,
17     ESRV_COUNTER_MAX_POWER_INDEX,
18     ESRV_COUNTER_MAX_POWER_DECIMALS_INDEX,
19     ESRV_COUNTER_MIN_POWER_INDEX,
20     ESRV_COUNTER_MIN_POWER_DECIMALS_INDEX,
```

```
21
22      //-------------------------------------------------------------------------
23      // optional counters
24      // Note: counters below are optional and may not be updated or set by
25      // the device driver.  It is recommended that un-implemented counters
26      // are set to zero.
27      //-------------------------------------------------------------------------
28      ESRV_COUNTER_CURRENT_INDEX,
29      ESRV_COUNTER_CURRENT_SIGN_INDEX,
30      ESRV_COUNTER_CURRENT_DECIMALS_INDEX,
31      ESRV_COUNTER_MAX_CURRENT_INDEX,
32      ESRV_COUNTER_MAX_CURRENT_SIGN_INDEX,
33      ESRV_COUNTER_MAX_CURRENT_DECIMALS_INDEX,
34      ESRV_COUNTER_MIN_CURRENT_INDEX,
35      ESRV_COUNTER_MIN_CURRENT_SIGN_INDEX,
36      ESRV_COUNTER_MIN_CURRENT_DECIMALS_INDEX,
37      ESRV_COUNTER_CURRENT_SECONDS_INDEX,
38      ESRV_COUNTER_CURRENT_SECONDS_SIGN_INDEX,
39      ESRV_COUNTER_CURRENT_SECONDS_DECIMALS_INDEX,
40      ESRV_COUNTER_VOLTAGE_INDEX,
41      ESRV_COUNTER_VOLTAGE_SIGN_INDEX,
42      ESRV_COUNTER_VOLTAGE_DECIMALS_INDEX,
43      ESRV_COUNTER_MAX_VOLTAGE_INDEX,
44      ESRV_COUNTER_MAX_VOLTAGE_SIGN_INDEX,
45      ESRV_COUNTER_MAX_VOLTAGE_DECIMALS_INDEX,
46      ESRV_COUNTER_MIN_VOLTAGE_INDEX,
47      ESRV_COUNTER_MIN_VOLTAGE_SIGN_INDEX,
48      ESRV_COUNTER_MIN_VOLTAGE_DECIMALS_INDEX,
49      ESRV_COUNTER_VOLTAGE_SECONDS_INDEX,
50      ESRV_COUNTER_VOLTAGE_SECONDS_SIGN_INDEX,
51      ESRV_COUNTER_VOLTAGE_SECONDS_DECIMALS_INDEX,
52      ESRV_COUNTER_POWER_FACTOR_INDEX,
53      ESRV_COUNTER_POWER_FACTOR_DECIMALS_INDEX,
54      ESRV_COUNTER_MAX_POWER_FACTOR_INDEX,
55      ESRV_COUNTER_MAX_POWER_FACTOR_DECIMALS_INDEX,
56      ESRV_COUNTER_MIN_POWER_FACTOR_INDEX,
57      ESRV_COUNTER_MIN_POWER_FACTOR_DECIMALS_INDEX,
58      ESRV_COUNTER_POWER_FACTOR_SECONDS_INDEX,
59      ESRV_COUNTER_POWER_FACTOR_SECONDS_DECIMALS_INDEX,
60      ESRV_COUNTER_CURRENT_FREQUENCY_INDEX,
61      ESRV_COUNTER_CURRENT_FREQUENCY_DECIMALS_INDEX,
62      ESRV_COUNTER_MAX_CURRENT_FREQUENCY_INDEX,
63      ESRV_COUNTER_MAX_CURRENT_FREQUENCY_DECIMALS_INDEX,
64      ESRV_COUNTER_MIN_CURRENT_FREQUENCY_INDEX,
65      ESRV_COUNTER_MIN_CURRENT_FREQUENCY_DECIMALS_INDEX,
66      ESRV_COUNTER_CURRENT_FREQUENCY_SECONDS_INDEX,
67      ESRV_COUNTER_CURRENT_FREQUENCY_SECONDS_DECIMALS_INDEX,
68      ESRV_COUNTER_VOLTAGE_FREQUENCY_INDEX,
69      ESRV_COUNTER_VOLTAGE_FREQUENCY_DECIMALS_INDEX,
```

```
70      ESRV_COUNTER_MAX_VOLTAGE_FREQUENCY_INDEX,
71      ESRV_COUNTER_MAX_VOLTAGE_FREQUENCY_DECIMALS_INDEX,
72      ESRV_COUNTER_MIN_VOLTAGE_FREQUENCY_INDEX,
73      ESRV_COUNTER_MIN_VOLTAGE_FREQUENCY_DECIMALS_INDEX,
74      ESRV_COUNTER_VOLTAGE_FREQUENCY_SECONDS_INDEX,
75      ESRV_COUNTER_VOLTAGE_FREQUENCY_SECONDS_DECIMALS_INDEX,
76
77      //-----------------------------------------------------------------------
78      // esrv specific counters
79      //-----------------------------------------------------------------------
80      ESRV_COUNTER_CHANNELS_INDEX,
81      ESRV_COUNTER_STATUS_INDEX,
82      ESRV_COUNTER_VERSION_INDEX
83
84  } ESRV_COUNTERS_BASE_INDEXES;
```

9.4 Energy Efficiency Reporting Code Structure

The following pseudo-code snippet shows a typical use of ESRV counters in a C program to report energy efficiency from an application. Note that this code can be implemented as a thread, or as a process, if the work data can be exchanged via an IPC mechanism.

In this sample, the application connects automatically to the latest ESRV instance. Alternately, the developer could allow the user to identify the ESRV instance via a dialog box, command line argument, or similar mechanism.

The sample code computes and reports energy efficiency based on data collected during one sample period. An alternate approach is to use cumulative measurements of work and energy and then report average energy efficiency.

NOTE

The ESRV client sample code shipped with the SDK (`iecsdk\src\samples\esrv_client`) provides both thread and process implementations. Please refer to the sample code for more details. This sample code also shows how to use other ESRV counters such as `Channel(s)`, `Status`, and `.decimals` suffix counters.

NOTE

ESRV exposes a counter named Status (index ESRV_COUNTER_STATUS_INDEX) to indicate if the PL is active. ESRV sets this counter to a positive value while it is running and sets the status counter to zero when terminating.

```
#include "productivity_link.h"
#include "productivity_link_helper.h"
#include "pub_esrv_counters.h"

#define COUNTERS_COUNT 5
#define EE_COUNTERS_DECIMALS 4
#define EE_SCALE 10000.0 // 10 ^ 4

int main(void) {

...

   uuid_t uuid;
   char application_name[] = "My Application";
   const char *counters_names[COUNTERS_COUNT] = {
      "Work Units Done",
      "Energy Consumed (in Joules)",
      "Joule(s) per Work Unit",
      "Joule(s) per Work Unit.decimals",
      "Work Unit(s) per Joule",
      "Work Unit(s) per Joule.decimals"
   };

   enum {
      WORK_UNITS_DONE_INDEX = 0,
      ENERGY_CONSUMED_INDEX,
      JOULES_PER_WORK_UNIT_INDEX,
      JOULES_PER_WORK_UNIT_DECIMALS_INDEX,
      WORK_UNIT_PER_JOULE_INDEX,
      WORK_UNIT_PER_JOULE_DECIMALS_INDEX
   };

   unsigned long long energy = 0;
   unsigned long long consumed_energy = 0;
   unsigned long long reference_energy = 0;
   unsigned long long work_units = 0;
   unsigned long long work_units_done = 0;
   unsigned long long reference_work_units = 0;
   double joules_per_work_unit = 0.0;
   double work_unit_per_joule = 0.0;
   unsigned long long value = 0;

   char esrv_pl_config_file [MAX_PATH] = { 0 };
   int ret = PL_FAILURE;
   int pl_esrv = PL_INVALID_DESCRIPTOR;
   int pld = PL_INVALID_DESCRIPTOR;

...

```

```
50      //-------------------------------------------------------------------------
51      // attach to the latest ESRV instance
52      //-------------------------------------------------------------------------
53      memset(
54         esrv_pl_config_file,
55         0,
56         sizeof(esrv_pl_config_file)
57      );
58      ret = plh_get_young_pl_by_application_name(
59         "esrv",
60         &esrv_pl_config_file[0]
61      );
62      assert(ret != PL_FAILURE);
63      pl_esrv = pl_attach(file_energy);
64      assert(pl_esrv != PL_INVALID_DESCRIPTOR);
65
66      //-------------------------------------------------------------------------
67      // open application's PL
68      //-------------------------------------------------------------------------
69      pld = pl_open(
70         application_name,
71         COUNTERS_COUNT,
72         counters_names,
73         &uuid
74      );
75      assert(pld != PL_INVALID_DESCRIPTOR);
76
77      //-------------------------------------------------------------------------
78      // write-out static counters
79      //-------------------------------------------------------------------------
80      value = EE_COUNTERS_DECIMALS;
81      ret = pl_write(
82         pld,
83         &value,
84         JOULES_PER_WORK_UNIT_DECIMALS_INDEX
85      );
86      assert(ret != PL_FAILURE);
87      ret = pl_write(
88         pld,
89         &value,
90         WORK_UNIT_PER_JOULE_DECIMALS_INDEX
91      );
92      assert(ret != PL_FAILURE);
93
94   ...
95
96      //-------------------------------------------------------------------------
97      // read the reference energy
98      //-------------------------------------------------------------------------
```

```
 99      ret = pl_read(
100         pl_esrv,
101         &reference_energy,
102         ESRV_COUNTER_ENERGY_JOULES_INDEX
103      );
104      assert(ret != PL_FAILURE);
105
106   ...
107
108      while(1) {
109
110      ...
111
112         //---------------------------------------------------------------------
113         // read the instantaneous energy
114         //---------------------------------------------------------------------
115         ret = pl_read(
116            pl_esrv,
117            &energy,
118            ESRV_COUNTER_ENERGY_JOULES_INDEX
119         );
120         assert(ret != PL_FAILURE);
121
122         //---------------------------------------------------------------------
123         // compute energy consumed and useful work units done
124         //---------------------------------------------------------------------
125         consumed_energy = energy - reference_energy;
126         // colect the work units done (assumed monotonously increasing)
127         work_units_done = work_units - reference_work_units;
128
129         //---------------------------------------------------------------------
130         // write-out work units done and energy consumed
131         // Note: it is not required to re-export the energy consumed data, it
132         // is done here just for a demonstration purpose.
133         //---------------------------------------------------------------------
134         ret = pl_write(
135            pld,
136            &consumed_energy,
137            ENERGY_CONSUMED_INDEX
138         );
139         assert(ret != PL_FAILURE);
140         ret = pl_write(
141            pld,
142            & work_units_done,
143            WORK_UNITS_DONE_INDEX
144         );
145         assert(ret != PL_FAILURE);
146
147         //---------------------------------------------------------------------
```

```
148         // compute and write-out joules per work unit EE counter
149         //-------------------------------------------------------------------
150         if(work_units_done != 0) {
151            joules_per_work_unit =
152               (double)consumed_energy / (double)work_units_done;
153         } else {
154            joules_per_work_unit = 0.0;
155         }
156         value = (unsigned long long)(joules_per_work_unit * EE_SCALE);
157         ret = pl_write(
158            pld,
159            &value,
160            JOULES_PER_WORK_UNIT_INDEX
161         );
162         assert(ret != PL_FAILURE);
163
164         //-------------------------------------------------------------------
165         // compute and write-out work unit per joule EE counter
166         //-------------------------------------------------------------------
167         if(consumed_energy != 0) {
168            work_unit_per_joule =
169               (double)work_units_done / (double)consumed_energy;
170         } else {
171            work_unit_per_joule = 0.0;
172
173         }
174         value = (unsigned long long)(work_unit_per_joule * EE_SCALE);
175         ret = pl_write(
176            pld,
177            &value,
178            JOULES_PER_WORK_UNIT_INDEX
179         );
180         assert(ret != PL_FAILURE);
181
182      ...
183
184         //-------------------------------------------------------------------
185         // save reference work units done and energy for next sample
186         //-------------------------------------------------------------------
187         reference_work_units = work_units;
188         reference_energy = energy;
189
190      ...
191      }
192   ...
193   }
```

9.5 ESRV Client Sample Code

The SDK is shipped with sample code demonstrating the use of ESRV counters to expose energy efficiency counters from an application (`esrv_client.c` and `esrv_client.h` files in `iecsdk\src\samples\esrv_client` folder).

To run this code, use the `CUMULATIVE_EE` symbol to activate the cumulative energy efficiency reporting.

By default, sample-level energy efficiency reporting is performed.

9.5.1 Building and Running the Sample

Use the appropriate Makefile and/or solution project to build the sample code (see Section 0 for details on the build process). To run the sample, an instance of ESRV must be running. Refer to the *Intel® Energy Checker SDK Device Driver Kit User Guide* for details. The example shown below demonstrates operation on a Windows operating system. All steps can be run under the other supported OSes.

```
cd iecsdk\bin\energy_server\windows\X86
esrv --start --library esrv_simulated_device.dll
```

```
*****************************************
* IECSDK Sample Energy Server: START        *
*****************************************

Sample Energy Monitor: esrv version 2009.03.03 - using PL version 2009.07.01(W)

Server Name:    [esrv]
Using Guid:     [bb784b0b-659f-435e-9aae-e8fa73c32898]
Samples:        [3]
```

```
cd iecsdk\build\windows\iesdk\Release
esrv_sample
```

```
Application Name: [My EE Application]
ESRV Name:        [C:\productivity_link\esrv_bb784b0b-659f-435e-9aae-
e8fa73c32898\pl_config.ini]
Using Guid:       [ba86cfa7-347f-4967-9f7f-068b83b15d4c]
Cumulative EE:    [Yes]
Samples:          [4]
```

```
cd iecsdk\bin\companion_applications\pl_gui_monitor\windows\X86
pl_gui_monitor --process --gdiplus --transparency 20 --top --geometry "gauges=2x2"
```

In the PL GUI Monitor, browse to the folder and select the `pl_config.ini` file: `c:\productivity_link\My EE Application_ba86cfa7-347f-4967-9f7f-068b83b15d4c`

Click the *Cancel* button the second time the *Open file* dialog box is displayed (read the *Intel® Energy Checker SDK Companion Applications User Guide* for details).

NOTE

*The GUID shown above in bold (**`ba86cfa7-347f-4967-9f7f-068b83b15d4c`**) will change each time the application is run. Therefore, the location of the initialization file in the browse directions above will need to change accordingly.*

Figure 35 shows the output of the SDK's GUI monitoring tool attached to the sample's PL. As `CUMULATIVE_EE` symbol was defined during build, no work-per-watt information is provided. Later on, the same output is shown with a binary generated without defining the `CUMULATIVE_EE` symbol. In Figure 36 it is possible to read the display of work-per-watt data in addition of the work-per-joule data.

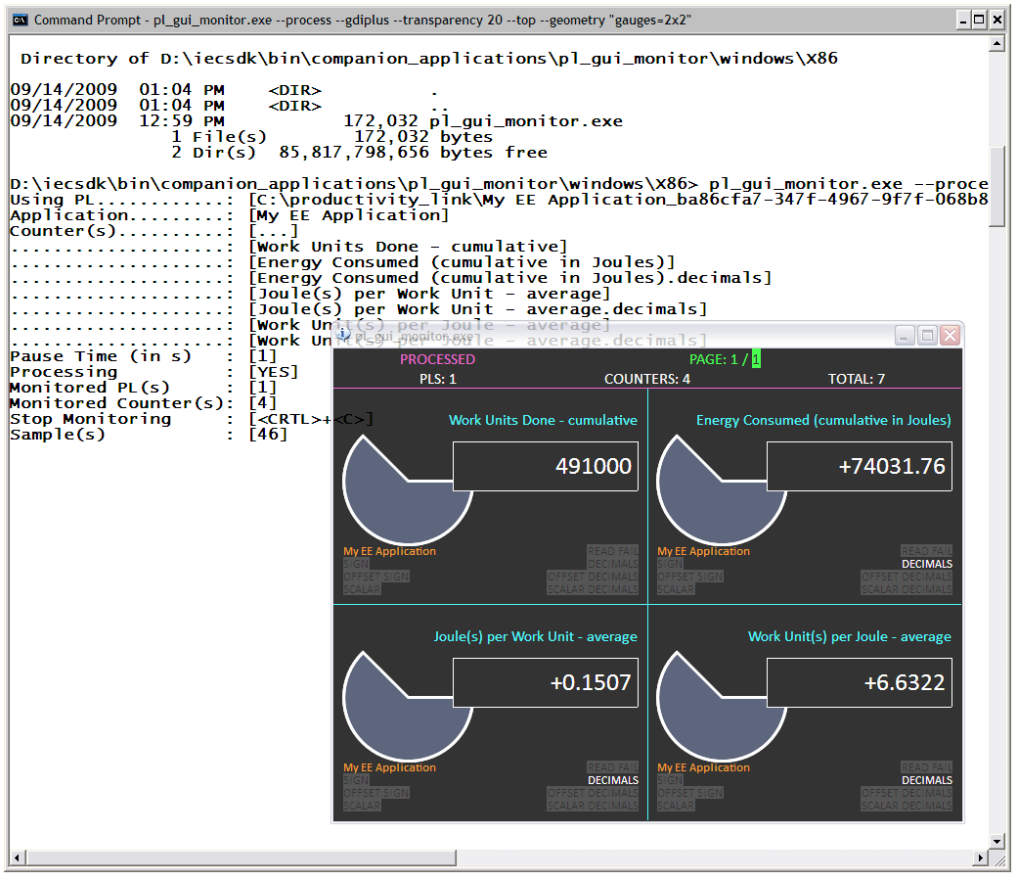

Figure 35: Sample data monitored with CUMULATIVE_EE symbol defined

Rebuild the sample without defining the `CUMULATIVE_EE` symbol and run the resulting binary as described above.

```
Application Name: [My EE Application]
ESRV Name:        [C:\productivity_link\esrv_bb784b0b-659f-435e-9aae-
e8fa73c32898\pl_config.ini]
Using Guid:       [9afbf9e7-e5d1-4c98-bea8-60e2d97b4b01]
Cumulative EE:    [No]
Samples:          [6]
```

```
pl_gui_monitor --process --gdiplus --transparency 20 --top --geometry "gauges=3x2"
```

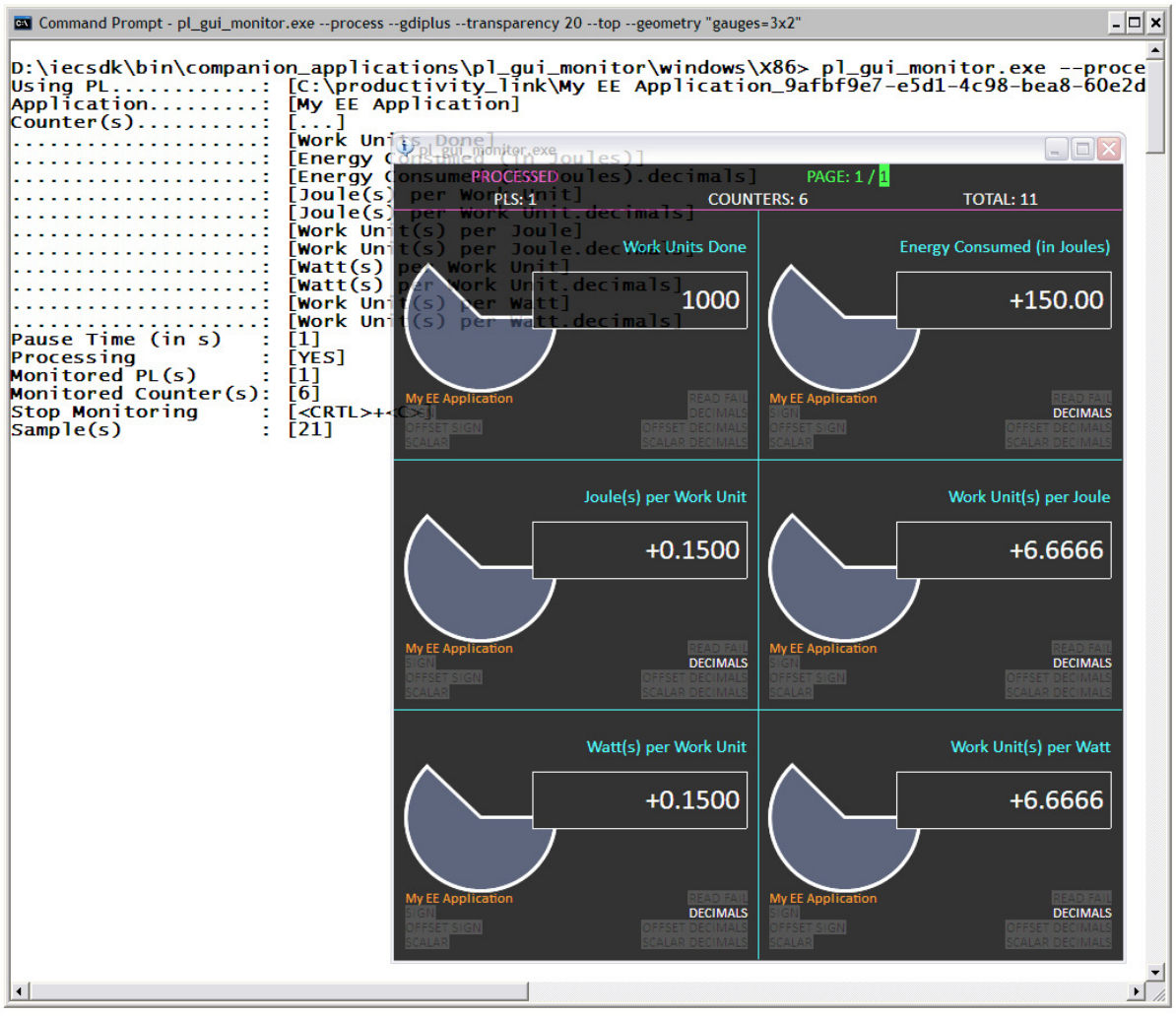

Figure 36: Sample data monitored without CUMULATIVE_EE symbol defined

9.6 Available TSRV Counters

The SDK is shipped with a utility to measure and report temperature and relative humidity (RH). The temperature and RH data is exposed using the API. Any application using the API can access TSRV data in a manner similar to ESRV.

To facilitate the use of TSRV counters using the API, a header file is provided in the SDK (`pub_tsrv_counters.h` in the \iecsdk`\src\energy_server` folder). This file should be included into each project. The `pub_tsrv_counters.h` file (partially shown below) contains counter index definitions to be used with the `pl_read()` API function calls.

```
1  //---------------------------------------------------------------------------
2  // counters definitions.
3  //---------------------------------------------------------------------------
4  typedef enum _tsrv_counters_base_indexes {
5
6        TSRV_COUNTER_TEMPERATURE_KELVIN_INDEX = 0,
7        TSRV_COUNTER_TEMPERATURE_KELVIN_DECIMALS_INDEX,
8        TSRV_COUNTER_MAX_TEMPERATURE_KELVIN_INDEX,
9        TSRV_COUNTER_MAX_TEMPERATURE_KELVIN_DECIMALS_INDEX,
10       TSRV_COUNTER_MIN_TEMPERATURE_KELVIN_INDEX,
11       TSRV_COUNTER_MIN_TEMPERATURE_KELVIN_DECIMALS_INDEX,
12       TSRV_COUNTER_TEMPERATURE_KELVIN_SECONDS_INDEX,
13       TSRV_COUNTER_TEMPERATURE_KELVIN_SECONDS_DECIMALS_INDEX,
14       TSRV_COUNTER_RELATIVE_HUMIDITY_PERCENTAGE_INDEX,
15       TSRV_COUNTER_RELATIVE_HUMIDITY_PERCENTAGE_DECIMALS_INDEX,
16       TSRV_COUNTER_MAX_RELATIVE_HUMIDITY_PERCENTAGE_INDEX,
17       TSRV_COUNTER_MAX_RELATIVE_HUMIDITY_PERCENTAGE_DECIMALS_INDEX,
18       TSRV_COUNTER_MIN_RELATIVE_HUMIDITY_PERCENTAGE_INDEX,
19       TSRV_COUNTER_MIN_RELATIVE_HUMIDITY_PERCENTAGE_DECIMALS_INDEX,
20       TSRV_COUNTER_RELATIVE_HUMIDITY_PERCENTAGE_SECONDS_INDEX,
21       TSRV_COUNTER_RELATIVE_HUMIDITY_PERCENTAGE_SECONDS_DECIMALS_INDEX,
22       TSRV_COUNTER_UPDATE_FREQUENCY_SECOND_INDEX,
23
24       //---------------------------------------------------------------------
25       // tsrv specific counters
26       //---------------------------------------------------------------------
27       TSRV_COUNTER_CHANNELS_INDEX,
28       TSRV_COUNTER_STATUS_INDEX,
29       TSRV_COUNTER_VERSION_INDEX
30
31 } TSRV_COUNTERS_BASE_INDEXES;
32
```

This page intentionally blank.

10 SDK Samples

This chapter describes the various samples found in the Intel Energy Checker SDK. The description of each sample will focus on the use of the API, rather than on the how-to of the program itself. For complex samples however, the overall structure is presented and code of interest is highlighted.

Each sample can be found in the `iecsdk\src\samples` folder of the SDK.

10.1 threading (Windows Implementation)

This sample was created for two reasons:

- To show that it is possible to safely use the API from threads.
- To show at least three (3) implementations of a threaded application using the API.

This sample is Windows-only as written, but could be applied to any OS the API supports.

The second part of the discussion provides a recommended approach to instrument a threaded application in a less intrusive way.

This sample can be compiled in three different ways by defining a `DEMO1`, `DEMO2`, or `DEMO3` symbol in the `main.c` file.

- `DEMO1` builds a threaded demo in which each worker threads own a PL and exposes a counter. A single counter per thread is used to simplify the code, but this can be easily modified in the source code.
- In `DEMO2`, the primary thread owns a PL and the counters for the worker threads.

Figure 37 illustrates this last organization.

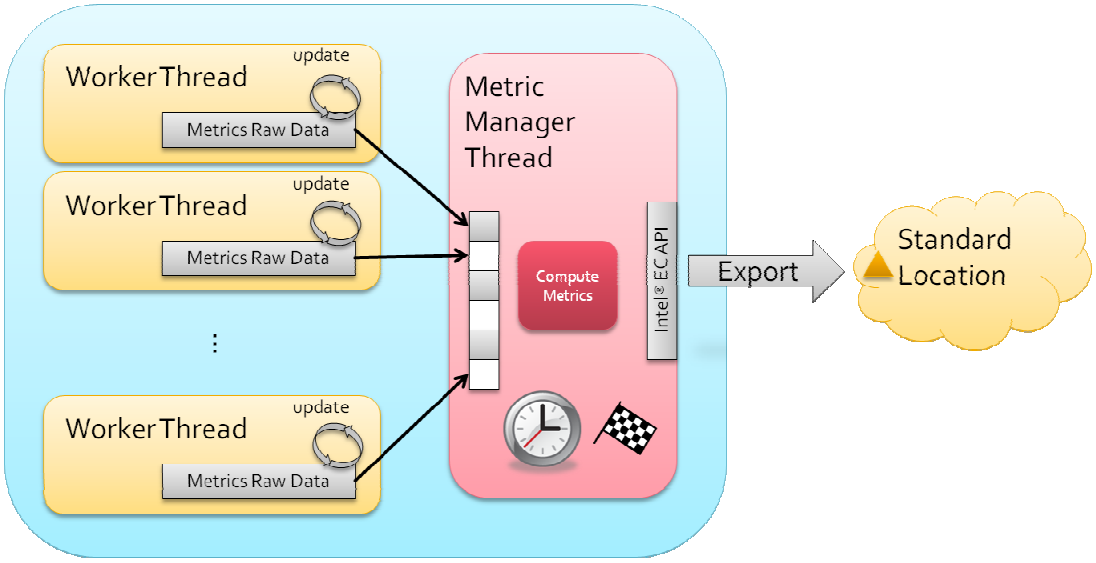

Figure 37: Recommended instrumentation for threader code

`DEMO3` is recommended for threaded code instrumentations, since it is the least intrusive for the application. It also facilitates decoupling the PL management from the actual work and the work unit accounting.

The work unit accounting (updating a counter in the collector/metric manager thread) has to be done within each worker thread. The counters updated by the worker threads could be simple, unsigned long long variables (eight bytes) in an array hosted by the metric manager thread, or they could be sophisticated structures hosted by the metric manager thread and accessed by reference from the worker threads. The decision is up to the application developer.

The actual reporting of the data via API can be adjusted as desired in the metric manager thread, without having to impact the individual worker threads. The metric manager thread can do whatever makes sense from a user point of view (never export counters, export them every four hours, report them every second, or even use to more complex criteria, such as every 20th work item recorded or every 5 minutes, whichever comes first). Of course, the collector thread and the metric(s) computation code — if any — can be aggregated into a single source file.

NOTE

For best results, be sure to align the counter variables to a cache line boundary in memory. This may alleviate a potential performance degradation issue known as "false sharing". Correct alignment (with the compiler using padding where necessary) resolves this problem.

10.2 esrv_client

This sample shows how to use the ESRV power and energy counters from an application. Please refer to Section 8 for more details.

10.3 java_calling_sample

This sample shows how to use the API from Java code. Please refer to Section 0 for more details.

10.4 csharp_calling_sample (Windows-only)

This sample shows how to use the API from .NET in C#. Please refer to Section 0 for more details.

10.5 objectivec_calling_sample (MacOS X-only)

This sample shows how to use the API from Objective-C via a wrapper class. Please refer to Section 0 for more details.

10.6 sample_logger

A logger is an application that records counter values over time for later analysis. It can collect data from one or multiple sources to aggregate them and allow cross-analysis of the data. This sample logger is a very basic program demonstrating such a capability.

- At startup, the sample logger asks which PL it should use for the work performed data source.
- Then, the sample logger asks which PL it should use for the energy consumed data source.
- Finally, it asks the name to use to save the log file.

In the results and various outputs reproduced here, the sample threading code from Section 10.1 is used with DEMO3 mode enabled.

After logging starts, each write into the log file is time stamped.

The sampling frequency used by the logger is 1 Hz (one sample per second).

The logger is interrupted when the user presses <CTRL>+<C>.

For the sake of simplicity in this example code, only the first counters of each PL are logged.

NOTE

This logger sample is very basic. The PL CSV Logger sample shipped with the SDK is a more complete logger and should be studied after this sample. In particular, it shows how to log multiple PLs and how to process counters and their associated suffix counters to compute and log counter real values. The PL CSV

Logger is a companion application of the SDK and can be found in the `iecsdk\src\companion_applications\pl_csv_logger` *folder.*

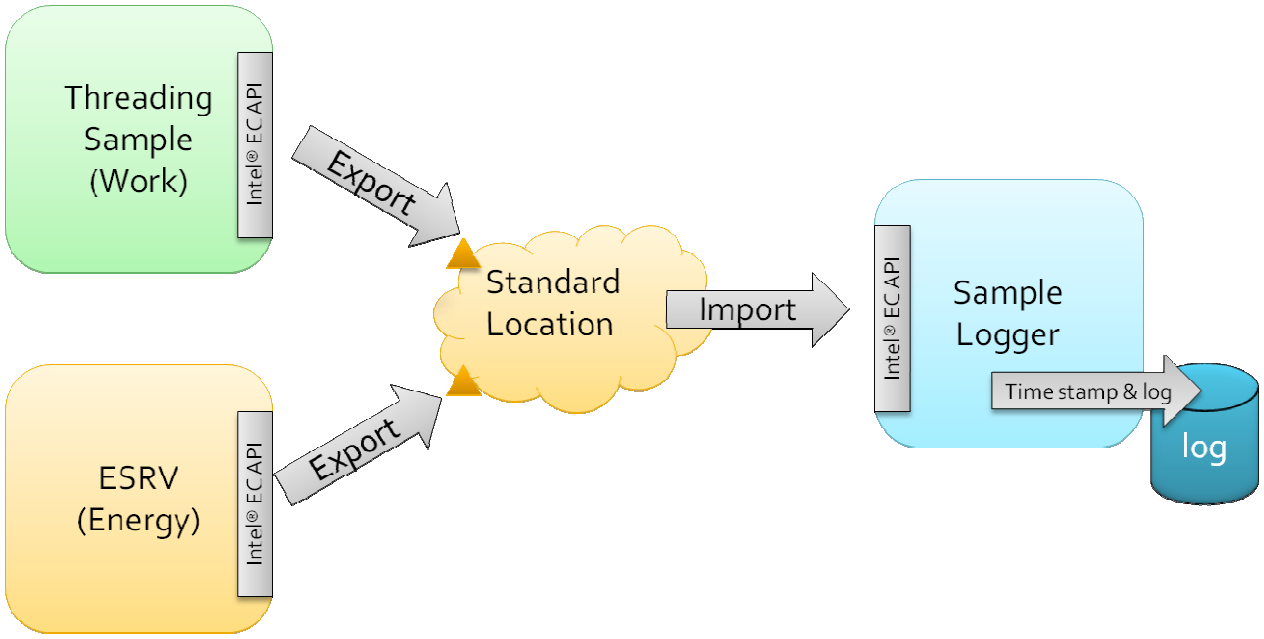

Figure 38: Sample logger setup

ESRV must be started before starting the sample_logger.

The listing below shows the messages of ESRV at startup using the diagnostic option.

```
esrv --start --device y210 --diagnostic
```

```
DIAG: ...Registering init_yokogawa_wt210_extra_data Function. [SERVER]
DIAG: ...Registering delete_yokogawa_wt210_extra_data Function. [SERVER]
DIAG: ...Registering open_yokogawa_wt210 Function. [SERVER]
DIAG: ...Registering close_yokogawa_wt210 Function. [SERVER]
DIAG: ...Registering parse_yokogawa_wt210_option_string Function. [SERVER]
DIAG: ...Registering read_yokogawa_wt210_power Function. [SERVER]
DIAG: ...Registering read_yokogawa_wt210_energy Function. [SERVER]
DIAG: Setting-up Inter-Process Communication. [COMMON]
DIAG: Energy Server Starts. [SERVER]

*****************************************
* IECSDK Sample Energy Server: START        *
*****************************************

Sample Energy Monitor: esrv version 2009.03.03 - using PL version 2009.03.06(W)

DIAG: Setup Thread Synchronization Data. [SERVER]
```

```
DIAG: Parsing Interface Options. [SERVER]
DIAG: Initializing Device Extra Data - First Call. [SERVER]
DIAG: No Device Options String To Parse. [SERVER]
DIAG: Initializing Device Extra Data - Second Call. [SERVER]
DIAG: Opening Interface. [SERVER]
DIAG: ...Opening Serial Port. [SERVER]
DIAG: ...Configuring Serial Port. [SERVER]
DIAG: ...Purging Serial Port. [SERVER]
DIAG: Opening Device. [SERVER]
DIAG: Installing <CTRL>+<C> Signal Handler. [SERVER]
DIAG: Spawning Collector Thread. [SERVER]
DIAG: Starting Inter-Process Communication Controller. [SERVER]
DIAG: ...Preparing PL Counters Data. [KL-SERVER]
DIAG: ...Opening PL. [KL-SERVER]
DIAG: Initializing Command Processing. [ML-COMMAND]
Using Guid: [753c2e96-ee12-4bc9-90db-17209e8b4190]
DIAG: ...Using The Guid [753c2e96-ee12-4bc9-90db-17209e8b4190]. [KL-SERVER]
DIAG: ...Writing Static Counters. [KL-SERVER]
DIAG: ...Channel [#1] Queried. [KL-SERVER]
DIAG: ...Initialize Hardware Energy Integration. [KL-SERVER]
DIAG: ...Start Hardware Energy Integration. [KL-SERVER]
Samples:    [147]
```

The workload now can be started. The listing below shows the messages of the altered threading sample code.

```
threading_long_run
```

```
Primary Thread Is Running.
Worker Thread 0 Is Running.
Worker Thread 1 Is Running.
Worker Thread 2 Is Running.
Worker Thread 4 Is Running.
Worker Thread 6 Is Running.
Worker Thread 8 Is Running.
Worker Thread 3 Is Running.
Collector Thread Is Running.
Worker Thread 5 Is Running.
Worker Thread 7 Is Running.
Worker Thread 9 Is Running.
```

Finally, the sample_logger is started. Once the required information is provided, the sample starts logging until the user interrupts it. The output below shows the various sources used and destination (in the form of Work PL + Energy PL = log).

```
sample_logger
```

```
Select the PL configuration file for the work counter.
Select the PL configuration file for the energy counter.
Select the log file name and location.
Logging. <CTRL>+<C> to stop logging.
Using [C:\productivity_link\collector_thread_cf0b38af-2e2f-4c27-8ab5-
fc9e886ba6ee\pl_config.ini] + [C:\productivity_link\esrv_753c2e96-ee12-4bc9-90db-
17209e8b4190\pl_config.ini] -> [C:\productivity_link\esrv_753c2e96-ee12-4bc9-90db-
17209e8b4190\log]
```

The listing below shows an extract of the log file generated by the sample code.

```
Time Stamps, Sample Count, Work, Energy
Thu Mar 12 17:41:50 2009, 0, 24, 2491488
Thu Mar 12 17:41:51 2009, 1, 25, 2522268
Thu Mar 12 17:41:52 2009, 2, 26, 2583792
Thu Mar 12 17:41:53 2009, 3, 27, 2583792
Thu Mar 12 17:41:54 2009, 4, 28, 2614536
...
Thu Mar 12 17:42:19 2009, 29, 53, 3414240
Thu Mar 12 17:42:20 2009, 30, 54, 3414240
Thu Mar 12 17:42:21 2009, 31, 55, 3445019
Thu Mar 12 17:42:22 2009, 32, 56, 3475764
Thu Mar 12 17:42:23 2009, 33, 57, 3506543
```

Once the sample logger is terminated, it is possible to import the log file into a spreadsheet program for further analysis (Figure 39).

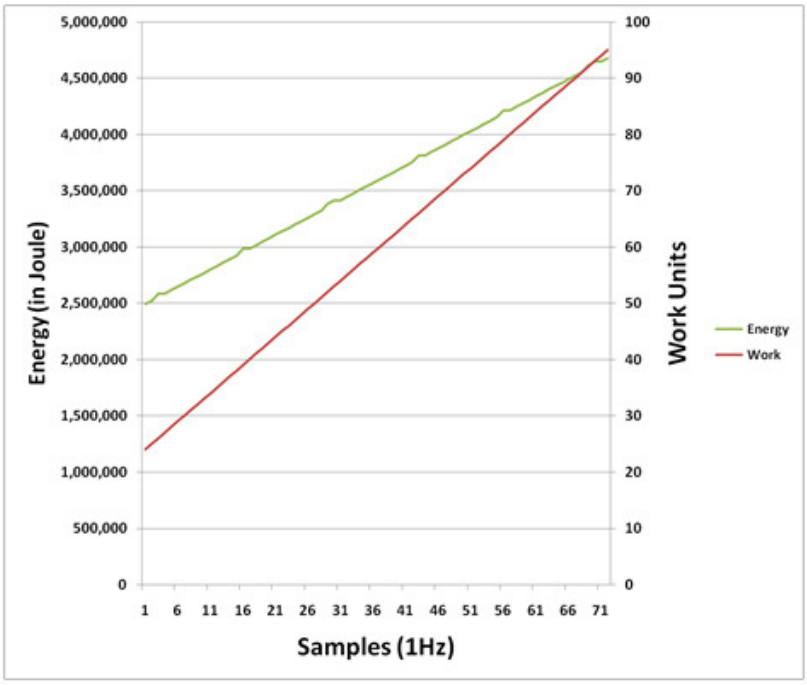

Figure 39: Graph of log file content

10.7 Energy Consummed by a Command –energy Sample

The energy sample reports the energy consumed between the invocation of the command and its termination.

An example of the output generated by the energy sample is:

```
Energy: [6016.03] Joule(s) – [0.00167112] kWh(s).
```

Because there is no way to measure the power consumed by a given process with today's platforms – by sheer lack of suitable hardware hooks – only the energy consumed by the entire system is reported by the energy sample. This sample also demonstrates two important tasks a programmer may have to perform when writing an energy-aware application.

- The first task is to programmatically drive the ESRV utility shipped with the SDK. ESRV is the SDK tool in charge of driving a power analyzer and sampling the power and energy readings. Even if it is simple to manually use ESRV, in some cases – like in this sample – the developer may want to shield the user from this complexity.
- The second task is to access and use the energy data provided by ESRV via its PL. This last task is detailed in the esrv_client sample of the SDK (`\iecsdk\src\samples\esrv_client`).

10.7.1 Build the Sample

Use the Microsoft Visual Studio 2005 project (right-click on the project and select `build`) or the Makefiles to build the agent (type `make energy` to start the build process).

10.7.2 Sample Structure

`energy` either starts a new instance of ESRV, or attaches to an existing instance of the ESRV to measure the energy consumed. This behavior is selected by the user's input. Whichever path is taken, once the energy utility has attached to the correct ESRV's PL, it will run the command the user provides . The energy counter (`([CHANNELx] – Energy (Joule)`) of the PL is read at the beginning, when the command begins to execute, and at the end, when the command ends. Note that these are the only times the energy counters of the PL are read.

The energy consumed while the command is run is computed as the difference of these two readings. It is reported in joules and in kWh (one kWh = 3,600,000 joules).

Once started, it is possible to interrupt the execution of the command by typing the <CTRL>+<C> key sequence. If this happens, then the end energy is read when the interrupt signal is processed by the signal handler of the sample.

10.7.3 Run the Sample

Because `energy` drives ESRV in a programmatic way, the user can specify which instance to attach to. This is done with the `--guid` option to specify the GUID of the instance to use. Otherwise, an environment variable can be used. It is then assumed that the environment variable holds the GUID of the ESRV instance to attach to. Section 0 shows how to programmatically set such an environment variable. By default, `energy` is seeking an ESRV_GUID-named environment variable. Another name can be specified using the `--guid_shell_variable` option. If none of these options works, `energy` starts and stops its own instance of ESRV.

The following shows three invocation examples:

```
1  energy -- sleep 10
2  energy --guid fd3e6616-d71c-4b12-8e2a-bee7e3cde0e2 -- sleep 10
3  energy --guid_shell_variable ESRV_GUID -- sleep 10
```

The following listing shows the energy help output from `energy --help`.

```
1
2   energy give the energy usage for a simple command.
3
4   Use: energy [options] -- command [arguments...]
5
6   [--guid <guid>]:
7      Read energy data from esrv instance identified by the guid.
8   [--guid_shell_variable <variable>]:
9      Read energy data from esrv instance identified by the guid stored in shell
10  variable.
11  [--channel <integer>]:
12     Use esrv channel.  By default, channel 1 is used.
13  [--esrv_options <string>]:
14     Start an esrv instance with options to read energy data from.
15     By default, the options used are: --device y210 and esrv default settings.
16     Use esrv --help for details on esrv defaults.
17  [--version]:
18     Print version info and exit successfully.
19  [--help]:
20     Print this help message and exit successfully.
21
22  Notes:
23     -  Energy is given in Joule and kWh -- 1 kWh equals 3,600,000 Joules.
24     -  If neither --guid nor --guid_shell_variable options are used, then energy
25  starts an
26     instance of esrv.  This instance is terminated and its PL is suppressed at
27     the end of the command execution.  Starting an esrv instance may add some
28     execution time at startup.
29     -  Esrv and support library -- if applicable -- must be in the search path or
```

```
    in the execution folder.  This also includes the library path under non-Windows*
OSes.
    -  User can interrupt the measurement by typing <CTRL>+<C>.  The command will
    be ended when the next output will be printed.
    -  Options --guid and --guid_shell_variable are mutually exclusive.

 Examples:
    energy -- dir c:\ /s
    energy --guid 4d3e6616-d71c-4b12-8e2a-bee7e3cde0e2 -- dir c:\ /s
    energy --guid_shell_variable ESRV_GUID -- dir c:\ /s
-al /usr

```

10.7.4 Automating energy (Linux implementation)

To fully automate the use of the energy sample, it is possible to create an ESRV startup process that sets the environment variables to be used by `energy`. Indeed, with no user options, `energy` searches for a ESRV_GUID environment variable.

To set this environment variable a simple script can be used. This script (listed below) first terminates any running instance of ESRV. It then starts a new instance of ESRV, using the options provided in the script source, and then updates the `.bashrc` script of the user. By doing this, each time the user logs-in (assuming that BASH is the default shell), the ESRV_GUID is set and ready to use. In addition to the script, a gawk program is used to modify the `.bashrc` script.

The listing of the `start_esrv` script is:

```
#!/bin/bash
# Note: run as . ./start_script
#-------------------------------------------------------------------------
# Start displaying information to stdout
#-------------------------------------------------------------------------
clear
echo ESRV CONFIGURATION
echo ==================

#setenforce 0
#-------------------------------------------------------------------------
# Stop esrv.
#-------------------------------------------------------------------------
echo Ending esrv instance...
killall -q esrv

#-------------------------------------------------------------------------
# Set paths
#-------------------------------------------------------------------------
```

```
ESRV_BINARY_PATH=/bin
ESRV_LIBRARY_PATH=/lib

#-------------------------------------------------------------------------
# Set esrv options.
# Note:
#    The grace time is required if the device to be used by esrv requires
#    a long start and configuration time.  If the time is not long enough
#    then this script may miss the output.  Other option is to loop
#    until the output file is created.
#-------------------------------------------------------------------------
ESRV_SHELL_VARIABLE_NAME=ESRV_GUID
ESRV_START_GRACE_TIME_IN_SECONDS=5
ENERGY_IN_HUNDREDS_OF_JOULE=0
ENERGY_IN_HUNDREDS_OF_KWH=2
POWER_IN_HUNDREDS_OF_W=6

#-------------------------------------------------------------------------
#ESRV_LIBRARY=yokogawa_wt210.so
#ESRV_LIBRARY=yokogawa_wt230.so
#ESRV_LIBRARY=yokogawa_wt500.so
#ESRV_LIBRARY=yokogawa_wt3000.so
#ESRV_LIBRARY=esrv_simulated_device.so
ESRV_LIBRARY=build_in_yokogawa_wt210

#-------------------------------------------------------------------------
# Create the gawk script to collect the GUID for our PL instance to read.
# gawk script file is saved in gawk.script file.
#-------------------------------------------------------------------------
echo "{ if(\$1==\"Using\") { print(substr(\$3, 2, length(\$3) - 2)); } }" >
./gawk.script

#-------------------------------------------------------------------------
# Start esrv instance
#-------------------------------------------------------------------------
echo Starting esrv instance...
esrv --start --device y210 --interface_options "com=0" --device_options "items=all" --
shell  > ./esrv.out &
echo "Pausing for $ESRV_START_GRACE_TIME_IN_SECONDS second(s)..."
sleep $ESRV_START_GRACE_TIME_IN_SECONDS

#-------------------------------------------------------------------------
# Collect the GUID for our PL instance created by this instance of esrv.
#-------------------------------------------------------------------------
gawk -f ./gawk.script ./esrv.out > ./guid.out

#-------------------------------------------------------------------------
# Export ESRV_GUID variable
#-------------------------------------------------------------------------
```

```
GUID=`cat ./guid.out`
echo "$ESRV_SHELL_VARIABLE_NAME=$GUID" > ./set_guid.script
source ./set_guid.script

#-------------------------------------------------------------------------
# Cleanup
#-------------------------------------------------------------------------
rm -rf ./gawk.script
rm -rf ./esrv.out
rm -rf ./guid.out
rm -rf ./set_guid.script

#-------------------------------------------------------------------------
# Display information to stdout
#-------------------------------------------------------------------------
echo ------------------
echo "ESRV_SHELL VARIABLE = "[$ESRV_SHELL_VARIABLE_NAME]
echo "ESRV GUID           = "[$GUID]
echo "ESRV LIBRARY        = "[$ESRV_LIBRARY]

#-------------------------------------------------------------------------
# Update UUID in .bashrc
#-------------------------------------------------------------------------
echo Updating .bashrc script...
gawk -f /bin/start_esrv.awk -v esrv_guid=$ESRV_GUID ~/.bashrc > ~/.bashrc.tmp
rm -f ~/.bashrc
cp -f ~/.bashrc.tmp ~/.bashrc
rm -f ~/.bashrc.tmp
echo Done.
```

The listing of the `start_esrv.awk` gawk program is:

```
BEGIN {

        /*************************************************************/
        /* Note:                                                     */
        /* esrv guid is passed using -v option                       */
        /* invoked with ... -v esrv_guid=$GUID ...                   */
        /*************************************************************/

        /*************************************************************/
        /* Sample of the line to filter-out                          */
        /* export ESRV_GUID = 1785ae67-d50f-4478-93e6-4a93de8e746d   */
        /* export ENERGY_IN_HUNDREDS_OF_JOULE = 0                    */
        /* export ENERGY_IN_HUNDREDS_OF_KWH = 2                      */
        /* export POWER_IN_HUNDREDS_OF_W = 6                         */
        /*************************************************************/
        FS="=";
```

```
17  }
18
19  {
20          if($1 == "export ESRV_GUID") {
21                  next;
22          }
23          if($1 == "export ENERGY_IN_HUNDREDS_OF_JOULE") {
24                  next;
25          }
26          if($1 == "export ENERGY_IN_HUNDREDS_OF_KWH") {
27                  next;
28          }
29          if($1 == "export POWER_IN_HUNDREDS_OF_W") {
30                  next;
31          } else {
32                  print $0;
33          }
34  }
35
36  END {
37
38          /*********************************************************/
39          /* Dump esrv variables settings.                         */
40          /*********************************************************/
41          printf("%s%s\n", "export ESRV_GUID=", esrv_guid);
42          printf("%s\n", "export ENERGY_IN_HUNDREDS_OF_JOULE=0");
43          printf("%s\n", "export ENERGY_IN_HUNDREDS_OF_KWH=2");
44          printf("%s\n", "export POWER_IN_HUNDREDS_OF_W=6");
45  }
```

NOTE

The script also sets the following environment variables that can be used with the scripting tools.

- *ENERGY_IN_HUNDREDS_OF_JOULE*
- *ENERGY_IN_HUNDREDS_OF_KWH*
- *POWER_IN_HUNDREDS_OF_W*

10.8 Linux System Information Utilities (Linux only)

The Linux system information utility samples demonstrate how to use the API to collect basic OS metrics and expose them via counters. There are four sample applications located in the `linux_system_infos` folder:

- linux_cpu_and_loadavg_info
- linux_diskstat_info
- linux_mem_info
- linux_vmstat_info

Each sample is built using a very similar structure. The differences essentially are the input data and the parsing required to extract counter values.

Once the data is collected, the counters are exposed using the `pl_write()` API function.

The PL is opened at startup and closed at termination.

The linux_cpu_and_loadavg_info sample uses the `/proc/loadavg` and `/proc/stat` files. linux_diskstat_info uses the `/proc/diskstats` file. linux_mem_info uses the `/proc/meminfo` file. linux_vmstat_info uses the `/proc/vmstat file`.

Table 9 lists the CPU and disk counters exposed by these sample utilities.

Table 9: CPU and disk counters

Linux CPU and Load Info (19 counters)	Linux Disk Stat Info (11 counters)
CPU usage (percentage)	# of reads completed
CPU usage (percentage).decimals	# of reads merged
Utilization of the last minute period	# of sectors read
Utilization of the last minute period.decimals	# of milliseconds spent reading
Utilization of the last 5 minutes period	# of writes completed
Utilization of the last 5 minutes period.decimals	# of writes merged
Utilization of the last 10 minutes period	# of sectors written
Utilization of the last 10 minutes period.decimals	# of milliseconds spent writing
# of currently running processe(s)	# of IOs currently in progress
Total number of processes	# of milliseconds spent doing IOs
Last process ID used	weighted # of milliseconds spent doing IOs
jiffies in user mode (jiffy = ~4ms)	
jiffies in low priority user mode - nice - (jiffy = ~4ms)	
jiffies in system mode mode (jiffy = ~4ms)	
jiffies in idle task mode (jiffy = ~4ms)	
jiffies in IO wait (jiffy = ~4ms)	
jiffies in hard IRQ - hardirq - (jiffy = ~4ms)	
jiffies in soft IRQ - softirq - (jiffy = ~4ms)	
jiffies in steal (jiffy = ~4ms)	

Table 10 lists the memory and virtual memory counters exposed by these sample utilities.

Table 10: Memory and virtual memory counters

Linux VM Stat Info (44 counters)	Linux Mem Info (28 counters)
nr_dirty	MemTotal
nr_writeback	MemFree
nr_unstable	Buffers
nr_page_table_pages	Cached
nr_anon_pages	SwapCached
nr_mapped	Active
nr_slab	Inactive
pgpgin	HighTotal
pgpgout	HighFree
pswpin	LowTotal
pswpout	LowFree
pgalloc_high	SwapTotal
pgalloc_normal	SwapFree
pgalloc_dma32	Dirty
pgalloc_dma	Writeback
pgfree	AnonPages
pgactivate	Mapped
pgdeactivate	Slab
pgfault	CommitLimit
pgmajfault	Committed_AS
pgrefill_high	PageTables
pgrefill_normal	VmallocTotal
pgrefill_dma32	VmallocUsed
pgrefill_dma	VmallocChunk
pgsteal_high	HugePages_Total
pgsteal_normal	HugePages_Free
pgsteal_dma32	HugePages_Rsvd
pgsteal_dma	Hugepagesize
pgscan_kswapd_high	
pgscan_kswapd_normal	
pgscan_kswapd_dma32	
pgscan_kswapd_dma	
pgscan_direct_high	
pgscan_direct_normal	
pgscan_direct_dma32	
pgscan_direct_dma	
pginodesteal	
slabs_scanned	

Linux VM Stat Info (44 counters)	Linux Mem Info (28 counters)
kswapd_steal	
kswapd_inodesteal	
pageoutrun	
allocstall	
pgrotated	
nr_bounce	

10.9 cpu_use_histogram (Windows Implementation)

The idea behind the cpu_use_histogram sample is to have a novel view of the CPU utilization of a system. Rather than tracking the CPU utilization over time (in percent as the OS displays it), cpu_use_histogram enables tracking CPU utilization as a distribution over time. This sample is Windows only, but the concepts apply to other operating systems as well. This same approach can be used for monitoring network utilization, tracking disk subsystem utilization, recording time at given temperatures, and similar applications.

NOTE

A simplified version of a CPU usage histogram implemented as a Linux shell script can be found in Section 0.

The cpu_use_histogram program performs the following operations:

- At startup, it creates a PL with 103 counters. Each CPU use percentage (0% - 100%) has its own counter, plus two counters to record the maximum and the minimum CPU usage. Internally, an array of `unsigned long long` counters is used to record the data.
- The code then samples the CPU use (Figure 40) once per second until interrupted by the user (<CTRL>+<C>). This information is directly collected from Windows. The entry of the array corresponding to the sampled CPU use percentage is incremented. The min and max variables are updated and the modified counters exported.
- At the end of the run, the distribution data is printed to stdout in comma separated values (CSV) format, so it can be re-used in a spreadsheet application for further analysis.

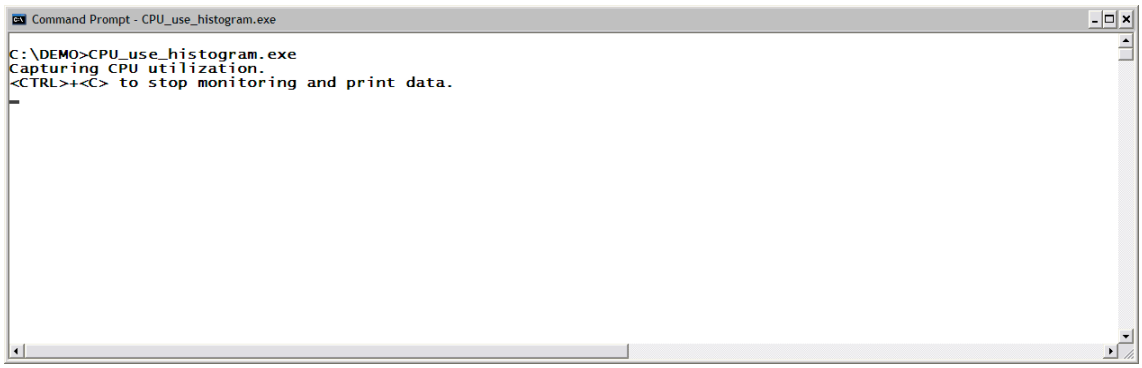

Figure 40: CPU_use_histogram sample

10.9.1 Integration with Windows Utilities

Used in conjunction with pl2w conversion utility (see Section 7.1), one can directly observe the histogram building-up over time in Perfmon (Figure 41 and Figure 42). Although this step is not required for the data collection, it demonstrates how one can use existing and standard tools to monitor Intel EC counters.

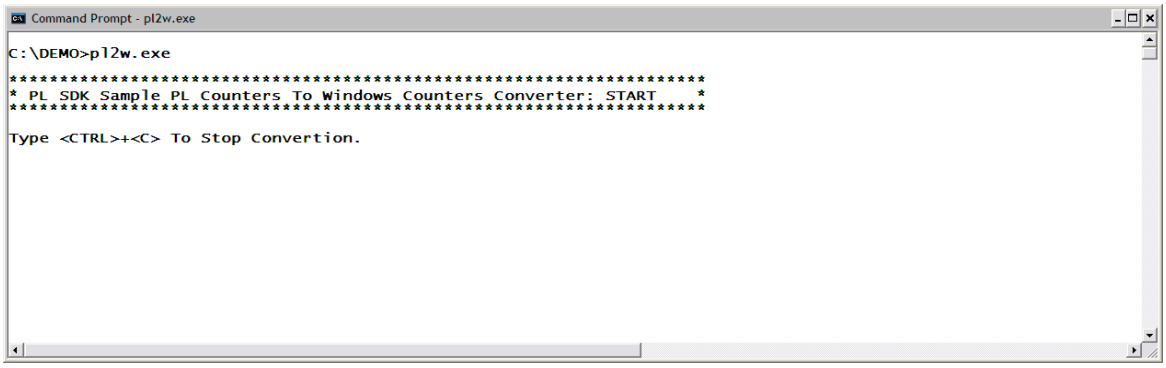

Figure 41: Converting CPU_use_histogram counters with pl2w

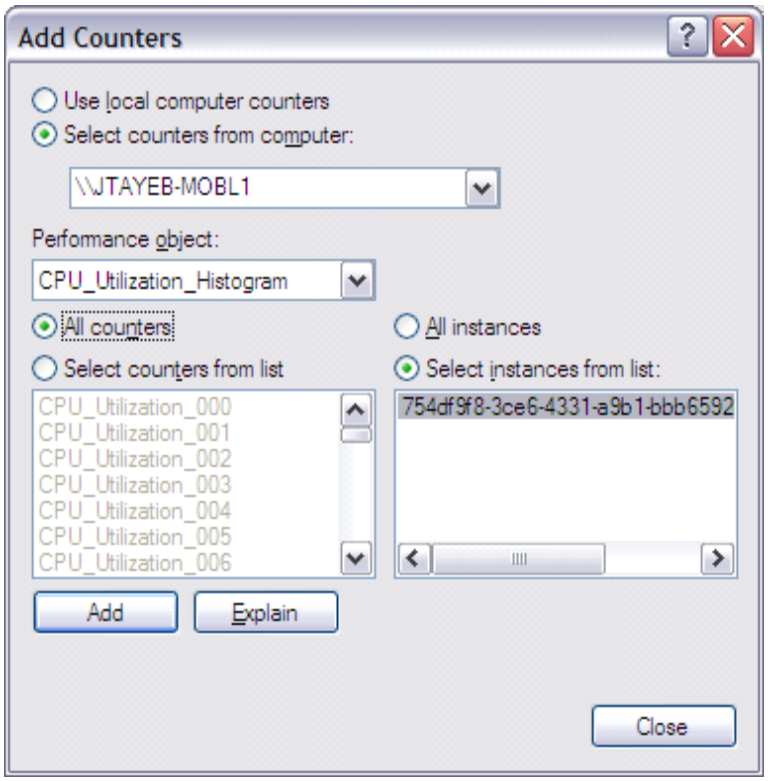

Figure 42: Adding the CPU_use_histogram counters to Perfmon

Taking this one step further, it is possible to use Intel EC to do cross-platform analysis. For example, a Linux system's PL counters could be monitored from a Windows system, and the Linux data could be viewed on a Windows system through the use of Perfmon and pl2w (refer to Section 12.1 for details). Figure 43 shows the distribution captured at different times on the same system.

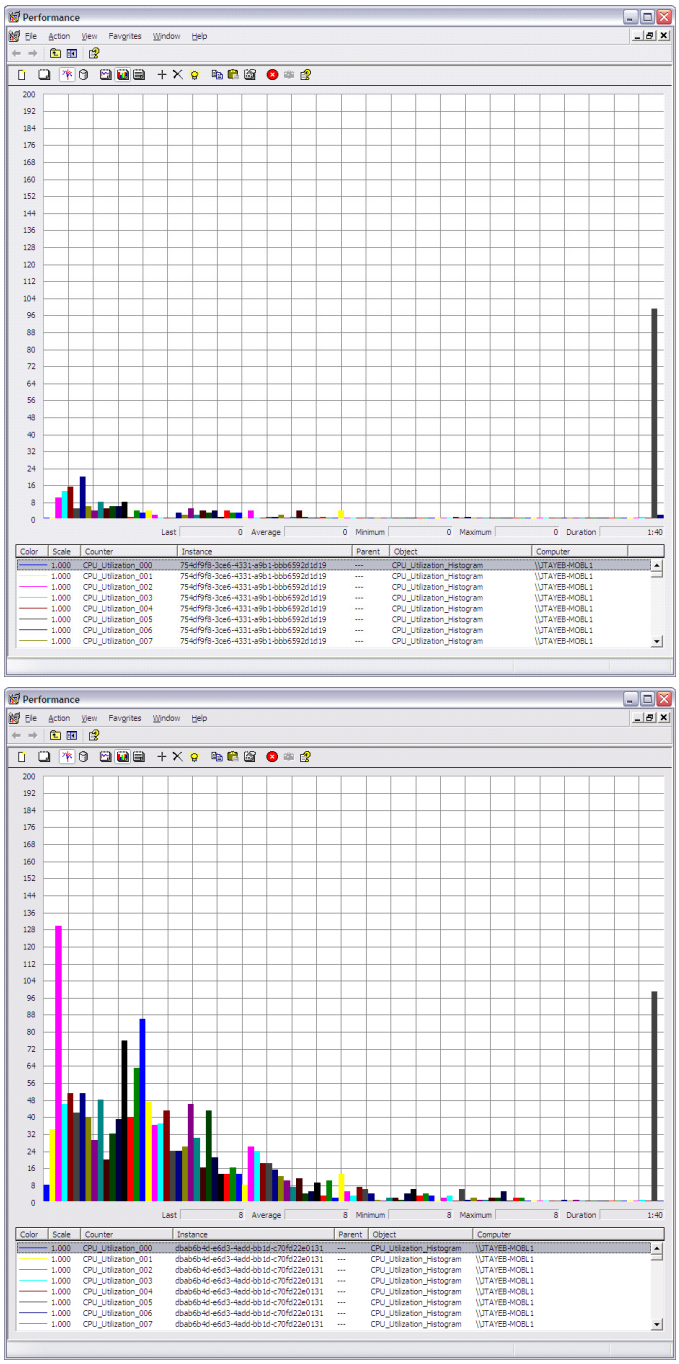

Figure 43: Two cpu_use_histogram snapshots

10.10 Cluster Energy Efficiency

The CEE (Cluster Energy Efficiency) sample shows how to handle the PL counters exposed by multiple instances of the same application running on different systems (Figure 44). This situation can be found in High-Performance Computing (HPC) environments, where clusters are used to run the same code to process different data sets of a same problem.

Assuming that the application was instrumented using the Intel EC SDK to expose the amount of useful work generated, this sample shows how an application can aggregate the amount of useful work done by all the nodes and compute the energy efficiency metric of the cluster(s).

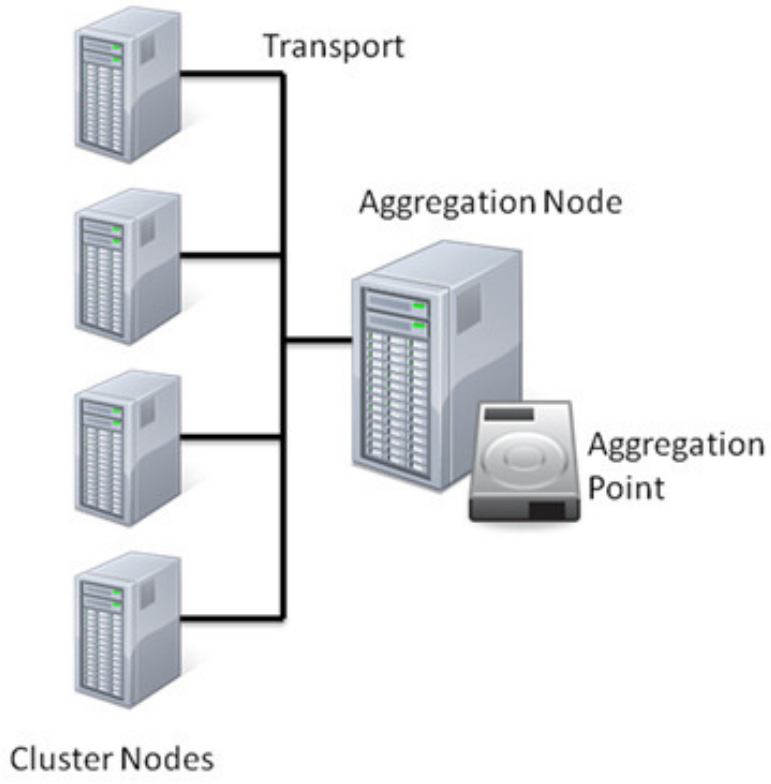

Figure 44: Typical CEE deployment

In such a scenario, the energy measurement can be done at different points. It can be achieved on each node, in or out-of-band, using a power sensor embedded into the PSUs (Power Supply Unit); Or, it can be done at the PDU (Power Distribution Unit) level for the cluster(s). The CEE sample can be configured to support either mode by defining (or not defining) the __CEE_ESRV_ON_EACH_NODE__ and __CEE_ESRV_ON_AGGREGATION_NODE__ symbols at compilation time.

In addition to this feature, the CEE sample demonstrates how it is possible to create a thread pool to parallelize the PL data collection from the cluster nodes and to load-balance the task. This allows the collection time to be optimized in case of a large cluster and / or if a low performance network is used by the distributed file system. In the latter case, it is always possible to build the sample in file system-less mode (define the __PL_FILESYSTEM_LESS__ symbol) and to use the PL Agent sample to aggregate the data locally on the node where CEE runs.

10.10.1 Build the Sample

A Microsoft Visual Studio 2005 solution and several Makefiles are shipped with the SDK to build the CEE sample. A set of symbols that can be set at compilation time are used to configure the sample's behavior. Table 11 summarizes these symbols. More information is available in the source code as comment blocks.

Table 11: CEE sample symbols and associated functions

Symbols	Functionalities
__CEE_ESRV_ON_EACH_NODE__	An instance of ESRV is supposed to run on each node of the cluster, measuring the energy consumption for that node.
__CEE_ESRV_ON_AGGREGATION_NODE__	An instance of ESRV is supposed to run on the aggregation node, measuring the energy consumed by the entire cluster.
__CEE_LIVE_MONITORING__	Activate the creation of a PL, allowing for real-time monitoring of the cluster energy efficiency.
__CEE_LOAD_BALANCING__	Activate the load ballancer. Required for big clusters. System running CEE must be SMP.
__CEE_FAVOR_TIMER_PRECISION_OVER_POWER_DRAW__	If precision issues are noted, use this symbol. However, it is not recommended.

The following counters are made available in a PL when live monitoring is activated (__CEE_LIVE_MONITORING__ symbol defined – Figure 45 on page 212):

- Node(s) aggregated
- Sampling interval (seconds)
- Sample(s) collected
- Accumulated work unit(s)
- Accumulated energy (Joule)
- Accumulated energy (Joule).decimals
- Accumulated energy (kWh)
- Accumulated energy (kWh).decimals
- Average energy per work unit (Joule per wu)
- Average energy per work unit (Joule per wu).decimals
- Average energy per work unit (kWh per wu)

- Average energy per work unit (kWh per wu).decimals
- Average work unit(s) per energy (wu per Joule)
- Average work unit(s) per energy (wu per Joule).decimals
- Average work unit(s) per energy (kWh per wu)
- Average work unit(s) per energy (kWh per wu).decimals
- Average work unit(s) per node (wu per node)
- Average work unit(s) per node (wu per node).decimals
- Average energy per node (Joule per node)
- Average energy per node (Joule per node).decimals
- Average energy per node (kWh per node)
- Average energy per node (kWh per node).decimals
- Average energy per work unit per node (Joule per wu per node)
- Average energy per work unit per node (Joule per wu per node).decimals
- Average energy per work unit per node (kWh per wu per node)
- Average energy per work unit per node (kWh per wu per node).decimals
- Average work unit(s) per energy per node (wu per Joule per node)
- Average work unit(s) per energy per node (wu per Joule per node).decimals
- Average work unit(s) per energy per node (wu per kWh per node)
- Average work unit(s) per energy per node (wu per kWh per node).decimals

10.10.2 Sample Structure

The CEE sample has a simple structure. CEE runs in the background; it does not interact directly with ESRV and cluster application instances.

- CEE starts by performing a discovery of the PL_FOLDER accessible from the system on which it runs.
- Based on its configuration and the user input, it counts the number of cluster application PLs and ESRV(s).
- Based on this count, it sets up data structures to handle the PLs.
- Based on the workload (PLs count), if load balancing is requested by the user, the pool of threads is created. Each thread is assigned a range of PLs to process. If the workload is not big enough, no load balancing threads are create.

If the sampling of the PLs is ever performed from the main thread's measurement loop, or from the load balancing threads, the work done is similar and can be summarized as:

- Loop
  - Import the cluster applications' work unit counter.
  - Import ESRV(s)' energy counter.

- Compute energy efficiency statistics.
- Export live monitoring counters' values – if required.

- Until interrupted by user or the requested samples count is reached.

10.10.3 Run the Sample

CEE is supposed to run on the cluster's aggregation node. It can be any system on the same network as the cluster. It is highly recommended for the aggregation node to have multiple processors, so the data collection can run in parallel. The aggregation node should mount the PL_FOLDER of all cluster nodes on its PL_FOLDER. Depending on the mode selected, one or many ESRV instances and each application instance should be run prior to starting CEE.

The following listing was generated by `cee --help`.

```
-------------------------------------------------------------------------------
 @@@@  @@@@@@ @@@@@@
@    @ @      @
@      @      @
@      @@@@@@ @@@@@@
@      @      @
@    @ @      @
 @@@@  @@@@@@ @@@@@@
-------------------------------------------------------------------------------
Cluster Energy Efficiency: version 2010.01.30
Using PL helper version 2009.05.18
Using PL version 2010.12.15(W)

 d:\cee.exe start a cluster energy efficiency sampling.

 [--pause <integer>]:
    Set the sampling interval in second(s). 1 by default.
 [--work_counter_index <integer>]:
    Set the work counter index. 0 by default.
 [--application_name <string>]:
        Set the application's name to <string>.
        It is set to "anonymous_application" by default.
 [--esrv_on_each_node]:
    Specify that ESRV runs on each node.
    By default it is assumed to run on the aggregation node.
 [--load_balancing [integer]]:
    Activate the load balancer.
    Optional [integer] sets the thread count. 2 by default,
    2 at minimum and 128 at maximum.
 [--samples <integer>]:
    End analysis after <integer> samples were collected.
    Sampling ends by default when interrupted by user (<CTRL>+<C>).
 [--help]:
    Print this help message.
```

```
35
36  This utility assumes the following:
37    - Each cluster node runs the same application.
38    - Each application PL is aggregated in one node.
39    - Application accumulates work and exposes it in
40      a single counter without suffix.
41    - This utility is run on the aggregated node.
42    - ESRV can either run on each node or on the aggregation node.
43
44    - Only channel 1 of ESRV is taken in account.
45
46  Notes:
47    - Load balancing is activated only if there are more than 1 nodes.
48    - If application name has spaces, use double-quotes (") around it.
49    - Several behaviors are set at compilation by:
50        __CEE_ESRV_ON_EACH_NODE__
51        __CEE_ESRV_ON_AGGREGATION_NODE__
52        __CEE_LOAD_BALANCING__
53        __CEE_LIVE_MONITORING__
```

NOTE

Samples shipped in the SDK show how ESRV instances can be programmatically managed. These features can be easily added to control ESRV instances either from the distributed application (__CEE_ESRV_ON_EACH_NODE__ mode), or from CEE itself (__CEE_ESRV_ON_AGGREGATION_NODE__ mode).

NOTE

For efficiency reasons, it is recommended to measure the energy consumed by the cluster, rather than on each node. The use of a PDU can help in this configuration. The SDK supports a PDU (see the Intel® Energy Checker Device Dirver Kit User Guide for details). For different PDUs, an ESRV support module has to be developed.

For example, the command below computes the energy efficiency of a cluster running an application named Cluster Application, requesting load balancing. The binary was built to perform energy measurement at the PDU level (from the aggragation node). The listing shows the output from the command.

```
cee --application_name "Cluster Application" --load_balancing
```

```
--------------------------------------------------------------------------------
 @@@@  @@@@@@ @@@@@@
@    @ @      @
@      @      @
@      @@@@@@ @@@@@@
@      @      @
@    @ @      @
 @@@@  @@@@@@ @@@@@@
--------------------------------------------------------------------------------
Cluster Energy Efficiency: version 2010.01.30
Using PL helper version 2009.05.18
Using PL version 2010.12.15(W)
--------------------------------------------------------------------------------
Sampling cluster energy efficiency data.
Type <CTRL>+<C> to stop sampling and print statistics.
--------------------------------------------------------------------------------
Binary built with the following symbol(s) defined:
   __PL_WINDOWS__.
   __CEE_ESRV_ON_AGGREGATION_NODE__.
   __CEE_LOAD_BALANCING__.
   __CEE_LIVE_MONITORING__.
--------------------------------------------------------------------------------
Application:                                                [Cluster Application]
Node(s):                                                    [6]
ESRV(s):                                                    [1]
Thread(s):                                                  [2]
--------------------------------------------------------------------------------
ESRV on aggregation node only:                              [Yes]
Load-balancing requested:                                   [Yes]
--------------------------------------------------------------------------------
Samples:                                                    [135]
```

If live monitoring is activated (__CEE_LIVE_MONITORING__ symbol defined), then it is possible to use the pl_gui_monitor to follow, in real-time, the cluster energy efficiency as shown in Figure 45.

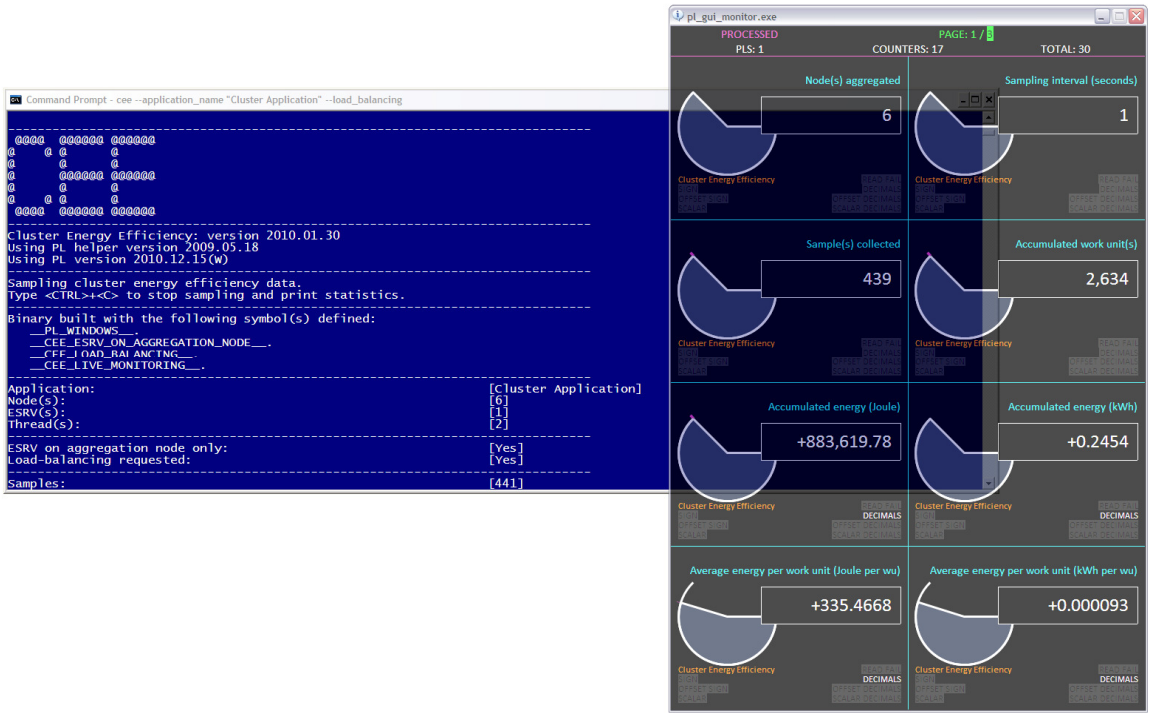

Figure 45: CEE sample monitored with pl_gui_monitor.

To stop CEE, simply press <CTRL>+<C>. A report, similar to the one listed below is printed out. Note that neither the ESRV instance(s), nor the application instances are stopped by CEE.

```
--------------------------------------------------------------------------------
                            *** Sampling Report ***
--- ALL NODES ------------------------------------------------------------------
Node(s) aggregated:                                                 [6]
Sampling interval (seconds):                                        [1]
Sample(s) collected:                                                [813]
Accumulated work unit(s):                                           [4871]
Accumulated energy (Joule(s)):                                      [939736]
Accumulated energy (kWh(s)):                                        [0.261038]
--- AVERAGE ALL NODES ----------------------------------------------------------
Average energy per work unit (Joule/wu):                            [192.925]
Average energy per work unit (kWh/wu):                              [5.35902e-005]
Average work unit per energy (wu/Joule):                            [0.00518337]
Average work unit per energy (wu/kWh):                              [18660.1]
--- AVERAGE PER NODE -----------------------------------------------------------
Average work unit per node (wu/node):                               [811.833]
```

```
Average energy per node (Joule/node):                                [156623]
Average energy per node (kWh/node):                                  [0.0435063]
Average energy per work unit per node (Joule/wu/node):               [32.1541]
Average energy per work unit per node (kWh/wu/node):                 [8.9317e-006]
Average work unit per energy per node (wu/Joule/node):               [0.000863895]
Average work unit per energy per node (wu/kWh/node):                 [3110.02]
-------------------------------------------------------------------------------
                            *** End of Report ***
-------------------------------------------------------------------------------
```

10.11 PL Agent

The PL agent sample shows how to write a server – or agent – to carry out API calls. The calls are issued by applications compiled with the __PL_FILESYSTEM_LESS__ symbol defined (client applications).

This symbol was introduced to allow instrumented applications to run on a filesystem-less machine, e.g. a smart phone, a table, or an embedded diskless system. Though most modern operating systems offer efficient distributed file systems, some users may want to depend solely on TCP/IP communications.

When the __PL_FILESYSTEM_LESS__ symbol is defined, each call to the API is sent to an agent for processing. The agent is identified using the PL_AGENT_ADDRESS and PL_AGENT_PL_PORT environment variables. If not defined, the 127.0.0.1 address and port 49253 are used. Note that these environemental variables are read at each time a `pl_open()` or `pl_attach()` is performed by a client code (instrumented code compiled with the __PL_FILESYSTEM_LESS__ symbol defined).

The agent will carry-out the API call and send back a set of data to the client. The PL protocol is used during these exchanges. See Section 3.9.5 for details on the PL protocol.

NOTE

When built in debug mode, the pl_agent displays a verbose log of all operations. This is useful when debugging an application in filesystem-less mode.

NOTE

When built with the __PL_DUMMY_AGENT__ symbol defined, the agent will not carry-out the API calls requested and will return a valid successful answer to the clients. This mode can be useful to develop a client application when an agent is not available.

NOTE

The __PL_IN_MEMORY__ symbol is reserved for future developments of the pl_agent.

The pl_agent is a multithreaded server. The following threads are used:

- ADMIN port listener and admin request processing thread.
- PL port listener thread.
- Pool threads (1 to 64, 16 by default).

The ADMIN port (port 49252 by default) is dedicated to receive management commands. The communication and the support threads are implemented in the sample, but no commands are implemented with the exception of the UUID look-up service (described later). Also, provision is made in the code for encryption, but no encryption is implemented. Encryption / decription call sites are marked in the code (tests on `p->f_encrypt`). Note that the encryption flag can be set by the user through the CLI.

10.11.1 Build pl_agent

Use the Microsoft Visual Studio 2005 project (right-click on the project and select `build`) or the Makefiles to build the agent (type `make pl_agent` to start the build process). Note that an agent must be built with the __PL_AGENT__ symbol defined. Do not define the __PL_FILESYSTEM_LESS__ symbol when building an agent. For pl_agent, all the symbols are pre-set appropriately.

10.11.2 Sample Structure

pl_agent's primary thread spawns the threads listed previously and waits until the agent is interrupted by the user (<CTRL>+<C>) and all threads have terminated. The PL port listener thread receives the API calls from the clients using the PL protocol. This thread searches for an idle worker thread in the pool. Once an idle thread is found, it is requisitioned so it can handle the client's request from there. The PL port listener thread then returns to waiting for the next API call request.

Once activated, a pool thread reads in the API data, decodes it and executes the API call. By default, the API call is executed locally on the system hosting the agent. The result of the call – plus the extra data required by the protocol – is sent back to the client. Statistical data is updated and then the threads return to the pool as idle.

The following functions can be studied in the sample:

- pl_port_listener_thread
- pool_thread
- admin_port_listener_thread

10.11.3 Run the Sample

The listing below shows the pl_agent help message (`pl_agent --help`).

Without any argument an agent is started with 16 worker threads, listening on port 49252 for ADMIN requests and port 49253 for PL API calls. These ports can be changed, so multiple agents can be hosted on a single system. It is possible to to set up load balancing, by setting the PL_AGENT_ADDRESS,

PL_AGENT_PL_PORT and PL_AGENT_ADMIN_PORT environment variables according to the target pl_agent instance's options on the client side. Since the environment variables on the client side are read for each `pl_open()` or `pl_attach()` calls, it is possible to dynamically modify the loadballencing scheme.

```
Start PL agent for instrumented applications compiled in __PL_FILESYSTEM_LESS__ mode.

Usage:  pl_agent [--admin_port <port>] [--pl_port <port>]
                 [--threads_in_pool <n>] [--encrypt]
Or      pl_agent [--version]
Or      pl_agent [--help]

   --admin_port specifies a TCP/IP port number to use for administrative tasks.
      By default, port 49252 is used.
   --pl_port specifies a TCP/IP port number to use by pl functions.
      By default, port 49253 is used.
   --threads_in_pool specifies the number of threads created to process PL calls.
      By default, 16 threads are created. (with 1 <= n <= 64)
   --encrypt uses encripted communications via the admin port.
   Notes:
      Only dynamic and/or private ports are accepted (49152 through 65535).
      Use netstat command to display current TCP/IP network connections.
   --version prints version information.
   --help prints this help message.

To interrupt the agent, type <CTRL>+<C>.
```

By default, 16 pool threads are created. Use the `--threads_in_pool` option to modify this value.

Note that a limitation to 64 threads is because of a Windows limitation. To overcome this limitation on Windows, one can restructure the sample and create a pool thread hierarchy, where a master pool thread could spawn up to 64 pool threads. If more than 4096 threads are required (this is not recommended), an additional hierarchy level could be introduced.

Once started, the agent is ready to receive requests from the clients. To end the agent, simply press <CTRL>+<C>.

10.12 Multi-touch Interface (MTI) Sample (Windows 7 only)

While most samples focus on server oriented environments, the Multi-Touch Interface (MTI) sample shows how one can use the SDK to instrument a GUI-based interactive software to compute energy efficiency metrics.

At its core, the MTI sample is a very simple application that allows the user to draw and manipulate a Bézier curve. The user simply defines in-roder, the first anchor point, the first control point, the second anchor point, and finally the second and last control point.

Figure 46 shows an example of such a Bézier curve. Note that the size and the color of each graphic element are designed to allow quick recognition – even via a computer vision system if required – and an easy manipulation using fingers.

- Anchor points are square and tan.
- Control points are round and red.
- The segment joining an anchor point to its control point is blue.
- The Bézier curve is black.
- Several visual feedbacks are displayed – assuming that the required symbols were defined at compilation time (see the source code for details) – in gray and light green.

Figure 47, Figure 48, Figure 49 and Figure 50 show the key interactions the user can have with the Bézier curve. An extra touch to the screen allows redrawing a new curve.

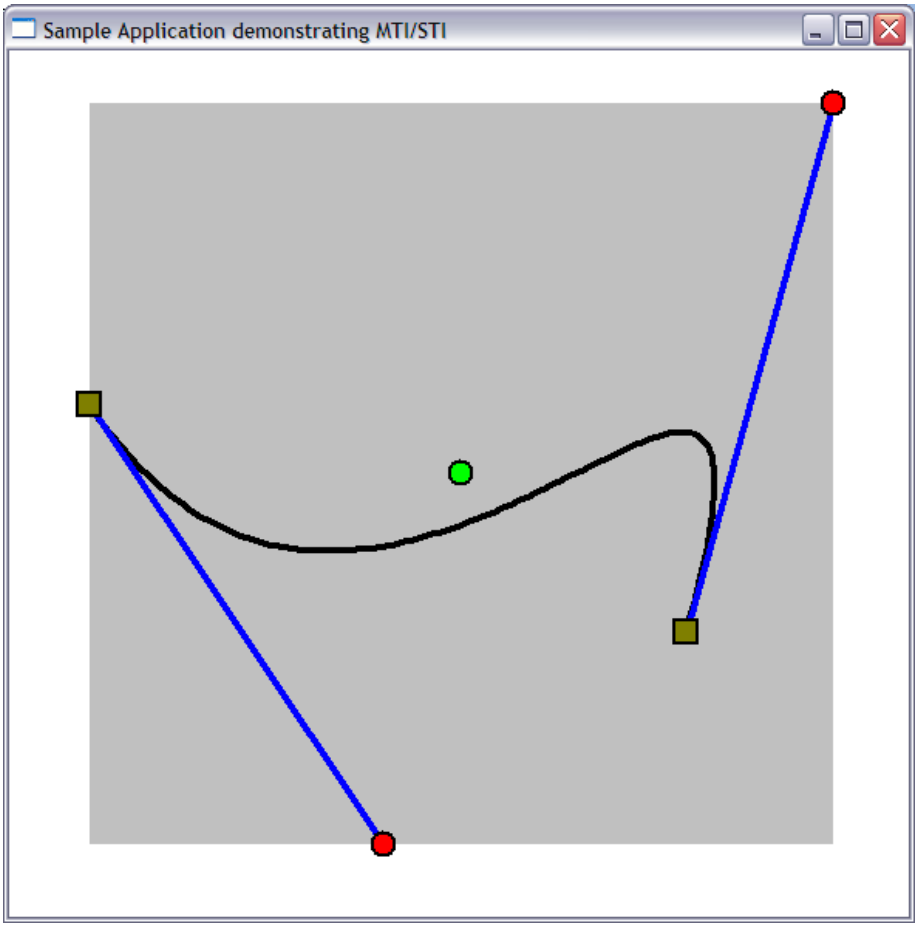

Figure 46: Typical display of the MTI sample.

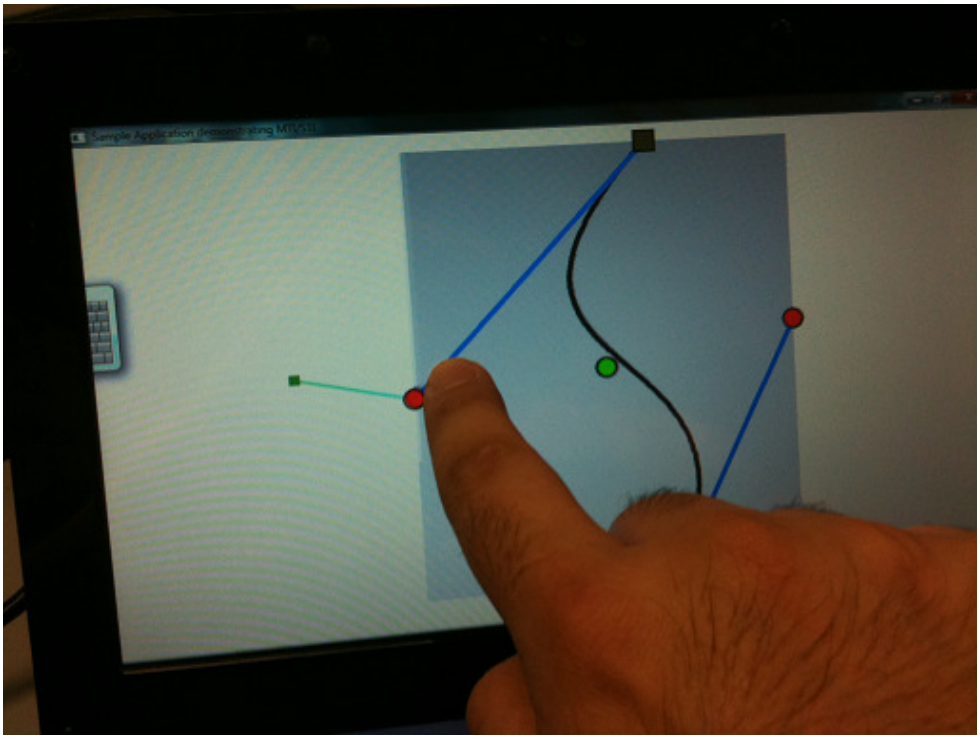

Figure 47: Move anchor and control points with one finger.

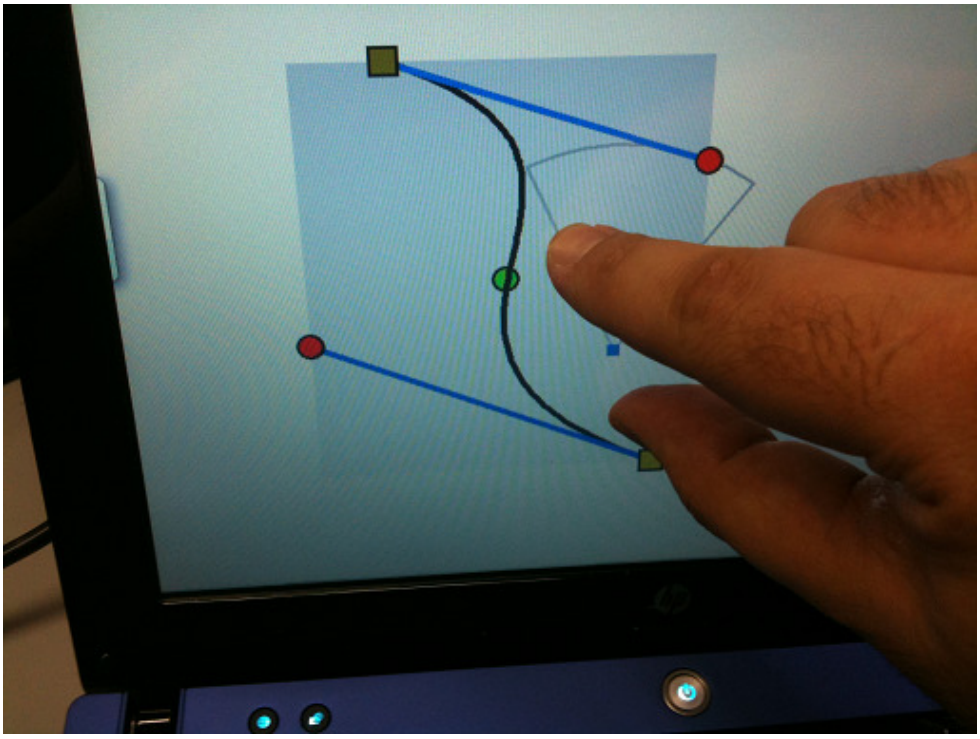

Figure 48: Rotate Bézier curve with two fingers.

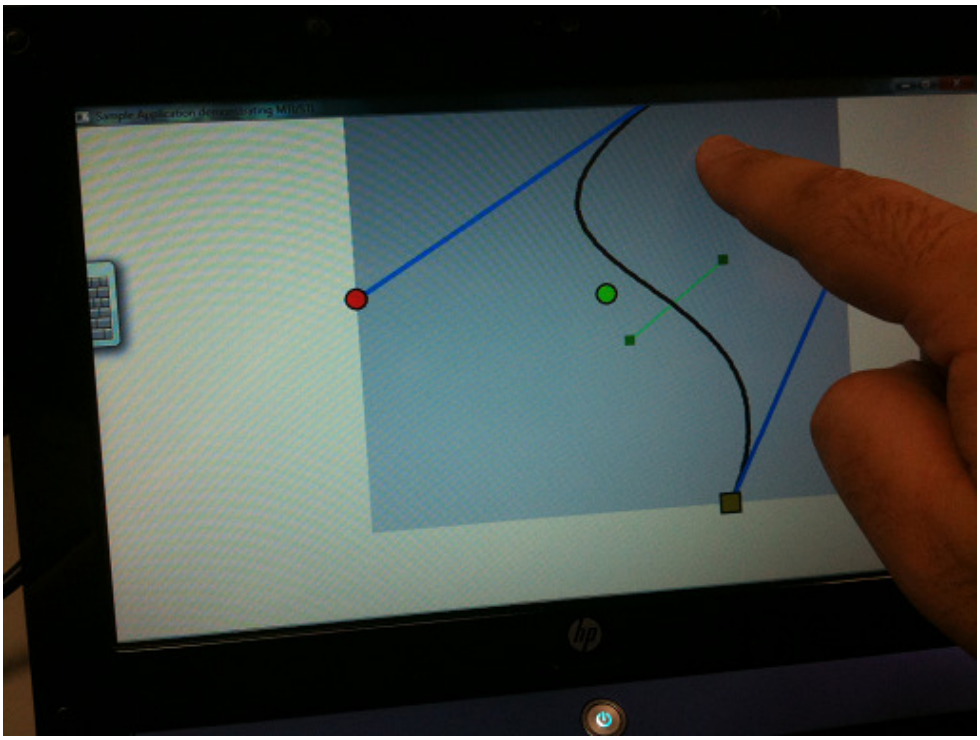

Figure 49: Move Bézier curve with one finger.

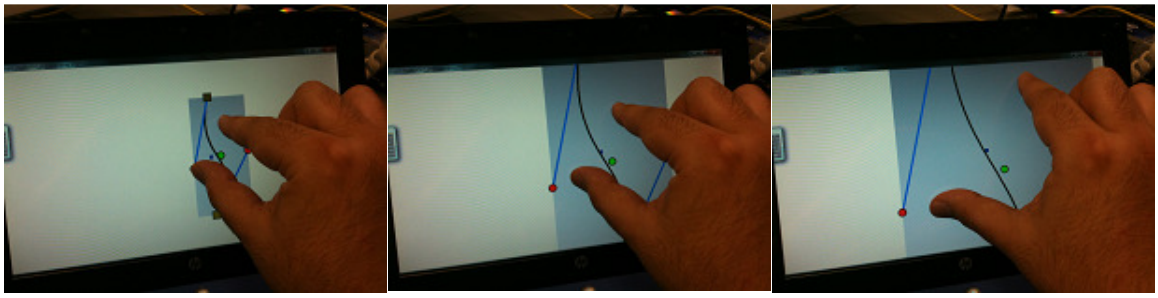

Figure 50: Scale Bézier curve with two fingers.

10.12.1 Build the Sample

A Microsoft Visual Studio 2005 project is shipped with the SDK for the MTI sample. The build process is as-usual. However, the Windows 7 SDK must be installed on the build system if the touch and sensor APIs are compiled. It may also require to update SDK path settings in the solution (see Figure 51 and Figure 52). In addition, to be able to execute the binary, the target system must run under Windows 7 (not Starter Edition) and needs support for a multi-touch capable touch screen.

Note that it is possible to activate and de-activate several key functions of the code, so a degraded version could be run on a standard laptop or desktop. However to benefit from the full user experience, a touch screen is recommended. For the sake of demonstration, running on a desktop is perfectly possible.

As mentioned earlier, this sample was built incrementally. A first basic version using the mouse and the keyboard was built. Several symbols were added to activate and de-activate various behaviors. Once this base application was finished, support for additional features were added, such as support for ambient light sensors and the instrumentation to report performance and energy efficiency (overall or per gesture) of the application.

Table 12 summarizes these symbols. More information is available in the source code as comment blocks.

Table 12: MTI sample symbols and associated functions

Symbols	Functionalities
__MTI_SAMPLE_FILE_LOG__	Enable file logging.
__MTI_SAMPLE_PER_GESTURE_FILE_LOG__	Enable file logging for each gesture.
__MTI_SAMPLE_NO_DISPLAY__	Disable application display.
__MTI_SAMPLE_DRAW_BOUNDING_BOX__	Display the Bézier curve's bounding box.
__MTI_SAMPLE_DRAW_BOUNDING_BOX_CENTER__	Display the Bézier curve's bounding box center.
__MTI_SAMPLE_DRAW_TRANSFORMATIONS_FEEDBACKS__	Display visual guids when applying a transformation to the Bézier curve.
__MTI_SAMPLE_INSTRUMENTED__ with __PL_WINDOWS__ __PL_GENERATE_INI__ __PL_GENERATE_INI_VERSION_TAGGING__ __PL_GENERATE_INI_BUILD_TAGGING__ __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__ __PL_BLOCKING_COUNTER_FILE_LOCK__ __PL_EXTRA_INPUT_CHECKS__	Activate instrumentation done using the Intel(r) Energy Checker SDK.
__MTI_SAMPLE_NON_AUTONOMOUS_METRICS_THREAD__	Activate message-loop timer driven metric thread activation. If not defined, thread is autonomous.
__MTI_SAMPLE_STI_TOUCH_CODE__	Activate the STI (Single Touch Interface) support code using touch messages.

__MTI_SAMPLE_MTI_CODE__	Activate the MTI (Multi Touch Interface) support code.
__MTI_SAMPLE_MTI_TOUCH_CODE__	Activate touch messages. Touch and gesture messages are mutually exclusive. By default, gesture messages are expected.
__MTI_SAMPLE_GUI_MONITOR__	Sample starts and stops a pl_gui_monitor instance.
__MTI_SAMPLE_POWER_AWARE__	Sample answers to system power suspend and resume messages.
__MTI_SAMPLE_SENSORS_CODE__	Activate sensor code.
__MTI_SAMPLE_LINEAR_COLOR_SCALE__	Activate linear color scaling based on ALS sensor readings. By default a logarithmic scale is used.

To run the sample on a system without multi-touch interface or sensors, such as a standard PC or laptop, build the code using the following symbols:

- __MTI_SAMPLE_PER_GESTURE_FILE_LOG__
- __MTI_SAMPLE_DRAW_BOUNDING_BOX__
- __MTI_SAMPLE_DRAW_BOUNDING_BOX_CENTER__
- __MTI_SAMPLE_DRAW_TRANSFORMATIONS_FEEDBACKS__
- __MTI_SAMPLE_INSTRUMENTED__
- __PL_WINDOWS__
- __PL_GENERATE_INI__
- __PL_GENERATE_INI_VERSION_TAGGING__
- __PL_GENERATE_INI_BUILD_TAGGING__
- __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__
- __PL_BLOCKING_COUNTER_FILE_LOCK__
- __PL_EXTRA_INPUT_CHECKS__
- __MTI_SAMPLE_NON_AUTONOMOUS_METRICS_THREAD__
- _CRT_SECURE_NO_DEPRECATE

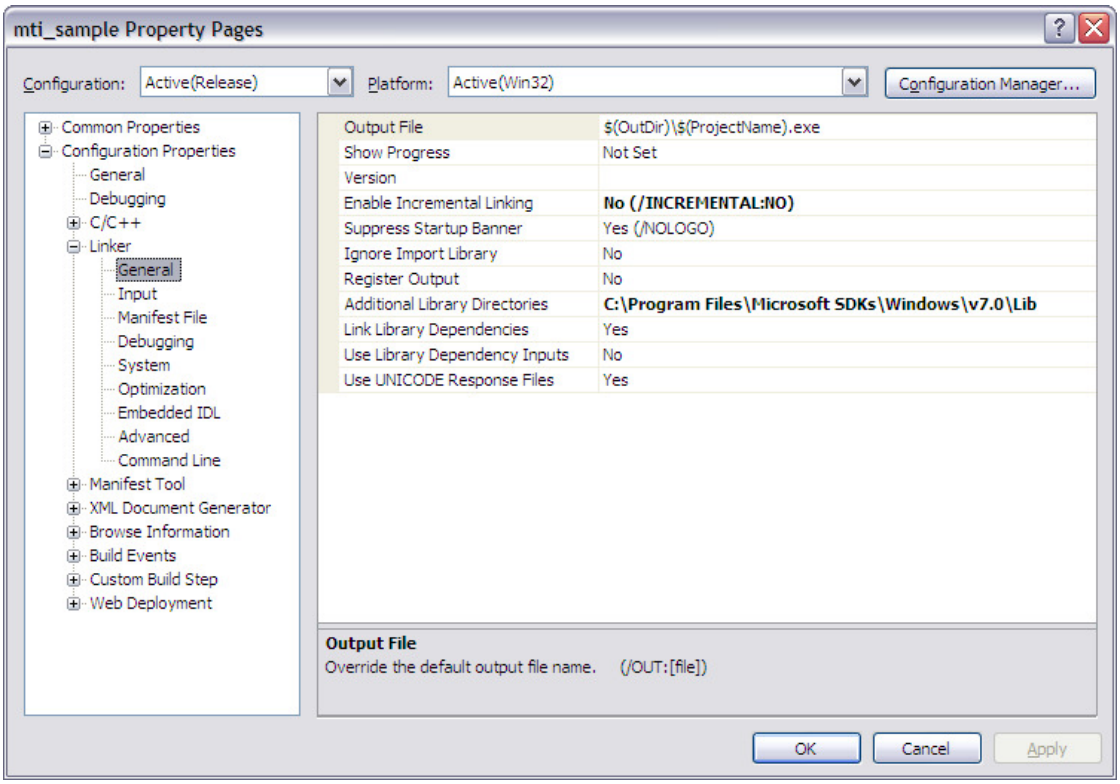

Figure 51: Update the library path adding the Windows 7 SDK libraries location.

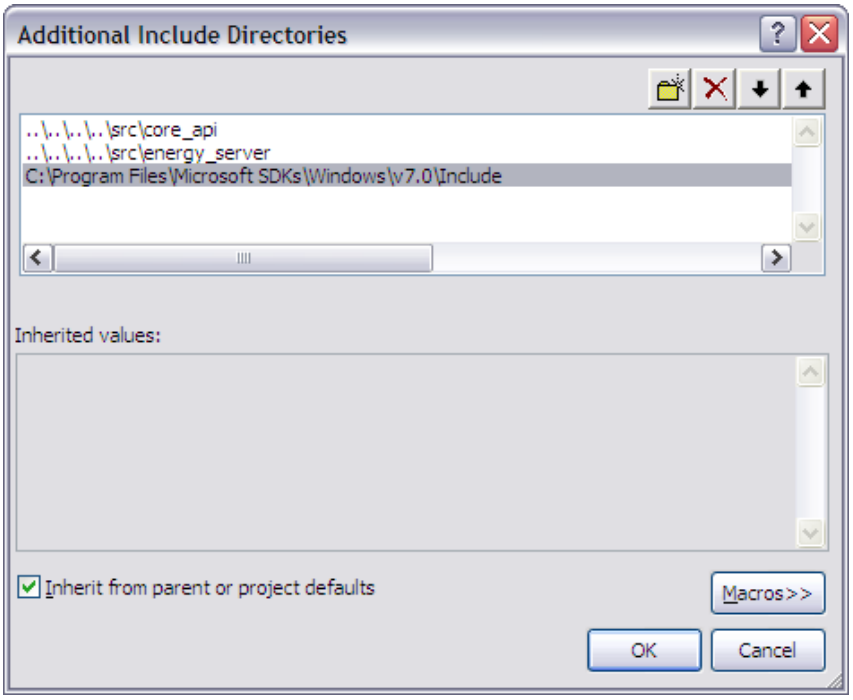

Figure 52: Update the include path adding the Windows 7 SDK headers location.

10.12.2 Sample Structure

In this section the focus is put on the SDK instrumentation only. Other ascpects of the sample can be studied using the comment blocks in the source code. The instrumentation allows the measurement of the execution time and the energy consumed during transformations applied to the Bézier curves. If the instrumentation is activated (__MTI_SAMPLE_INSTRUMENTED__), then a PL is created and the following counters are maintained:

- Bezier curves
- Bezier curve redraws
- Translations
- Scalings
- Rotations
- Sum of joules consumed for Bezier curves
- Sum of joules consumed for translations
- Sum of joules consumed for scalings
- Sum of joules consumed for rotations
- Sum of joules consumed for Bezier curves.decimals
- Sum of joules consumed for translations.decimals
- Sum of joules consumed for scalings.decimals
- Sum of joules consumed for rotations.decimals
- ms elapsed drawing Bezier curves
- ms elapsed performing translations
- ms elapsed performing scalings
- ms elapsed performing rotations
- Average Bezier curves per joule
- Average translations per joule
- Average scalings per joule
- Average rotations per joule
- Average Bezier curves per joule.decimals
- Average translations per joule.decimals
- Average scalings per joule.decimals
- Average rotations per joule.decimals
- Average joules per Bezier curve
- Average joules per translation
- Average joules per scaling
- Average joules per rotation
- Average joules per Bezier curve.decimals
- Average joules per translation.decimals
- Average joules per scaling.decimals

- Average joules per rotation.decimals
- Average ms per Bezier curve
- Average ms per translation
- Average ms per scale
- Average ms per rotation
- Average ms per Bezier curve.decimals
- Average ms per translation.decimals
- Average ms per scale.decimals
- Average ms per rotation.decimals
- Status [1 means running]

When the data logging is activated, then two different sets of data are captured. The first represents a running average of statistics for all the gestures. The second represents statistics for individual gestures. If the __MTI_SAMPLE_FILE_LOG__ symbol is defined, then the following data is logged:

- Time stamp
- Bézier curves
- Bézier curve redraws
- Translations
- Scalings
- Rotations
- Sum of joules consumed for Bézier curves
- Sum of joules consumed for translations
- Sum of joules consumed for scalings
- Sum of joules consumed for rotations
- ms elapsed drawing Bézier curves
- ms elapsed performing translations
- ms elapsed performing scalings
- ms elapsed performing rotations
- Average Bézier curves per joule
- Average translations per joule
- Average scalings per joule
- Average rotations per joule
- Average joules per Bézier curve
- Average joules per translation
- Average joules per scaling
- Average joules per rotation
- Average ms per Bézier curve

- Average ms per translation
- Average ms per scale
- Average ms per rotation

If the __MTI_SAMPLE_PER_GESTURE_FILE_LOG__ symbol is defined, the following data is logged:

- Time stamp
- Gesture
- Gesture type
- Gesture duration in ms
- Gesture energy in Joules

In this sample, the PL and the log management are off-loaded to a metric thread, as recommended in the threading sampleabove (metrics_thread_function). This metric thread computes all the relevant metrics and updates the log file and the PL counters. The metric thread can be either auto-triggered, or can be triggered by a timer created in the sample message loop. In the latter case, the activation of the metric thread will never have priority over the application.

To measure energy, and be energy-aware, the sample uses a function called get_esrv_pld. This function programmatically manages ESRV. The function first searches for an ESREV_GUID environment variable. It is assumed that if an instance of ESRV was started manually, this variable was defined and was set to the ESRV UUID. Such a mechanism allows other applications to attach to the ESRV PL.

Of course, this is just one example on how the UUID can be exchanged between applications. If no such environment variable is found, then the function starts its own instance of ESRV. This requires that the ESRV and required support modules – if any – are in the path.

Note also that if the sample is built in debug mode, then the simulated device is used, so no physical analyzer is required during debugging.

At this stage, the application is energy-aware. It can compute its energy efficiency metrics and generate gesture performance and energy statistics.

The main thread runs the application core, which is the Windows message loop. Each time a message is received that is associated with a gesture, for example, the processing code reads the ESRV PL counters at the beginning of the processing and at the end. Execution time is similarly measured in parallel.

It is important to notice that the best results are obtained if the power measurement device has hardware integration of power to measure energy.

The other point to note is that if the time elapsed between the beginning and the ending of a gesture is too short, the energy counter’s value may be constant. If that happens, then the sample uses the constant value and multiplies it by the elapsed time.

Figure 53, Figure 54 and Figure 55 are example graphs generated using a spreadsheet application to process the log files generated by the application.

NOTE

The log file is named mti_sample_log .csv. It is a comma-separated ASCII file that can be imported in a spreadsheet application for post processing. Note that the sample may fail to run if a previously generated log file is still open by another application. Close the file prior to running the sample again, so it can delete it.

NOTE

When started by the MTI sample, the ESRV PL is removed at the application end. In addition to this clean up, the program asks if it should remove its own PL.

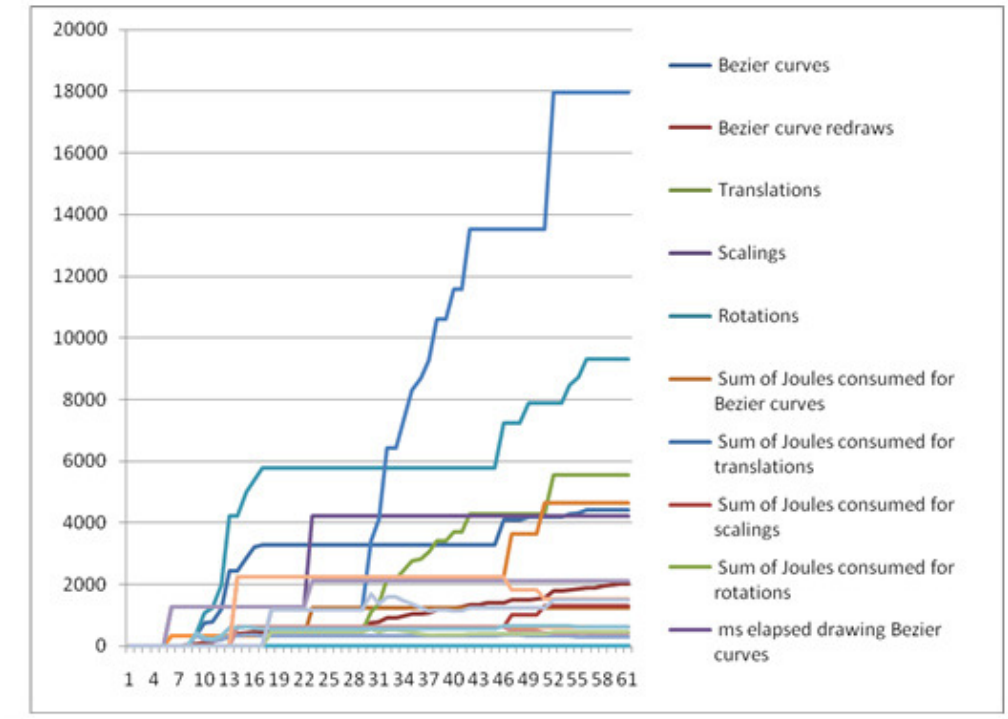

Figure 53: Running averages and counts using log data.

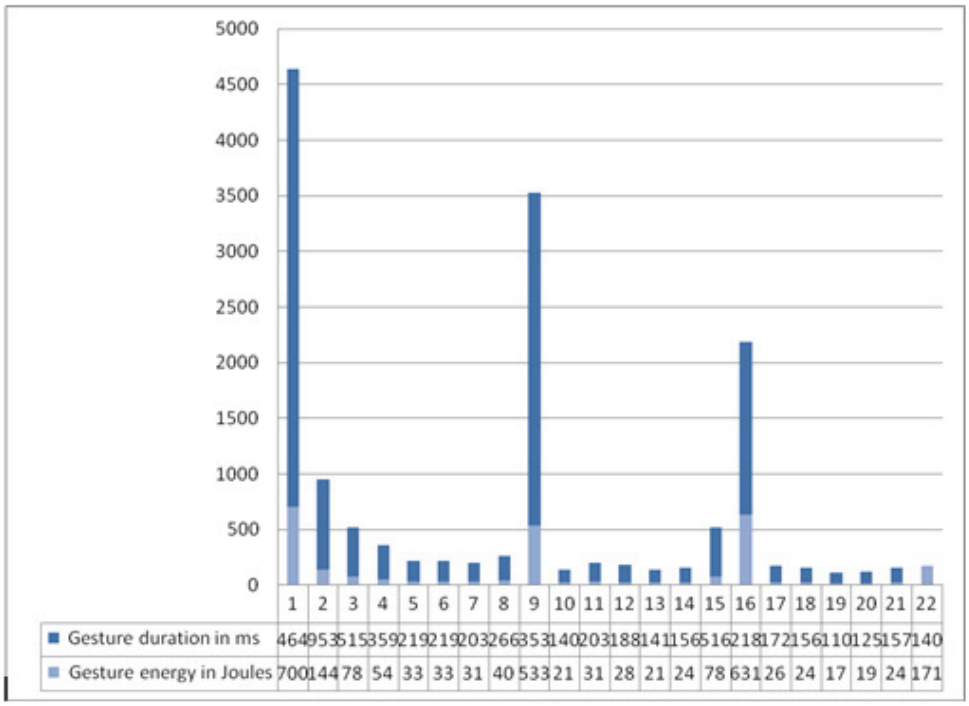

Figure 54: Per gesture time and energy using per gesture log data.

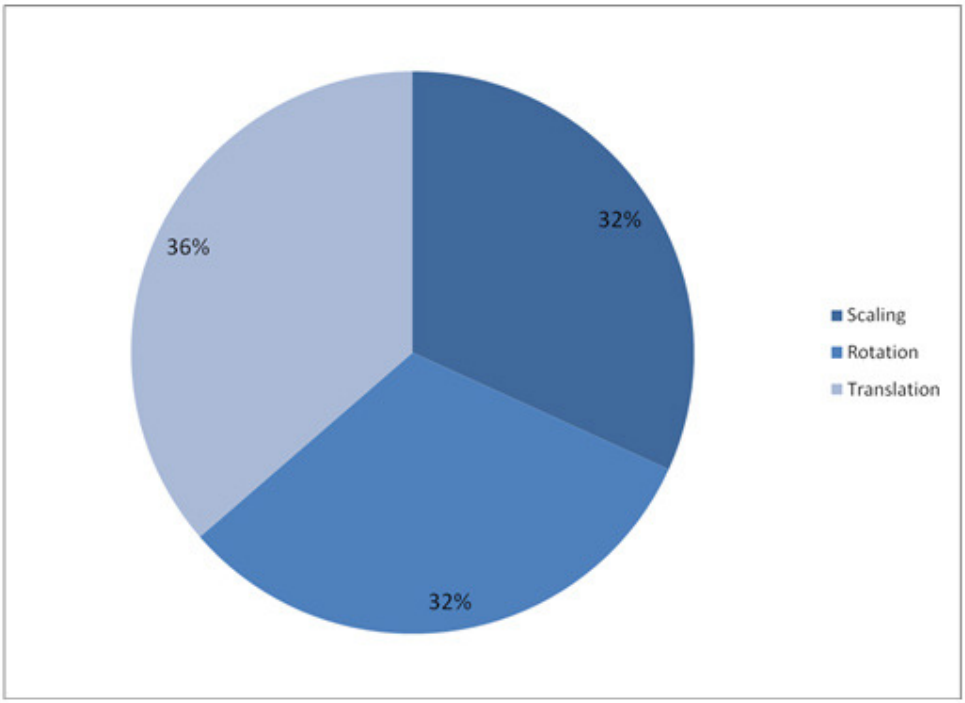

Figure 55: Gesture type split using per gesture log data.

10.13 Other Code Samples

The Intel Energy Checker SDK provides several additional sources of sample code to help facilitate the use and understanding of the API. Table 13 gives each additional sample and its location in the SDK tree.

Table 13: Other samples and SDK locations

Sample name	Sample location
consumer	`iecsdk\src\core_api\porting_test_applets\consumer`
producer	`iecsdk\src\core_api\porting_test_applets\producer`
logger	`iecsdk\src\core_api\porting_test_applets\logger`
core_api_unit_tests	`iecsdk\src\core_api\unit_tests\core_api_unit_tests`
pl_csv_logger	`iecsdk\src\companion_applications\pl_csv_logger`
benchmark	`iecsdk\src\core_api\benchmark`
helper_api	`iecsdk\src\helper_api`
energy_meter_driver	`iecsdk\utils\device_driver_kit\src\energy_meter_driver`
temperature_meter_driver	`iecsdk\utils\device_driver_kit\src\temperature_meter_driver`

11 Proofs of Concept

This chapter presents two proofs of concept using the SDK and several Intel hardware and software technologies to save energy through software. The objective of this chapter is to show examples of what an ISV (Independent Software Vendor) or an OEM (Original Equipment Manufacturer) can do to save energy without sacrificing performance, by leveraging intimate software knowledge and Intel® technologies.

11.1 Memory-Aware Energy Savings

The SDK is used to instrument a PostgreSQL (PostgreSQL, 2010) database and the TPCC-UVa (Llanos, 2010) benchmark. This database server and transactional workload are used in two case studies presented in this section. Similar instrumentation can be performed on other databases.

One recommendation the SDK documentation suggests to developers, is to limit the impact of the instrumentation to the existing source code. This can often be achieved by adding a dedicated *metric thread* to the application (Figure 56). This metric thread is then responsible for collecting the useful work data from the worker thread(s), importing the energy readings from the right instance of ESRV, and for computing and exporting the energy efficiency metrics.

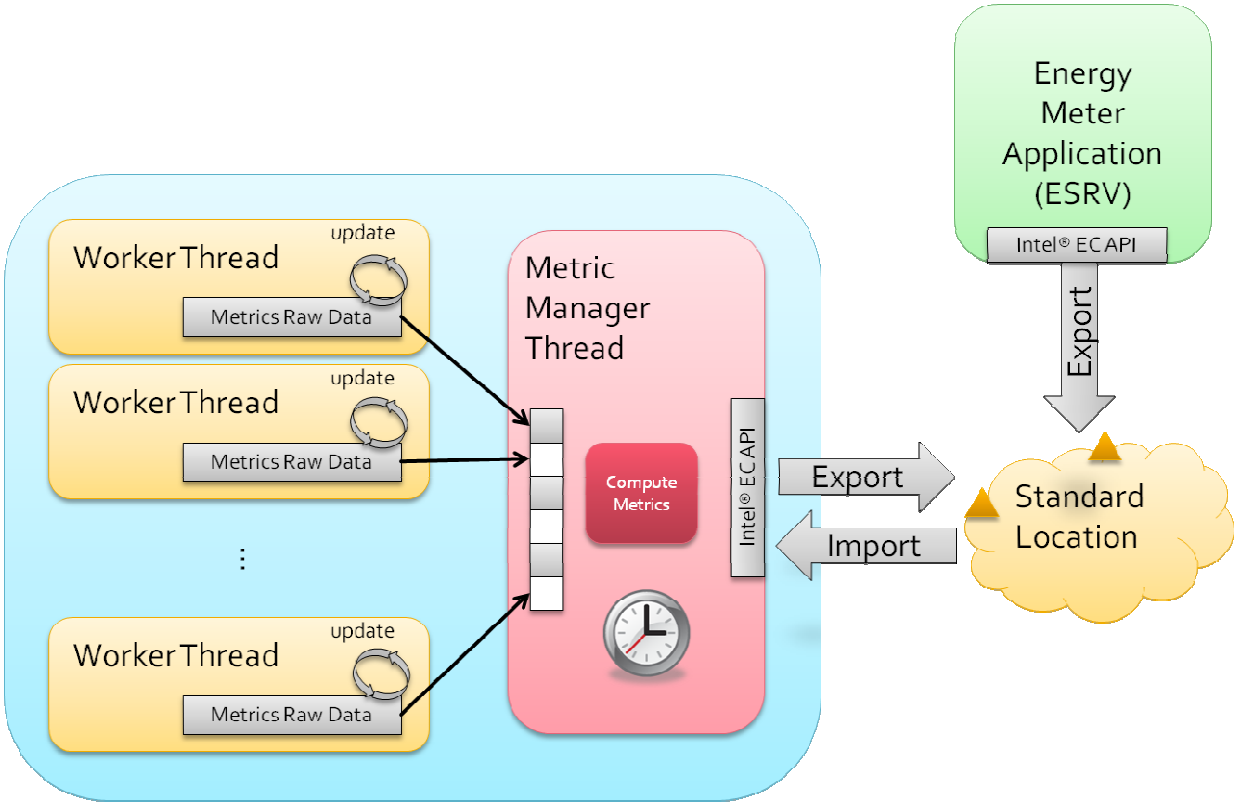

Figure 56: Dedicated metric thread limits the instrumentation impact in an existing code.

With a database engine, such as PostgreSQL, you can completely avoid touching the software's source code and simply rely on the database's existing statistics-gathering mechanism to extract the amount of useful work done. For PostgreSQL, useful work is simply defined as the number of SQL statements executed. Thus, we can devise a set of dedicated processes – called *loggers* – which communicate with the database back-end, using the PostgreSQL native interface (version 8.4.4).

A similar approach is taken when it comes to instrumenting the workload, which is a transactional benchmark in this case (TPCC-UVa). Logs generated on-the-fly by the front-end, are parsed by a logger, and the extracted data is used to compute and expose energy efficiency metrics and benchmark results using the SDK API.

Figure 57 represents the basic data exchange flows between the data base and the main logger.

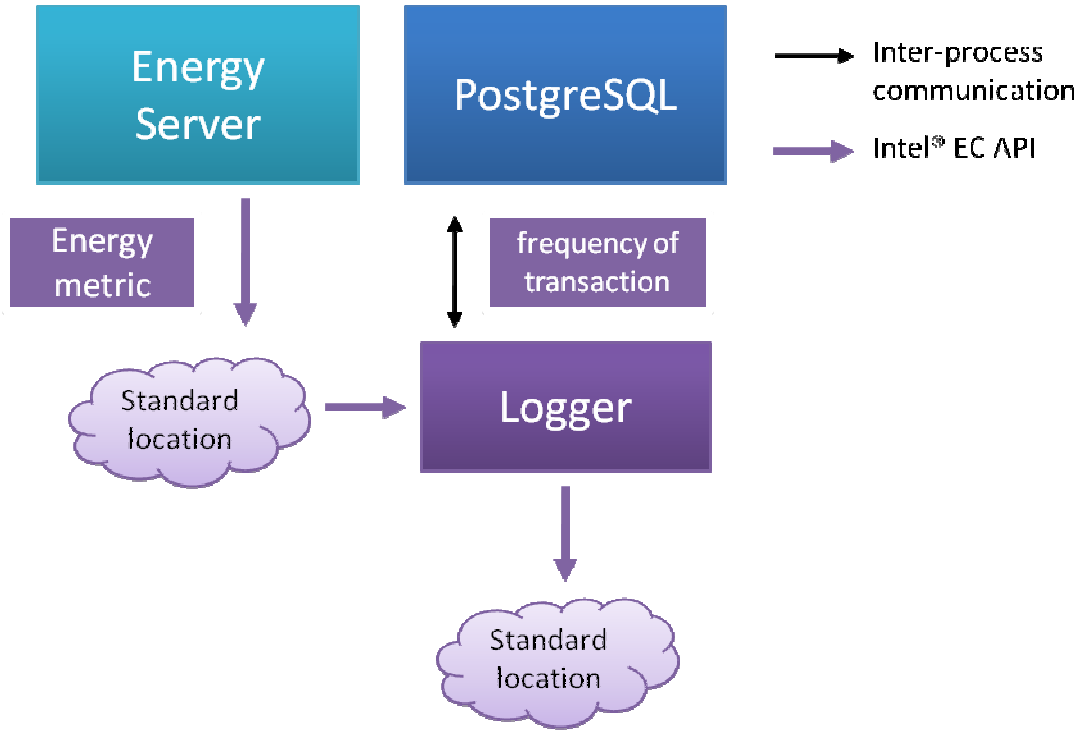

Figure 57: Data exchanges between the back-end and the main logger.

Figure 58 illustrates in more detail the overall experimental setup architecture and the added instrumentations.

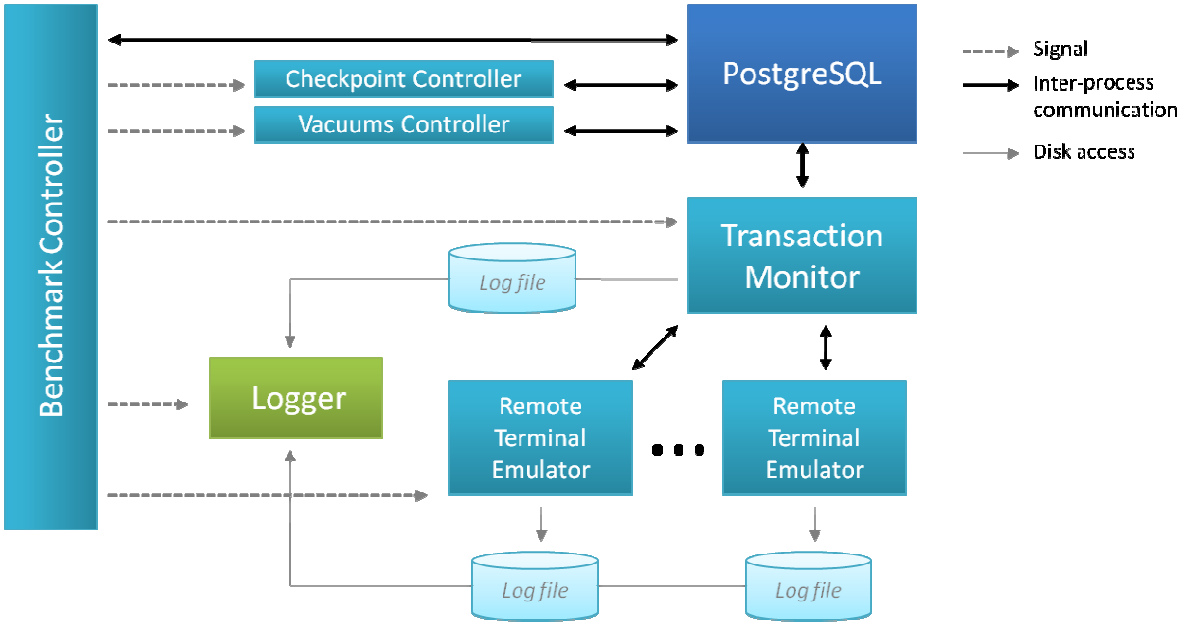

Figure 58: PostgreSQL and TPCC-UVa overall architecture.

Figure 59 also illustrates the setup and instrumentation. In addition to the dynamic logger, a static logger is added for run validation, so it can report final TPCC-UVa results and statistics in a dedicated PL.

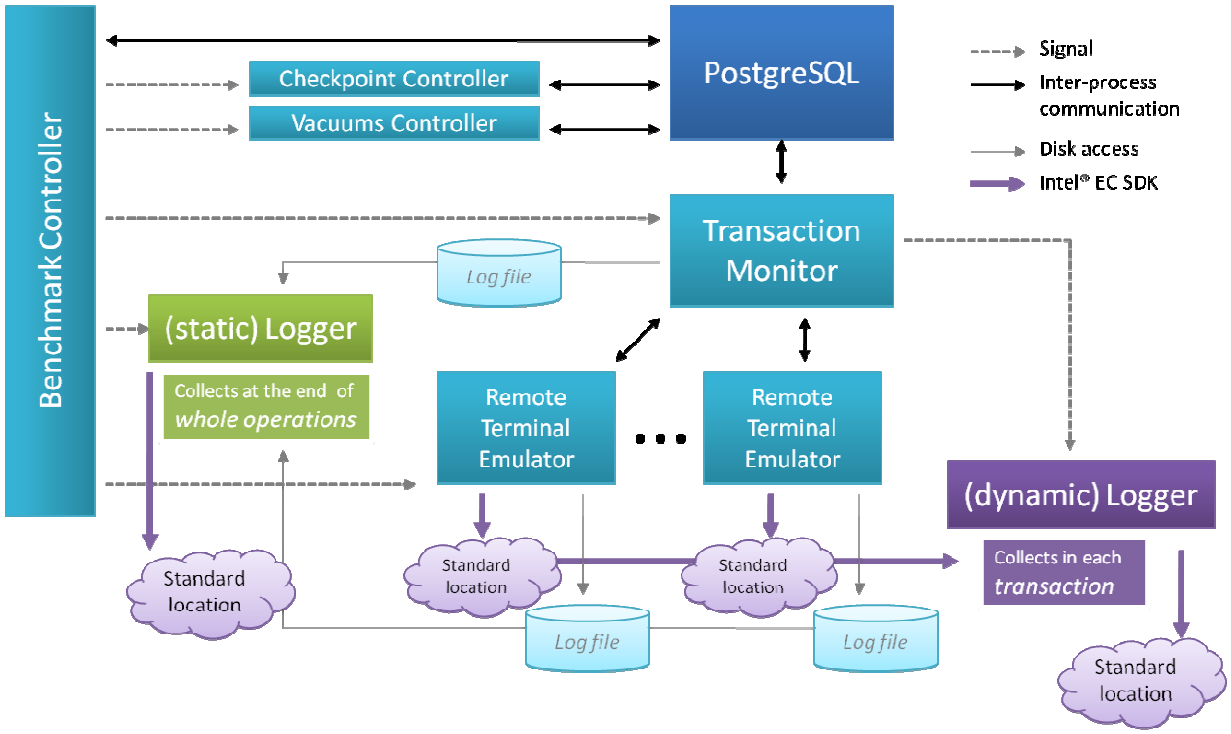

Figure 59: Overall architecture with instrumentation.

Each PL generated by an application that is instrumented with the SDK has a configuration file (pl_config.ini). This ASCII file contains the description of the exposed counters with various additional data. The listing below shows the content of the dynamic logger's PL. The energy efficiency counters are highlighted in bold. Readers interested in the meaning and the function of other elements present in the PL configuration file can refer to the SDK user guides.

```
1  iec_pgsql
2  4b5e9aa1-5733-42e8-8380-d29d52299776
3  /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/
4  26
5  /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Total Number of
6  Transactions (PGSQL)
7  /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Total Number of
8  Transactions in recent 5 seconds (PGSQL)
9  /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Heap Allocations
10 in Bytes (PGSQL)
11 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Heap Allocations
12 in Bytes (PGSQL) in recent 5 seconds
13 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Measurement Time
14 in second
15 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Measurement Time
16 in second.decimals
17 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/System Energy
18 Consumption in joule
19 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/System Energy
20 Consumption in joule.decimals
21 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/System Energy
22 Consumption in Wh
23 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/System Energy
24 Consumption in Wh.decimals
25 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/System Energy
26 Consumption in joule in recent 5 seconds
27 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/System Energy
28 Consumption in joule in recent 5 seconds.decimals
29 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/System Energy
30 Consumption in Wh in recent 5 seconds
31 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/System Energy
32 Consumption in Wh in recent 5 seconds.decimals
33 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
34 Transactions per joule
35 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
36 Transactions per joule.decimals
37 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
38 Transactions per Wh
39 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
40 Transactions per Wh.decimals
41 /opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
42 Transactions per seconds (PGSQL)
```

```
/opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
Transactions per seconds.decimals (PGSQL)
/opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
Transactions per joule in recent 5 seconds
/opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
Transactions per joule in recent 5 seconds.decimals
/opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
Transactions per Wh in recent 5 seconds
/opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
Transactions per Wh in recent 5 seconds.decimals
/opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
Transactions per seconds in recent 5 seconds(PGSQL)
/opt/productivity_link/iec_pgsql_4b5e9aa1-5733-42e8-8380-d29d52299776/Number of
Transactions per seconds in recent 5 seconds.decimals (PGSQL)
2010.01.18(L)
__PL_LINUX__
__PL_GENERATE_INI__
__PL_GENERATE_INI_VERSION_TAGGING__
__PL_GENERATE_INI_BUILD_TAGGING__
__PL_GENERATE_INI_DATE_AND_TIME_TAGGING__
__PL_BLOCKING_COUNTER_FILE_LOCK__
PL created on Fri Sep 17 06:19:35 2010
```

With these energy efficiency counters, it is now possible to add innovative energy saving heuristics into PostgreSQL so it can save energy without compromising performance or sacrificing functionality.

11.2 Memory-Usage-Aware Energy Saving Heuristic

While the SDK can be used under many operating systems and virtually any architecture, the energy saving heuristics are likely to be very system specific. Indeed, to save energy, software can easily leverage its intimate knowledge of the work it does and the data it is processing. But it still requires means – often provided by the hardware – to act upon the host systems' power draw. These hardware features are called *actuators*.

In many cases, power-saving hardware feature logics cannot make any sense of what is happening in an application and must therefore rely on measurements to search for power-saving opportunities. Whatever the hardware detects through its measurements, the software action at the origin has already elapsed.

Luckily, the software has this essential knowledge right when it happens and in some cases it can project its future actions using some form of modeling. Memory utilization is one of these key knowledge points that the application knows best.

11.2.1 Memory Energy Saving Actuator

The Intel® Nehalem EX platform offers a feature called dynamic memory power management (code named Monroe technology). Systems having this technology

can manage their physical DIMMs' power draw. In the short, the BIOS and the hardware allow the positioning of the DIMMs' power draw to various levels, including an unpowered state. Even so, there are two dawbacks when this feature is used.

The first drawback is the performance hit taken when a DIMM in low-power state is first accessed. Indeed, the hardware needs to power-up the DIMM to its operational power state. This injects a slight deterministic delay. In our transactional workload, this delay is insignificant; it doesn't translate in performance degradation.

The second drawback is related to the abstraction introduced by the virtual address space visible to user space processes. Undeniably, this requires either a means for the application to map virtual addresses to physical addresses – using, for example, a page table walk mechanism in a kernel module / driver –, or to amend the operating system's page allocator to linearly allocate memory pages in the physical DIMMs associated to a NUMA (Non-Uniform Memory architecture) node. In other words, this means that for a given NUMA node, the amended operating system linearly allocates consecutive memory in the physical address space. The second path was taken in this example, and the Linux* kernel was modified to expose such behavior (kernel version 2.6.28.8).

11.2.2 Energy Saving Heuristic

Since in PostgreSQL it is possible to monitor the data base memory usage, and since the amount of memory buffers maintained by the software to store table data is proportional to the data base size, a simple energy saving heuristic can be implemented.

Two key metrics (`buffer_hit` and `buffer_usage`) were defined for this purpose. The first metric, similar to the processor cache hit/miss ratio, indicates the percentage of successful look-ups in the buffers for requested data (the higher, the better). The second reflects the use of the buffer pools (the higher, the better). The API is used to dynamically expose these counters, so they can be monitored and used as inputs to the energy saving heuristic. Figure 9.8 shows the associated PL's configuration file. Note that this instrumentation required modifying several source files of PostgreSQL.

```
1  pgsql
2  1124f161-773f-4da9-abfe-fa02688e19ab
3  /opt/productivity_link/pgsql_1124f161-773f-4da9-abfe-fa02688e19ab/
4  3
5  /opt/productivity_link/pgsql_1124f161-773f-4da9-abfe-fa02688e19ab/buffer_hit
6  /opt/productivity_link/pgsql_1124f161-773f-4da9-abfe-fa02688e19ab/buffer_usage
7  /opt/productivity_link/pgsql_1124f161-773f-4da9-abfe-fa02688e19ab/buffer_state
8  2010.01.18(L)
9  __PL_LINUX__
10 __PL_GENERATE_INI__
11 __PL_GENERATE_INI_VERSION_TAGGING__
12 __PL_GENERATE_INI_BUILD_TAGGING__
13 __PL_GENERATE_INI_DATE_AND_TIME_TAGGING__
14 __PL_BLOCKING_COUNTER_FILE_LOCK__
15 PL created on Fri Sep 17 06:18:00 2010
```

By monitoring these two counters and by varying the data set size while running the workload, we can identify several interesting domains and thresholds valued for the counters. In particular, it allows us to note that the buffer hit percentage can be reliably used to detect when the database back-end is expanding its buffers pool.

Two memory use cases were defined (*lightweight 0* & *1* and *heavyweight*). Figure 60 and Figure 61 show the evolution over time of both memory counters, while running the TPCC-UVa workload and varying the data set size. In addition to revealing a few unstable transitional intervals that must be provisioned for in the final energy-saving heuristic, Figure 60 shows how the buffer hit percentage metric can be used to detect the increase of the buffer pool size when transitioning from a lightweight data set to a heavyweight data set. It also shows that this same metric cannot be used to detect a situation where the buffer pool size could / should be reduced so the associated DIMMs could be powered off.

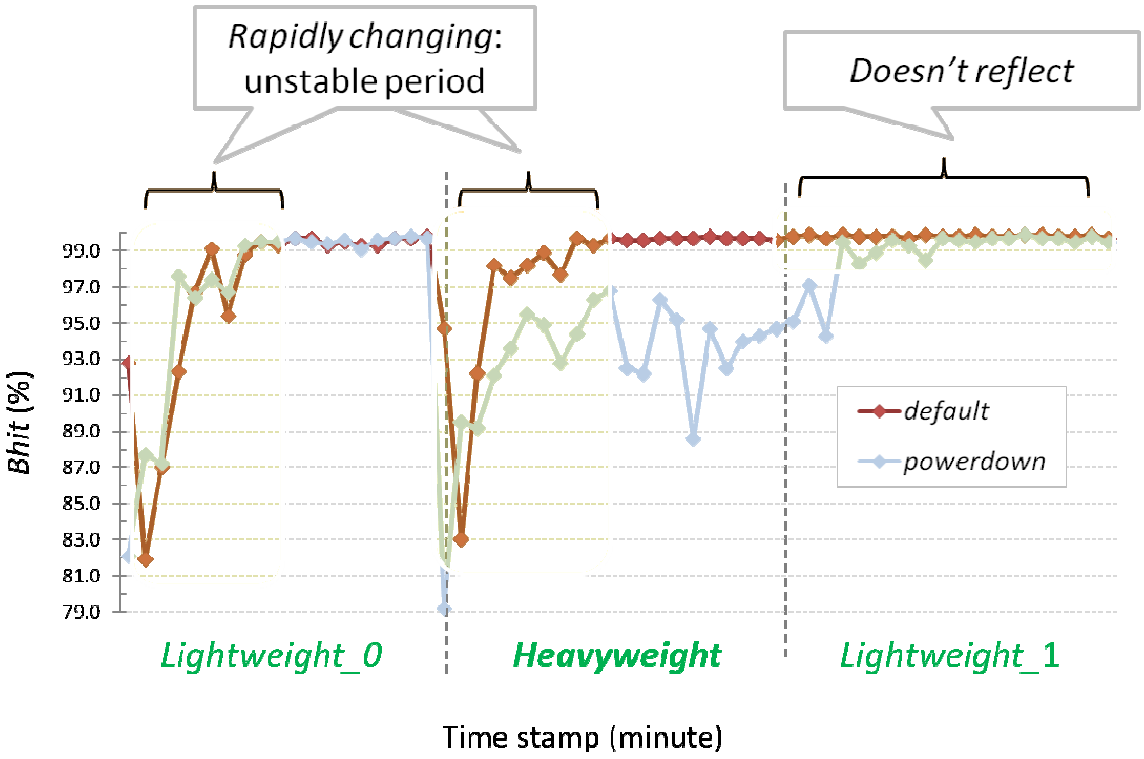

Figure 60: Evolutions over time of the buffer hit percentage.

This is where the buffer utilization metric fills the gap as shown in Figure 61. Indeed, this metric can inform a situation where the buffer pool is not fully used anymore, and unused memory could be freed and powered down.

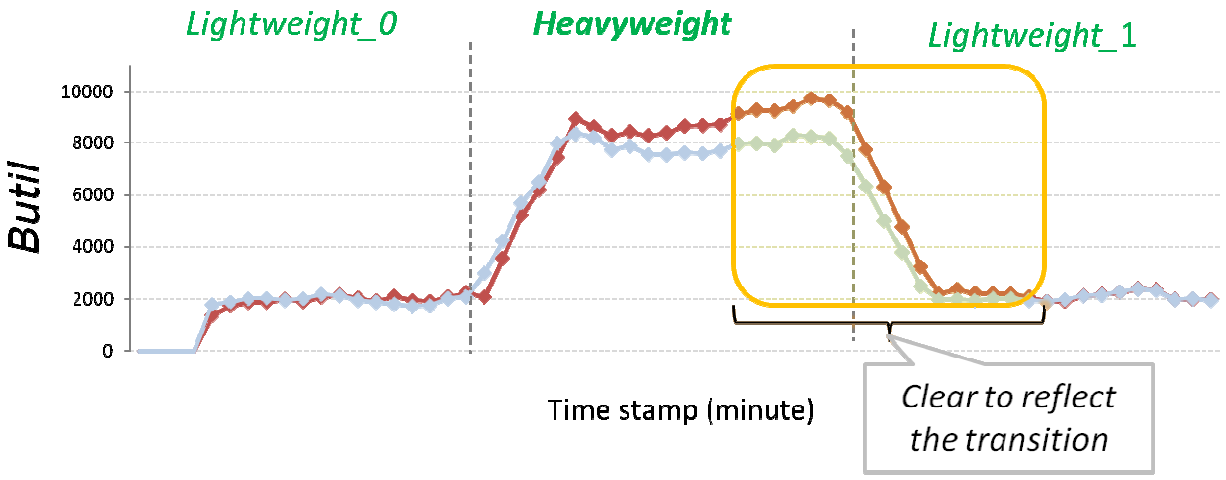

Figure 61: Evolutions over time of the buffer utilization.

Since the version of PostgreSQL used in this study is capable of dynamically expanding its buffers pool, but cannot shrink it, we amended its buffer management routines to add this required feature. This work was done by modifying a limited number of source code files and at a very reasonable development cost.

The knowledge gained by this study, and the two memory utilization metrics, associated with the new buffer pool shrinking feature, allow the definition of a third counter (`buffer_state`). This counter is used to provide a hint from the software to signal the memory domains not used and can therefore be powered down. Note that the control of the actuator (the physical DIMMs power level) is not performed directly from the data base. Though it could be done autonomously, especially in the scope of an enterprise application likely running on dedicated nodes, we decided to present an indirect management path. This way, a management middle-ware – or even the operating system if it was aware of this – could use this counter to marshal the memory power draw setting. Multiple instances of the same application or different memory-aware applications could harmoniously share the same platform.

Figure 62 shows the state machine implemented to drive the memory-usage-based energy saving heuristic, using the two memory metrics defined earlier.

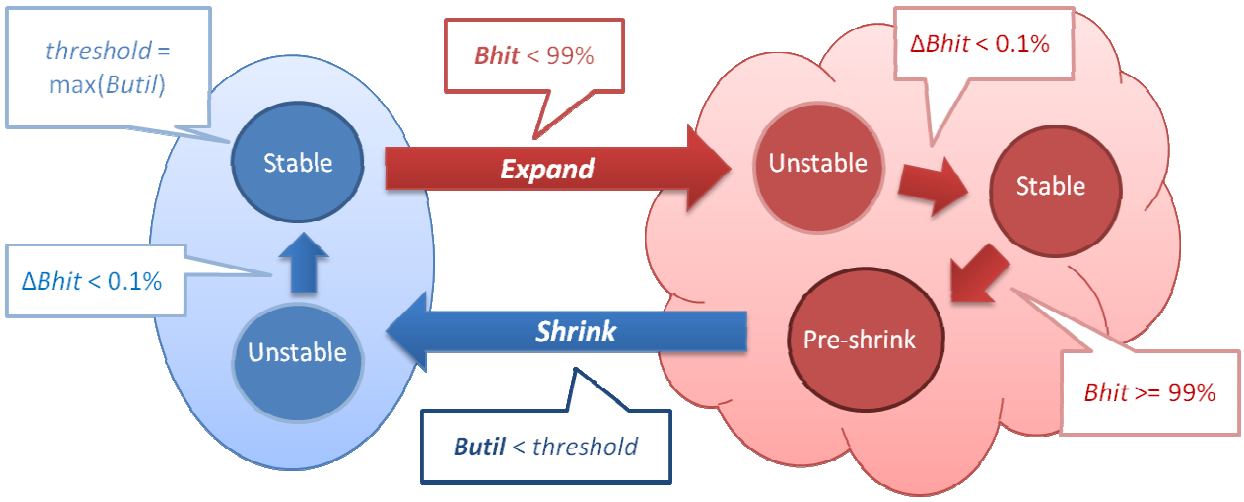

Figure 62: State machine used to drive the energy saving heuristic.

11.2.3 Experimental Results

To measure the energy savings, we ran the TPCC-UVa benchmark with two data set sizes in two different cases. The first case was not using the memory power management feature of the Nehalem EX platform (*default* case). The second case used the memory module power manager developed in the scope of this experiment (*adaptive* case).

Both cases ran on the same hardware with the same version of the operating system. The data set sizes where selected so they guaranty two memory configurations, where one fits in one half of a memory module (*lightweight*) and the other does not (*heavyweight*). The expansion of the data set size was done following the TPCC-UVa rules, adding warehouses and terminals to the problem rather than increasing the size of the data handled by each warehouse. Data collected during unsuccessful runs were rejected.

Since the performance between the lightweight and heavyweight test cases is naturally different, the average response time is used to measure the system's performance. In addition to being independent from the data set size, the average response time is a good metric because it directly influences the user experience. In the next sections, we first describe the experimental setup with detailed configurations and settings, and then present the experimental results.

11.2.3.1. Experimental Settings

A high-end server (Intel® Software Development Platform) was used to run the experiments. Table 14 summarizes the SDP's key features and characteristics. Figure 63 and Figure 64 show the memory architecture of the server and depicts how memory modules are physically implemented. Each memory module's power draw can be controlled independently by using the power management technology of the Nehalem EX platform.

A Yokogawa* WT210 Power Analyzer was used to measure the server's energy consumption. An instance of ESRV is started at the beginning of the test.

PostgreSQL is started and the TPCC-UVa benchmark is run, implementing the lightweight, heavyweight, and lightweight cases in that order, without interruption. The database and the workload energy efficiency metrics are computed and exposed, as described earlier in this section, by the loggers. In addition, the buffer utilization and buffer hit percentage metrics are computed and exposed along with the requested memory module power settings as deduced from the data base back-end's memory utilization. The memory power management module – a simple independent process importing the memory counters – uses the state machine to dynamically set the server's memory modules' power state. Note that the lowest non-zero power level was used in these experiments.

Figure 63: Eight memory modules are installed in the test server.

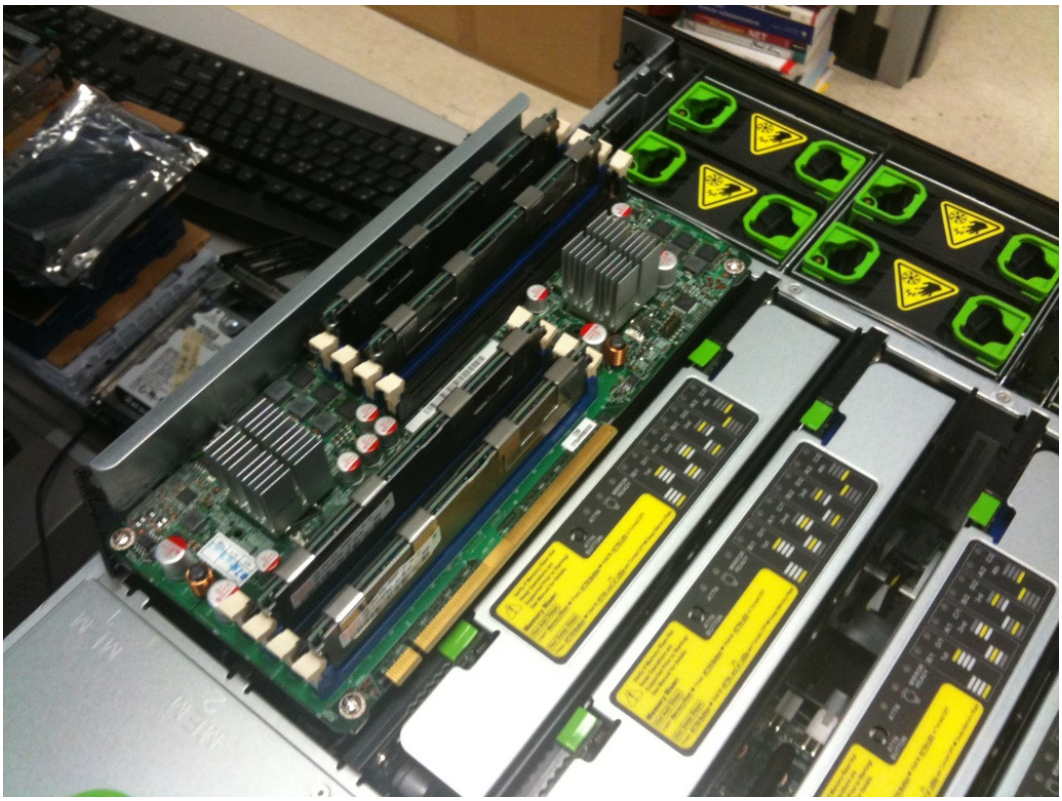

Figure 64: Each memory module can hold up to eight DIMMs.

Table 14: Test system configuration details

Subsystem	Configuration
Processor & chipset	Model/Speed/Cache: X7560 – Q3X4– 2.26GHz 64 cores/4 chips/16 cores per chip 4 sockets Chipset: Emerald Ridge Gold System bus: QPI – 6.4GT/s
Platform	Intel Software Development Platform Chassis: Boxboro – 4U rack mount Baseboard: Quanta QSSC-S4R Board revision: "Silver" (equivalent to "qual") BIOS: BIOS 26, BMC 17, FRU 10, HSC 2.14, ME 1.83 DIMM slots: 64 (8 on each of 8 memory risers) PCI slots: 9 – PCIe; 1 PCI Drive controller: Intel SRCSASBB8I RAID controller Power supply: 2x – 850W – Delta DPS-850FB A Rev S3F NIC: 2x Intel 82576 Gigabit Dual Port
Memory	Memory Size: (nGB ((n x XGB)) 128GB – 32 x 4GB Brand/model: Hynix / HMT151R7BFR4C-G7 DIMMs: 4GB – 2Rx8 – PC3 – 8500R
Mass storage	Spindles: 2ea 146GB SAS

11.2.3.2. Experimental Results

Figure 9.14 shows the experimental results of response time reported by TPCC-UVa during the running of the benchmark for both the baseline and memory utilization-aware power saving. Table 15 shows the energy savings data.

From the experimental results, we see that comparing with no memory power saving in the operating system, more than 8 percent energy reduction is achieved with memory-utilization-aware energy saving, while the average response time of the benchmark gets little impact.

The experiment demonstrates that comparing with no insight and support on the application's memory utilization in the operating system, additional energy can be saved with little performance impact by incorporating application-level memory utilization feedback into power management software.

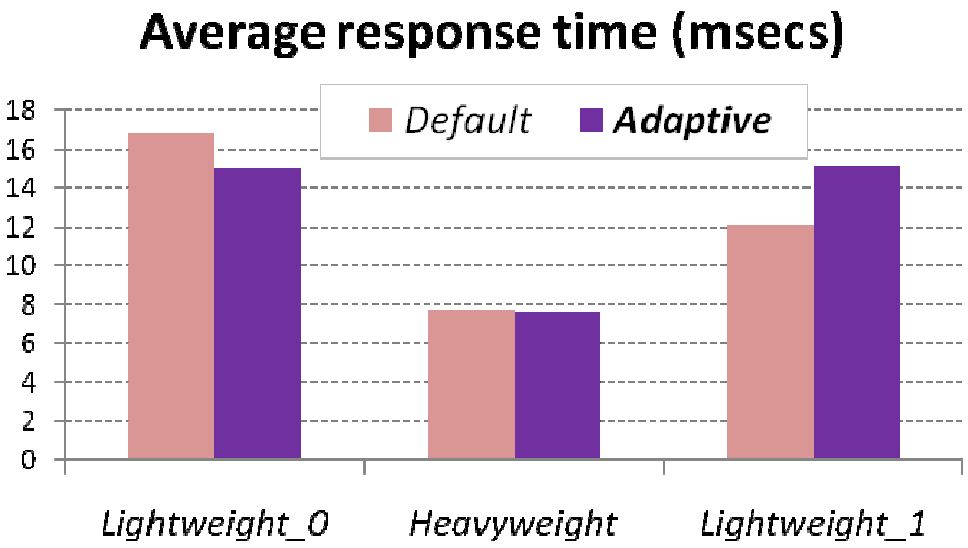

Figure 65: response time between default and the adaptive test cases.

Table 15: Results of energy saving on TPCC-UVa (default vs. adaptive)

Data Size	Energy Savings
Lightweight_x	+8.25%
Heavyweight	+4.12%

11.3 Performance-Aware Energy Savings

In a typical online transaction processing configuration, a power-saving application is built. The application monitors the performance counters of transaction processing from both the frontend and the backend, and checks whether the performance gets impacted or not. The information is provided to the middleware component, which learns and updates the optimal power limiting policy on the system. At the lowest level, a firmware component provides the basic building block to cap the system's power for energy reduction, acting as the policy executer.

The experimental results demonstrate that significant energy reduction is achieved without performance impact.

11.3.1 Performance-Aware Power Saving

Various power-saving policies are commonly provided by the power management module in operating systems. For example, several CPU utilization counters are considered by the power management module in Windows when the balanced power plan is chosen. When CPU utilization is low, the power management module regulates the CPU to a P-state with lower performance for reduction of power consumption. When CPU utilization is high, the power management module regulates it to a P-state with higher performance, to avoid the impact of user or application experiences. Although such an approach provides generic power saving functionality, which is widely applicable to a large variety of scenarios, the room for energy reduction is not fully exploited. This is because the power management module does not gain the visibility of application performance.

Recently there is growing interest in performance-aware power saving, where intelligent power controllers take into consideration application-level performance indicators.

Figure 66 depicts the high-level system diagram of a typical setting of performance-aware power saving. The job scheduler is responsible for scheduling workloads to computing systems. It acts as the performance manager to monitor the application-level performance. The information is provided to the power controller, with which the intelligent power policy is computed. The power policy is then executed to reduce the power consumption of the computing systems.

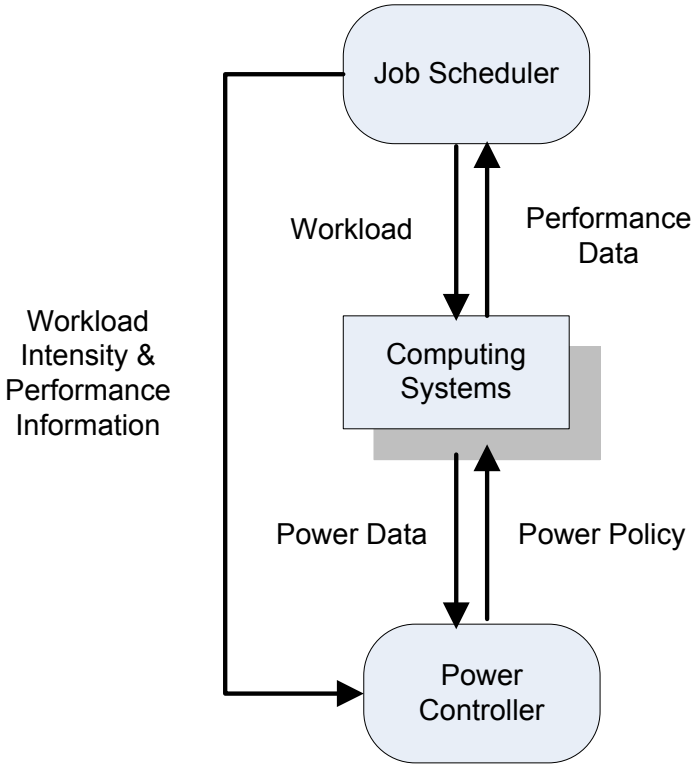

Figure 66: High-level system diagram of a typical performance-aware power saving.

11.3.2 Power Saving on TPCC-UVa

We adapt performance-aware power saving to the benchmark of TPCC-UVa, in which a coordinated approach among different components is proposed. Figure 67 shows the system diagram of performance-aware power saving adapted to the benchmark.

There are several components in the system. The centralized coordinator, the power saving application, reads the performance information from the benchmark through Intel Energy Checker's instrumentation. It provides to the Intel® Data Center Manager the information on whether the workload is running or not, as well as whether the performance gets impacted or not. The Intel Data Center Manager is a power management middleware that provides the building block for performance-aware power saving. Inside Intel Data Center Manager, a utility learner collects the performance feedback, and it builds the utility functions on how well the power saving task is performed. It considers both the power reduced and the performance impact. The utility functions are then used by the policy calculator to generate the decision on the best power policy, which is a power limit on the entire server. The power limit is enforced to Intel® Node Manager, a firmware component. This component is responsible for *capping* the server's power below the power limit with a control loop of retrieving power readings from the power supply unit and regulating the CPU P-state. The process is repeated over time.

The components are described in details in the next sections.

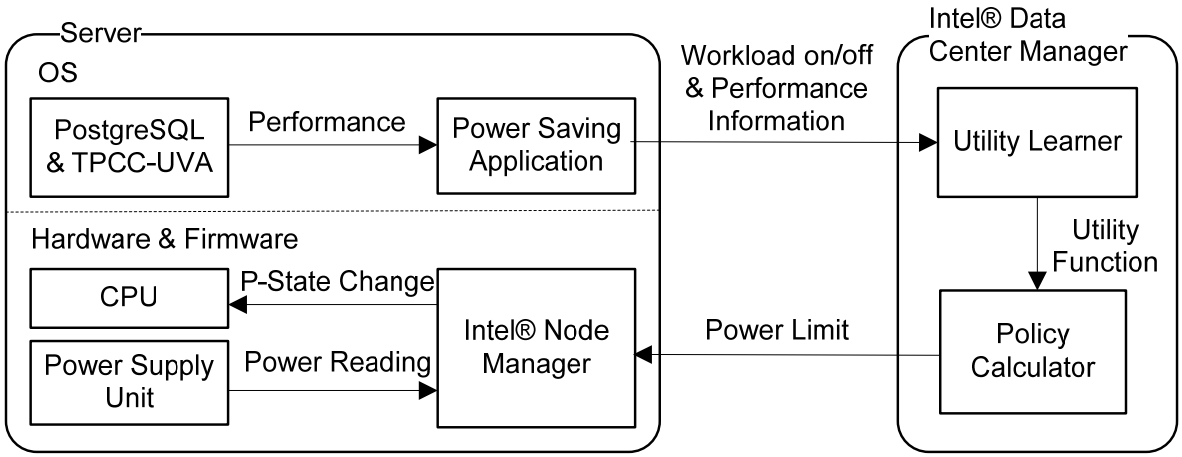

Figure 67: System diagram of power saving on TPCC-UVa.

11.3.2.1. Intel® Node Manager

Intel® Node Manager is a firmware component providing platform level power management capabilities, among which *power capping* is one of the key features.

Power capping (or budgeting) limits the power consumption of a certain server below a configurable threshold under varying workload intensity. The capability enables implementing a variety of advanced usage models, e.g., protecting the platform under failures of some of the power supply units and, as we show in the experiment, reducing the power consumption of the platform with little performance impact.

Intel Node Manager receives the desired power limit from high-level power management software, and employs a feedback controller to cap the platform power. In power capping, the highest P-state of CPU configurable by the OS is taken as the control knob. For a fine grain control of the platform power, Intel Node Manager cycles the highest P-state between two adjacent options at different time lengths. It measures the platform power consumption, given the current highest P-state from the power supply unit, and aggregates the average power consumption over a slightly longer period.

Intel Node Manager then compares the average power consumption with the desired enforced power limit and regulates the two adjacent options for cycling, as well as the elaped times to stay at the two options according to the error measured.

Given such a mechanism, Intel Node Manager is able to maintain the platform power consumption close to the desired threshold, providing fast correction for power bursts under changing workload and maintaining the best possible performance.

11.3.2.2. Intel® Data Center Manager

Intel® Data Center Manager is a middleware that addresses the power and cooling challenges emerging in data centers. It leverages the platform's power

management capabilities, builds advanced usages at the data center level from the perspective of software interaction, and exposes the web service interface for ease of integration with data center management software or cloud operating environment.

Intel Data Center Manager provides the building blocks for performance-aware power saving. APIs are exposed for management software to hint at power optimization with the current workload intensity and the performance impact. Inside Intel Data Center Manager, the optimal power limit policy is sought by maximizing the utility of the power capping decision under the workload intensity.

Considering the background noise, e.g., background processes, minor randomness of workload (whether the performance gets impacted or not given the workload intensity) and the power capping decision are not considered as deterministic.

A probabilistic approach is proposed to evaluate the utility of a certain power limiting decision.

Let *w* denote the current workload intensity and *c* denote the power capping decision of Intel Data Center Manager. Let $R \in \{-1,1\}$ denote the random variable on whether performance gets impacted or not. The utility of the power capping decision under the workload intensity is calculated as

$$U(w,c) \stackrel{\text{def}}{=} P(R=1|w,c)\frac{c_{max}-c}{c_{max}} - k \cdot P(R=-1|w,c)$$

where c_{max} is the maximum power capping allowed, and *k* is a constant giving the weight for trade-off between the reward of power saved without performance impact and the penalty with performance impact.

In Intel Data Center Manager, the probability of whether performance gets impacted or not is estimated online by referring to the performance feedback from the management software. Before estimating the probabilities with maximum likelihood estimation, a heuristic is employed to augment the training set with some *pseudo samples* derived from the existing ones. The idea is that if at time *t*, the performance gets impacted given the workload intensity w_t and the power capping c_t, we regard that with the unknown noise at that time, the performance will also gets impacted, given a larger workload intensity ($w > w_t$) or a smaller power capping ($c < c_t$). Similarly, if at time *t*, the performance does not get impacted, given the workload intensity w_t and the power capping c_t, we regard that with the unknown noise at that time, the performance will not get impacted as well given a smaller workload intensity ($w < w_t$) or a larger power capping ($c > c_t$). The heuristic balances exploration and exploitation in the typical online learning paradigm and smoothes the probabilities estimated.

The pseudo code of performance-aware power saving is outlined below.

```
Initialize T ← ∅
Repeat
    If workload intensity w_t is updated
        c_t ← argmax_c U(w_t, c)
        Enforce power capping decision c_t
    End If
    If performance feedback r_t is updated
        If r_t = 1
            T ← T ∪ {(r_t, w, c)|w ≤ w_t & c ≥ c_t}
        End If
        If r_t = −1
            T ← T ∪ {(r_t, w, c)|w ≥ w_t & c ≤ c_t}
        End If
        Estimate probability distribution P(R = ∗ |W = ∗, C = ∗)
          with T
        c_t ← argmax_c U(w_t, c)
        Enforce power capping decision c_t
    End If
End Repeat
```

NOTE

Additional information on the Intel Data Center Manager and download instructions can be found here:
http://software.intel.com/sites/datacentermanager/

11.3.2.3. Energy Saving Application

The power saving application acts as a coordinator, bridging the TPCC-UVa benchmark with the power saving building block in Intel Data Center Manager. It provides the 0-1 workload intensity information to Intel Data Center Manager, indicating whether the benchmark is running or not. The application also reads the performance counters output by Intel Energy Checker in TPCC-UVa, and it judges whether the performance gets impacted or not. The performance feedback is provided to Intel Data Center Manager for updating the utility of power capping decisions.

In judging whether the performance gets impacted or not, the application first periodically checks the frontend counters of each of the client terminals. The number of transactions processed, and number of transactions regarded as well-done for each of the five transaction types within the period, are collected. The data is aggregated across all the client terminals and the percentage of transactions processed as well-done ones are calculated. By comparing the percentage values of each of the five transaction types with a predefined threshold, the application judges whether the client side experiences get impacted or not.

The application also monitors the performance counter of the backend database. The number of transactions completed in the recent five seconds is periodically

sampled. Before we start power saving, we performed a preliminary run of the benchmark without power saving turned on, collecting a set of samples. During the run with power saving, the application collects a few samples within an interval, and compares the running samples with the original sample set. Welch's t-test is performed to test whether the difference of the mean number of transactions completed from the two sample sets are statistically significant or not. Specifically, it computes the *t* value as

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}$$

where $\overline{X}_i$, s_i^2, and N_i are the *i*th sample mean, sample variance, and sample size. The Welch-Satterthwaite Equation is used to approximate the degrees of freedom associated with the variance as

$$\upsilon \approx \frac{(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2})^2}{\frac{s_1^4}{N_1^2(N_1 - 1)} + \frac{s_2^4}{N_2^2(N_2 - 1)}}$$

Once t and υ are computed, the values are used with the t-distribution to test whether the difference of the means is statistically significant or not, in terms of comparing the p-value with a predefined significance level.

For the power saving application to provide the feedback that the performance does not get impacted, it requires both the frontend performance and the backend performance pass the check.

11.3.3 Experimental Results

Experiments were conducted for a comparative study on TPCC-UVa benchmark, investigating the effects of performance-aware power saving versus power saving with the power management module in operating system.

In the next sections, we first describe the experimental setup with detail configurations and settings and then present the experimental results.

11.3.3.1. Experimental Settings

We used a server with two (2) Intel® Xeon® processor X5570 and 12GB memory in the experimental running of the benchmark. Each processor has four cores, and hyper-threading is enabled.

The operating system on the server was Red Hat Enterprise 5.2. Throughout the experiment, we enabled the on-demand governor in the operating system, which provides the functionality of P-state regulation based on CPU utilization. We regard this configuration as the baseline.

TPCC-UVa benchmark is set up on the server. Before running the experiments, we tried different configurations of the benchmark as a preliminary step to maximize the throughput, reaching the settings of 24 warehouses and 10 terminals per warehouse. In our experiments, we kept the ramp-up period of TPCC-UVa to ten (10) minutes and the measurement period to be four (4) hours.

In performance-aware power saving with a coordinated approach, the power saving application is located on the same server running TPCC-UVa. The middleware, Intel Data Center Manager v2.1, however, is set up on a different server, simulating the deployment scenario that the power management middleware locates on a centralized node managing the others in the entire data center. Intel Node Manager v1.5 is enabled on the server running the benchmark and is managed by the instance of Intel Data Center Manager.

The power saving application judges whether the performance gets impacted or not every minute. To check whether the frontend performance gets impacted or not, 99 percent is used as the threshold for the values of percentage of well-done transactions for each of the five transaction types in the minute.

To check whether the backend performance gets impacted or not, we first ran the benchmark for one (1) hour. Every 15 seconds we looked into the performance counter for number of transactions completed in the most recent five (5) seconds, resulting in a static sample set of 240 samples without power saving. During performance-aware power saving, in every minute, the power saving application samples the performance counter four (4) times, resulting in a running sample set with four (4) samples. Welch t-test is then applied to the two data sets with a significance level of 0.005 to judge whether the backend performance gets impacted or not.

In Intel Data Center Manager, the weight for penalizing with the performance impact in utility calculation is set to one (1).

11.3.3.2. Experimental Results

We ran the TPCC-UVA benchmark with power saving of on-demand governor in Red Hat Enterprise 5.2 and with performance-aware power saving respectively. Table 16 shows the experimental results of performance reported by TPCC-UVa and average power during the running of the benchmark for both the baseline and performance-aware power saving.

From the experimental results, we see that comparing with power saving of the power management module in the operating system, more than 10 percent power reduction is achieved with performance-aware power saving, while the performance of the benchmark gets little impact.

The experiment demonstrates that compared to power saving of the power management module in operating systems that refer to the resource utilization counters, additional energy can be saved with little performance impact by incorporating application level performance feedback into power management software through a coordination of multiple components.

Table 16: Results of of Power Saving on TPCC-UVa

Power Saving Method	Performance	Average Power
Baseline	300 tpmC	183W
Performance-Aware	300 tpmC	164W

NOTE

The experimental system has relatively strong computational capabilities compared to a relatively weak I/O subsystem. As a typical online transaction processing workload, the performance of TPCC-UVa is likely to be bounded by the performance of the disk on the system. In such a scenario, it is not a difficult conjecture that additional power reduction could be exploited by throttling the CPUs, if the workload performance is monitored and is observed to get maintained.

11.4 Ideas for Furture Work

In this heuristic, basic software knowledge and virtualization technology are leveraged (see Figure 68). Virtualization is often used in the data center to consolidate applications hosted by several older servers on a newer and more energy efficient system. Even if this approach yields measurable energy savings, in this heuristic, virtualization is exclusively used as a convenient method to shuffle energy-aware applications among various servers. To describe the dynamic migration energy saving heuristic, the previously instrumented PostgreSQL is showcased. Of course, any other application can be used, assuming it is instrumented to expose (at minimum) the amount of useful work it does.

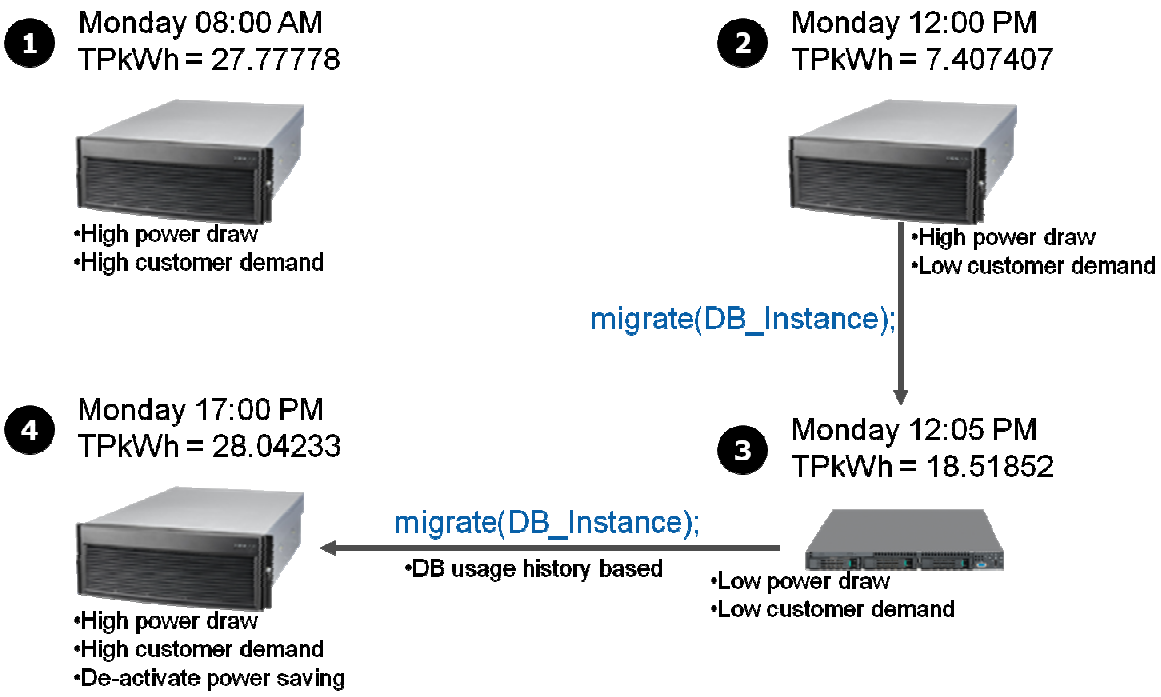

Figure 68: Four steps of the dynamic migration scenario.

The main PostgreSQL monitoring process dynamically computes the energy efficiency metrics of the data base. To handle manageable values, we add the transactions per kilowatt-hour energy efficiency metric to the existing counters (TPkWh – the higher the better). The Intel Energy Checker SDK is used as-is in a virtual machine. The sole difference with previous case studies is that a distributed setup must be used. Refer to the SDK manuals for details on how to use the Intel Energy Checker SDK in a distributed configuration.

Once the energy efficiency metrics are known, it is possible for the database engine to use the virtual machine on which it runs, associated with the hypervisor's VM migration mechanism, to autonomously – or in a middle-ware assisted way – initiate a migration to a different server. Table 17 summarizes the various values used during the following discussion.

Table 17: Useful work, power and energy efficiency metrics

Transactions per second	Power draw per second (W)	TPkWh
100,000	1000	27.77778
20,000	820	6.775068
20,000	300	18.51852
106,000	1050	28.04233

The dynamic migration heuristics can be described as follows. It is Monday morning, and as usually, the customers are actively placing their orders. Thus, a considerable amount of useful work is done by the database back-end. The server running the VM hosting the PostgreSQL instance is powerful enough to allow the application to process 100,000 transactions every second. The TPkWh metric is then equal to 27.78.

As the day goes on, the closer we get to noon, the number of customer orders is dropping. The related transactions count is then 20,000. Even if the operating system and the hardware compensates for the system's utilization drop by activating several power saving features, the TPkWh metric drops to 6.78. Since this value is less than an arbitrary threshold – e.g., 9 – the back-end can autonomously initiate or request a migration toward a less powerful server. While that server has less power to process customer requests, it only needs to process 20,000 transactions, and thus the TPkWh metric rises to 18.52.

Another important part of the heuristic consists of monitoring the utilization of the system by customers over time. Based on this historical data, the application can identify periods of time where spikes in demand usually happen. In addition to a self-learning system, a database administrator-inputted policies can be added to override the autonomous mechanism.

In our scenario, this second part of the heuristics has learned over the last months of operation, that around five o'clock in the afternoon, a huge demand happens during week days. Therefore, a few minutes prior to this time, the back-end can request or initiate its migration back to a powerful server. In this case, it can also request the de-activation of some performance limiting power saving features of the operating system and / or the hardware. In these conditions, the platform can gain a few thousand additional transactions,

reaching a peak of 106,000 for a power draw of 1050 watts. The TPkWh metric rises to 28.04. And so on…

This simple scenario can easily be implemented, since all the building blocks are available: the Intel Energy Checked SDK can be used to compute and dynamically exchange the TPkWh metric – likely using a PSU (power Supply Unit)-based power sensor readable via IPMI (Intelligent Platform Management Interface) –, and hypervisors offering VM migration capabilities are commonly available.

12 Appendix

12.1 Network Setup

Networking is not within the scope of this SDK. Supported operating systems and best administration practices should be used to setup the network infrastructure. In a single system configuration, no network is required.

NOTE

The use of Intel Energy Checker SDK ensures there is no data collision if all the PL data is aggregated into a single point. However, it is not the role of the SDK to provide a transport mechanism.

The following section presents the setup and configuration of two systems running heterogeneous operating systems with remote data aggregation. NFS is used in this example, for its ubiquity and good performance, but any other transport mechanism can be used to aggregate the PL counter data.

12.1.1 Example Setup

This section presents the steps required to setup a configuration with one *monitoring* system hosting the PL data aggregation point and one *monitored* system providing the PL data (Figure 69).

To illustrate this setup, the aggregation system runs 32-bit Microsoft Windows* Server 2003 and the monitored system runs 64-bit Linux*. The transport used in this example is NFS. In this setup, it is trivial to multiply the number of monitored systems (running under any supported operating system).

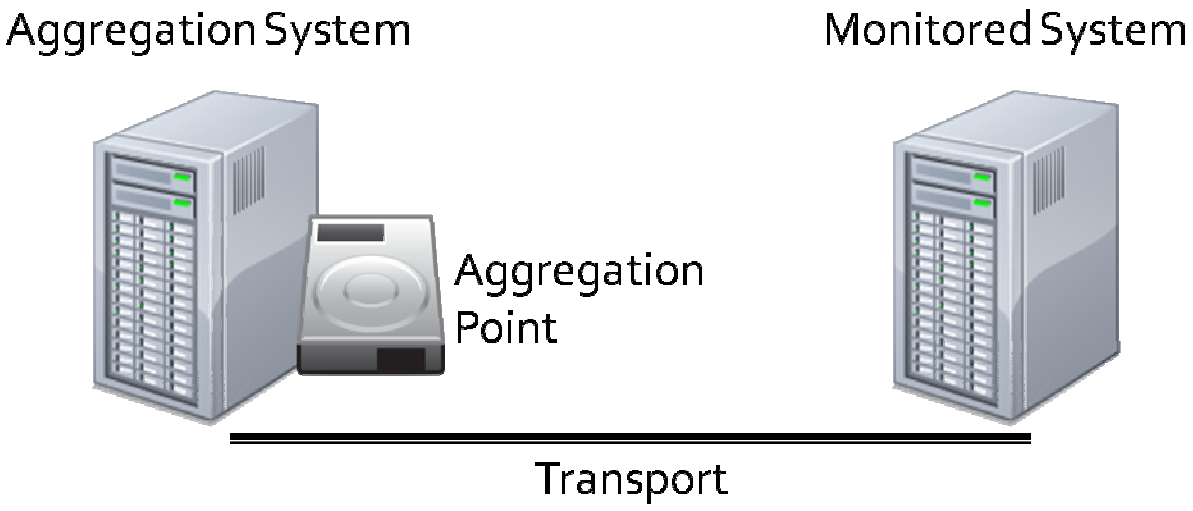

Figure 69: Remote monitoring setup

Figure 70 depicts a remote energy monitoring scenario, in which the monitored system is attached to the wall via an external power analyzer. Arbitrarily, ESRV is running on the aggregation system. Note that this doesn't change anything for any of the applications running on the monitored or the aggregated system, as PL_FOLDERs are aggregated. An energy-aware application running on the monitored system can still use the ESRV data (see Section 8) to expose its own energy efficiency counters.

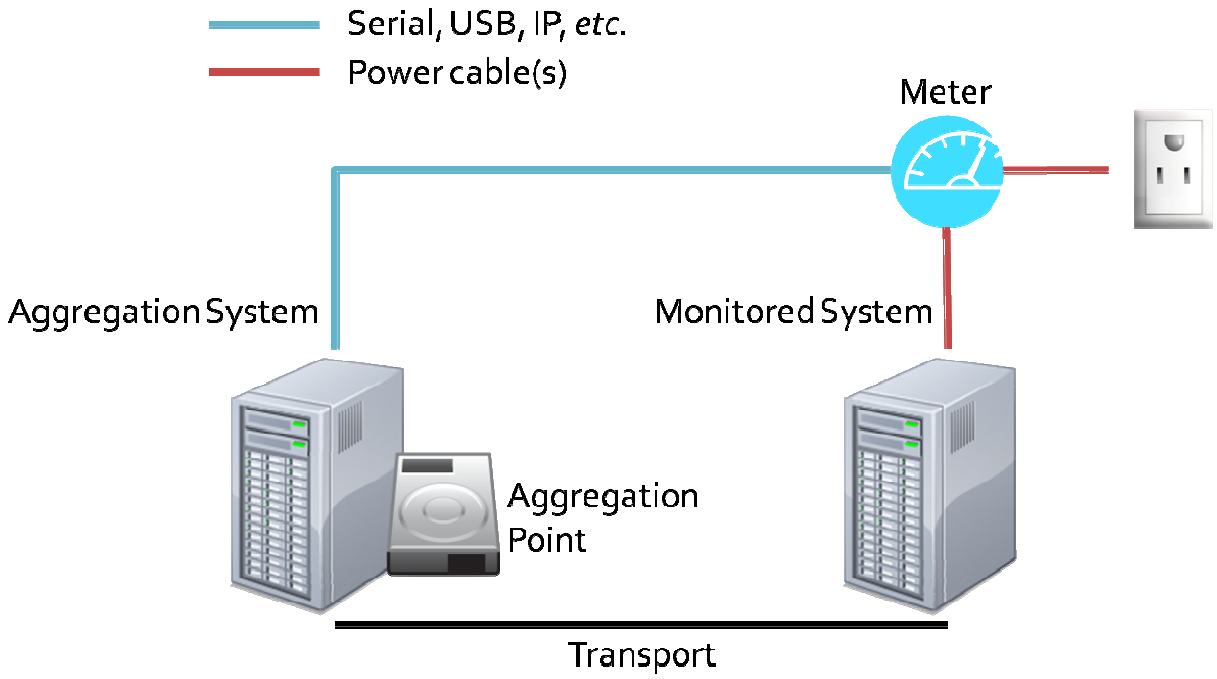

Figure 70: Remote energy monitoring setup

NOTE

The setup presented doesn't apply best known methods that administrators should implement in a production environment. This setup is targeting a test configuration an engineer would use to perform energy efficiency analysis of a given application.

12.1.2 Configure NFS on Windows

Microsoft makes available for its 32-bit servers the Windows Services for UNIX Version 3.5 for Windows Server 2003 which contains an NS server (http://www.microsoft.com/downloads/details.aspx?FamilyID=896C9688-601B-44F1-81A4-02878FF11778&displaylang=en). Note that multiple vendors provide NFS implementations for Windows. Windows Services for UNIX was chosen for this demonstration because it is available for free from Microsoft. Please follow the installation instructions for Windows Services for UNIX.

In addition to the default installation, the extra configuration steps are shown in this section. Start the Microsoft Management Console (mmc) and open the Windows Services for UNIX snap-in.

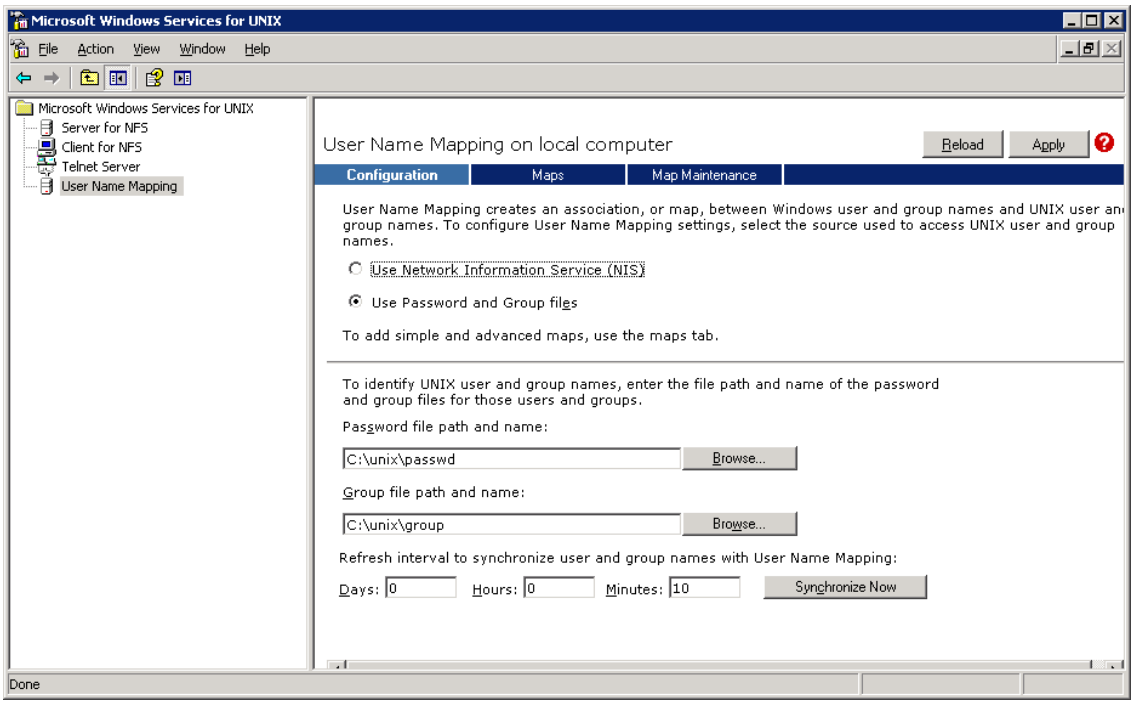

Figure 71: Password and groups settings

The password and group user mapping are selected.

Copy the `password` and `groups` files from the monitored (Linux) system to the aggregation system and point them to the *Password file path and name* and *Group file path and name* section of the mmc (Figure 71).

After providing the monitored system users and group data it is possible to map users between both systems. Note that this is required to manage the access rights to the PL aggregation point (Figure 72).

Mapping is straight-forward: select first the Windows user in the *Windows Users* list. Select then the Linux, Solaris 10, or MacOS X user in the *UNIX Users* list and click on *Add*. If no conflicts are detected, click on *Apply*.

Now create the NFS share (sharing the PL_FOLDER – `c:\productivity_link` here) and apply access rights based on the mapped users. If security is not a concern, a mapping between *Administrator* and *root* (with no ACL on the NFS share) allow full access.

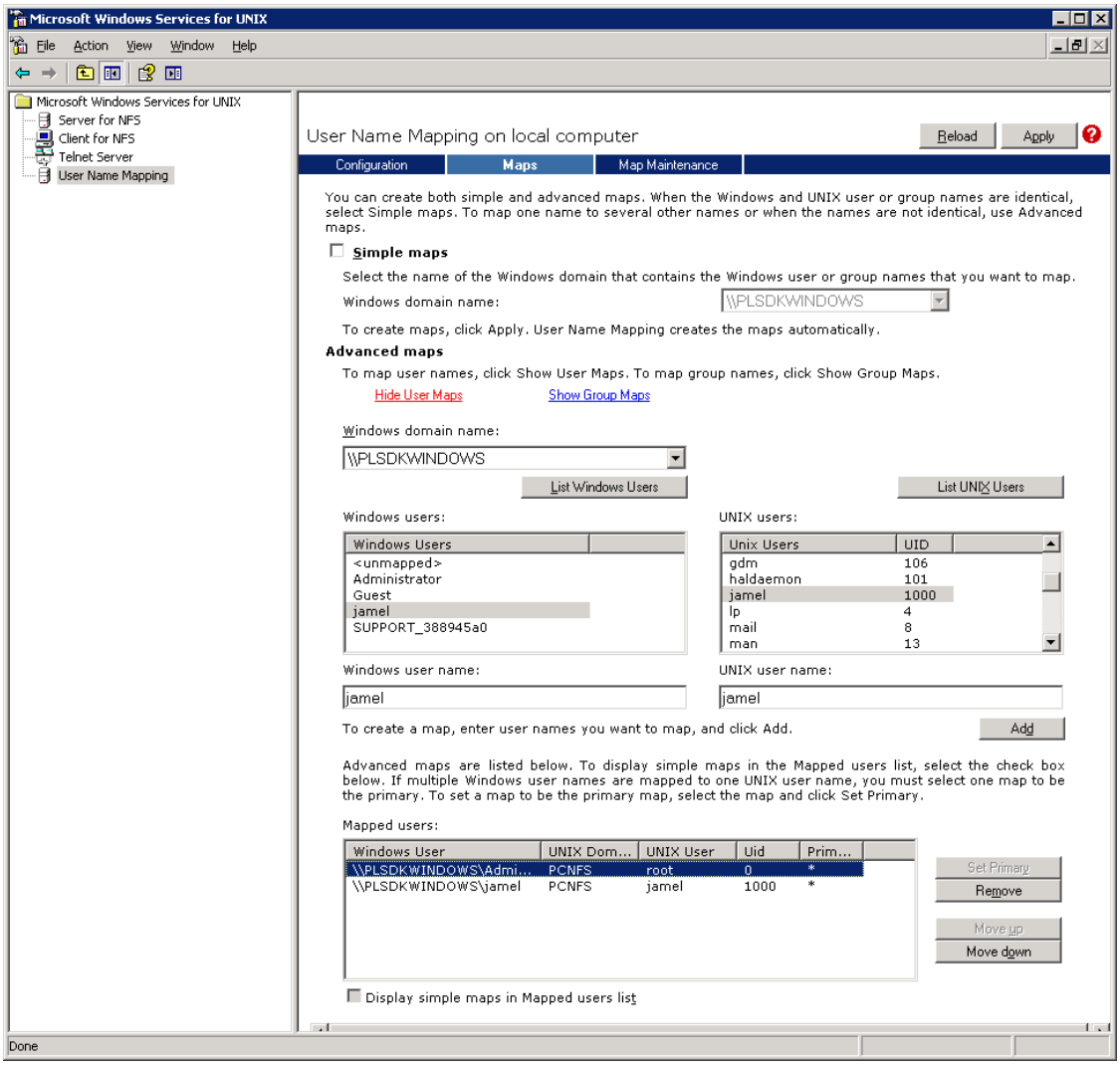

Figure 72: Remote monitoring setup

12.1.3 Configure NFS on Linux

Install NFS support on the Linux system if not already installed.

In this setup, the Linux systems are clients and don't require the NFS server components to be installed and configured. The sole configuration step consists in mounting the Windows system's NFS share (`c:\productivity_link`) on the local PL_FOLDER (`/opt/productivity_link`). Use Table 18 as a reference, if required, and use `<ip_address>` as the IP address of the NFS server.

Table 18: NFS mount commands for supported OSes

OS	mount command
Solaris 10	`mount -F nfs <ip_address>:/productivity_link /opt/productivity_link/`
Linux	`mount -t nfs <ip_address>:/productivity_link /opt/productivity_link/`
MacOS X	`mount -t nfs <ip_address>:/productivity_link /opt/productivity_link`

12.1.4 Remote Monitoring Example

At this stage, any PL data generated by applications running on the Linux, Solaris 10, or MacOS X system(s) are visible from the aggregation system running under Windows. As for our example, we will use the linux_vmstat_info sample (see Section 10.4 for details) to generate PL data (exposing Linux vmstat data). Start the sample as illustrated below.

```
./linux_vmstat_info
```

```
@      @@@@@  @    @ @    @ @   @         @    @ @    @         @@@@   @@@@@   @@    @@@@@@@@@@  @@@@
 @        @    @@   @ @    @  @ @          @    @ @@  @@        @    @    @    @  @     @ @      @    @
 @        @    @ @  @ @    @   @           @    @ @ @@ @        @         @   @    @    @ @      @    @
 @        @    @ @  @ @    @   @           @    @ @    @         @@@@     @   @    @    @ @@@@@  @    @
 @        @    @  @ @ @    @   @             @   @ @    @             @    @   @@@@@@    @ @      @    @
 @        @    @   @@ @    @  @ @             @ @  @    @             @    @   @    @    @ @      @    @
 @@@@@@ @@@@@  @    @  @@@@  @   @             @   @    @        @@@@@     @   @    @    @ @       @@@@
 
Using PL.........: [da684348-0588-4a7a-9c95-c2276cd93e40]
Exporting........:
.................: [nr_dirty]
.................: [nr_writeback]
.................: [nr_unstable]
.................: [nr_page_table_pages]
.................: [nr_anon_pages]
.................: [nr_mapped]
.................: [nr_slab]
.................: [pgpgin]
.................: [pgpgout]
.................: [pswpin]
.................: [pswpout]
```

```
.................: [pgalloc_high]
.................: [pgalloc_normal]
.................: [pgalloc_dma32]
.................: [pgalloc_dma]
.................: [pgfree]
.................: [pgactivate]
.................: [pgdeactivate]
.................: [pgfault]
.................: [pgmajfault]
.................: [pgrefill_high]
.................: [pgrefill_normal]
.................: [pgrefill_dma32]
.................: [pgrefill_dma]
.................: [pgsteal_high]
.................: [pgsteal_normal]
.................: [pgsteal_dma32]
.................: [pgsteal_dma]
.................: [pgscan_kswapd_high]
.................: [pgscan_kswapd_normal]
.................: [pgscan_kswapd_dma32]
.................: [pgscan_kswapd_dma]
.................: [pgscan_direct_high]
.................: [pgscan_direct_normal]
.................: [pgscan_direct_dma32]
.................: [pgscan_direct_dma]
.................: [pginodesteal]
.................: [slabs_scanned]
.................: [kswapd_steal]
.................: [kswapd_inodesteal]
.................: [pageoutrun]
.................: [allocstall]
.................: [pgrotated]
.................: [nr_bounce]
To Stop Logging  : [<CRTL>+<C>]
Samples          : [6167]
```

On the **monitoring** system, start the pl_gui_monitor as shown below (see *Intel Energy Checker SDK Companion Applications User Guide* for details).

```
1  iecsdk\bin\companion_applications\pl_gui_monitor\windows\X86>pl_gui_monitor.exe --
2  process --gdiplus --transparency 20 --top --title vmstat --format --page 2
```

Figure 73 depicts the monitoring result. Note that any application (assuming security allows) using the API has access to any of this information using a call to the `pl_read` API function.

An extra step can be performed, as described in Section 7, which consists of converting the Intel Energy Checker counters into native Windows counters. Once the conversion is done, it is now possible to monitor the PL data from the Linux remote machine using standard Windows monitoring / administration tools, such as Perfmon. Figure 74 show the simultaneous monitoring with Perfmon and pl_gui_monitor.

Figure 75 shows the result of cross-monitoring of multiple PL data between Windows and Linux. Eleven (11) applications are running simultaneously on the networked systems. The Linux system is running five (5) applications: one CSV logger – pl_csv-logger – logging data from the instrumented POV-Ray running under Windows; and the four (4) monitoring applets – linux_system_infos – exporting a total of 102 system variables using the Intel Energy Checker API.

On the Windows box, we have four (4) instances of GUI monitors – pl_gui_monitor – each consuming the data exported by one of the linux_system_infos applets. An additional instance of GUI monitor is monitoring the same POV-Ray instance, which is logged on the Linux system by the CSV logger. All these applications (11) are running in parallel and are using five (5) PLs – aggregated on the Windows system.

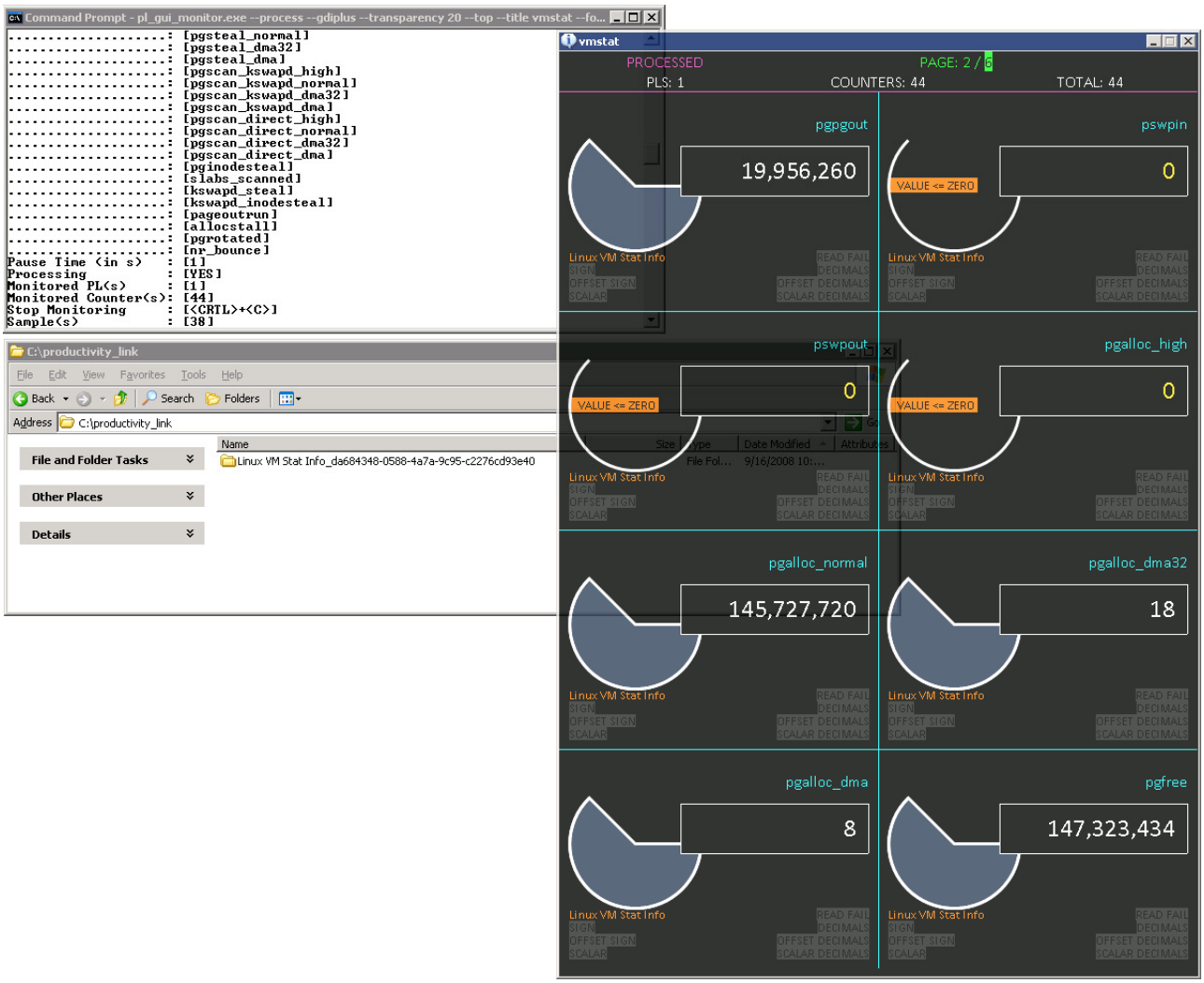

Figure 73: Remote monitoring of Linux vmstat on Windows

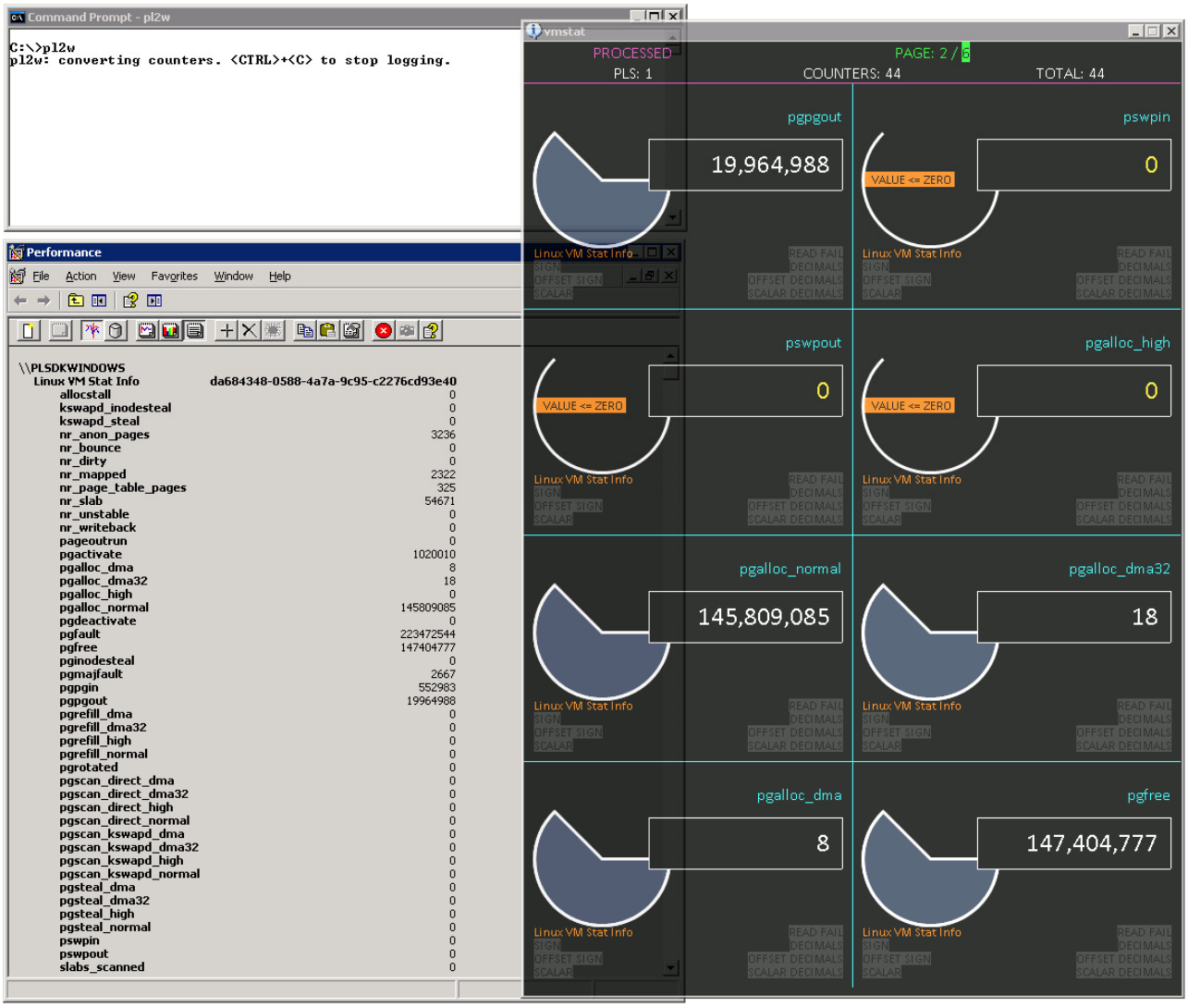

Figure 74: Remote monitoring Linux vmstat via Windows Perfmon

NOTE

It is possible to manually apply the `.decimals` *suffix counter in Perfmon to Intel® EC converted counters with pl2w. For example, if* `.decimals` *equals two (2) for a PL counter monitored in Perfmon, do the following:*

- *Select the counter and right-click on it*
- *Select Properties*
- *In the Data tab, select 0.01 in the Scale drop-down list (for 10^-2)*

This only applies to the `.decimals` *suffix and there is no way to take other suffix counters into account in Perfmon.*

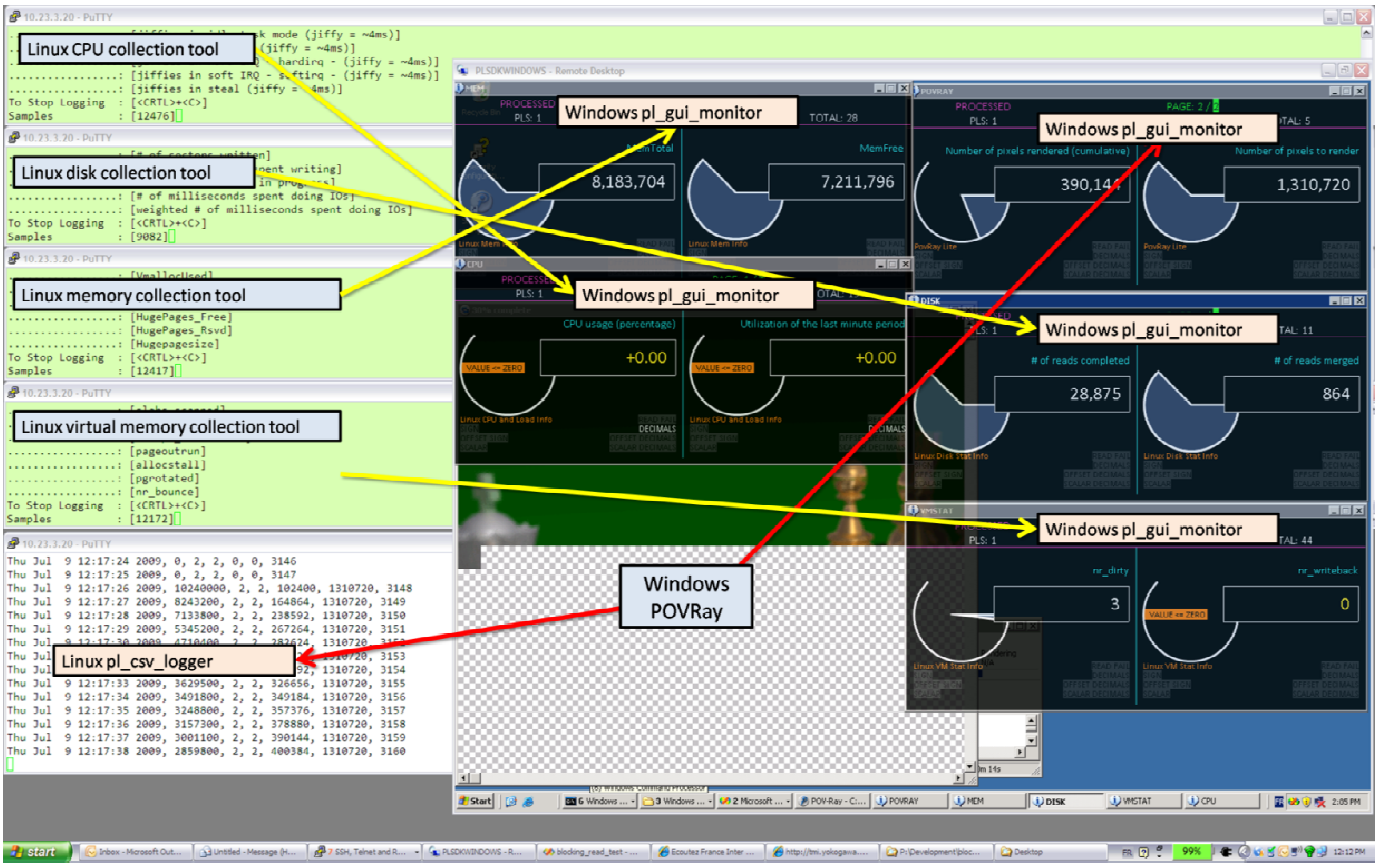

Figure 75: Remote monitoring example with Linux and Windows