PAPI Programmer's Reference

This document is a compilation of the reference material needed by a programmer to effectively use PAPI. It is identical to the material found in the PAPI man pages, but organized in a way that may be more portable and accessible. The information here is extensively hyperlinked, which makes it useful in electronic formats, but less useful in hardcopy format.
For other PAPI documentation, see also, the **PAPI User's Guide**.

NAME

NAME

PAPI - Performance Application Programming Interface

SYNOPSIS

The PAPI Performance Application Programming Interface provides machine and operating system independent access to hardware performance counters found on most modern processors. Any of over 100 preset events can be counted through either a simple high level programming interface or a more complete low level interface from either C or Fortran. A list of the function calls in these interfaces is given below, with references to other pages for more complete details. For general information on the Fortran interface see: PAPIF

PAPI Presets

An extensive list of predefined events is implemented on all systems where they can be supported. For a list of these events, see: PAPI_presets

PAPI Native Events

PAPI also supports interface functions for discovering the native events on a given platform. For more information on native events, see: PAPI_native

High Level Functions

A simple interface for instrumenting end-user applications. Fully supported on both C and Fortran. See individual functions for details on usage.

```
PAPI_num_counters  - get the number of hardware counters available on
the system
PAPI_flips  - simplified call to get Mflips/s (floating point
instruction rate), real and processor time
PAPI_flops  - simplified call to get Mflops/s (floating point operation
rate), real and processor time
PAPI_ipc  - gets instructions per cycle, real and processor time
PAPI_accum_counters  - add current counts to array and reset counters
PAPI_read_counters  - copy current counts to array and reset counters
PAPI_start_counters  - start counting hardware events
PAPI_stop_counters  - stop counters and return current counts
```

Note that the high-level interface is self-initializing. You can mix high and low level calls, but you *must* call either PAPI_library_init or a high level routine before calling a low level routine.

Low Level Functions

Advanced interface for all applications and performance tools. Some functions may be implemented only for C or Fortran. See individual functions for details on usage and support.

```
PAPI_accum  - accumulate and reset hardware events from an event set
```

PAPI_perror - convert PAPI error codes to strings
PAPI_profil - generate PC histogram data where hardware counter overflow occurs
PAPI_query_event - query if a PAPI event exists
PAPI_read - read hardware events from an event set with no reset
PAPI_register_thread - inform PAPI of the existence of a new thread
PAPI_remove_event - remove a hardware event from a PAPI event set
PAPI_remove_events - remove an array of hardware events from a PAPI event set
PAPI_reset - reset the hardware event counts in an event set
PAPI_set_debug - set the current debug level for PAPI
PAPI_set_domain - set the default execution domain for new event sets
PAPIF_set_event_domain - set the execution domain for a specific event set. Fortran only.
PAPI_set_granularity - set the default granularity for new event sets
PAPI_set_multiplex - convert a standard event set to a multiplexed event set
PAPI_set_opt - change the option settings of the PAPI library or a specific event set
PAPI_set_thr_specific - save a pointer as a thread specific stored data structure
PAPI_shutdown - finish using PAPI and free all related resources
PAPI_sprofil - generate hardware counter profiles from multiple code regions
PAPI_start - start counting hardware events in an event set
PAPI_state - return the counting state of an event set
PAPI_stop - stop counting hardware events in an event set and return current events
PAPI_strerror - return a pointer to the error message corresponding to a specified error code
PAPI_thread_id - get the thread identifier of the current thread
PAPI_thread_init - initialize thread support in the PAPI library
PAPI_unlock - unlock one of two PAPI internal user mutex variables
PAPI_unregister_thread - inform PAPI that a previously registered thread is disappearing
PAPI_write - write counter values into counters

PAPI Utility Commands

A collection of simple utility commands is available in the \utils directory. See individual utilities for details on usage.

papi_avail - provides availability and detail information for PAPI preset events
papi_clockres - provides availability and detail information for PAPI preset events
papi_cost - provides availability and detail information for PAPI preset events
papi_command_line - executes PAPI preset or native events from the command line
papi_decode - decodes PAPI preset events into a csv format suitable for PAPI_encode_events
papi_event_chooser - given a list of named events, lists other events that can be counted with them
papi_mem_info - provides information on the memory architecture of the current processor
papi_native_avail - provides detailed information for PAPI native events

SEE ALSO

The PAPI Web site: http://icl.cs.utk.edu/papi
PAPIF, PAPI_presets, PAPI_native

NAME

PAPIF - Performance Application Programming Interface (Fortran)

SYNOPSIS

```
#include fpapi.h
call PAPIF_function_name(arg1,arg2,...,check)
```

DESCRIPTION

Fortran Calling Interface The PAPI library comes with a specific Fortran library interface. The Fortran interface covers the complete library with a few minor exceptions. Functions returning C pointers to structures, such as PAPI_get_opt and PAPI_get_executable_info , are either not implemented in the Fortran interface, or implemented with different calling semantics.
Semantics for specific functions in the Fortran interface are documented on the equivalent C man page. For example, the semantics and functionality of **PAPIF_accum** are covered in the PAPI_accum man page. For most architectures the following relation holds between the pseudo-types listed and Fortran variable types.

Pseudo-type	Fortran type	Description
C_INT	INTEGER	Default Integer type
C_FLOAT	REAL	Default Real type
C_LONG_LONG	INTEGER*8	Extended size integer
C_STRING	CHARACTER*(PAPI_MAX_STR_LEN)	Fortran string
C_INT FUNCTION	EXTERNAL INTEGER FUNCTION	Fortran function returning integer result
C_INT(*)	Array of corresponding type	C_TYPE(*) refers to an array of the corresponding Fortan type. The length of the array needed is context dependent. It may be e.g. PAPI_MAX_HWCTRS or PAPIF_num_counters.
C_FLOAT(*)		
C_LONG_LONG(*)		

Array arguments must be of sufficent size to hold the input/output from/to the subroutine for predictable behavior. The array length is indicated either by the accompanying argument or by internal PAPI definitions. For details on this see the corresponding C routine.
Subroutines accepting **C_STRING** as an argument are on most implementations capable of reading the character string length as provided by Fortran. In these implementations the string is truncated or space padded as necessary. For other implementations the length of the character array is assumed to be of sufficient size. No character string longer than **PAPI_MAX_STR_LEN** is returned by the PAPIF interface.

RETURN VALUES

The return code of the corresponding C routine is returned in the argument **check** in the Fortran interface.

SEE ALSO

The PAPI Interface: PAPI

NAME

PAPI_presets - PAPI predefined named events

SYNOPSIS

```
#include <papi.h>
```

DESCRIPTION

The PAPI library names a number of predefined, or preset events. This set is a collection of events typically found in many CPUs that provide performance counters. A PAPI preset event name is mapped onto one or more of the countable native events on each hardware platform. On any particular platform, the preset can either be directly available as a single counter, derived using a combination of counters or unavailable.
The PAPI preset events can be broken loosely into several categories, as shown in the table below.
PAPI Preset Event Definitions by Category:

Name	Description
Conditional Branching	
PAPI_BR_CN	Conditional branch instructions
PAPI_BR_INS	Branch instructions
PAPI_BR_MSP	Conditional branch instructions mispredicted
PAPI_BR_NTK	Conditional branch instructions not taken
PAPI_BR_PRC	Conditional branch instructions correctly predicted
PAPI_BR_TKN	Conditional branch instructions taken
PAPI_BR_UCN	Unconditional branch instructions
PAPI_BRU_IDL	Cycles branch units are idle
PAPI_BTAC_M	Branch target address cache misses
Cache Requests:	
PAPI_CA_CLN	Requests for exclusive access to clean cache line
PAPI_CA_INV	Requests for cache line invalidation
PAPI_CA_ITV	Requests for cache line intervention
PAPI_CA_SHR	Requests for exclusive access to shared cache line
PAPI_CA_SNP	Requests for a snoop
Conditional Store:	
PAPI_CSR_FAL	Failed store conditional instructions
PAPI_CSR_SUC	Successful store conditional instructions

PAPI_CSR_TOT	Total store conditional instructions
Floating Point Operations:	
PAPI_FAD_INS	Floating point add instructions
PAPI_FDV_INS	Floating point divide instructions
PAPI_FMA_INS	FMA instructions completed
PAPI_FML_INS	Floating point multiply instructions
PAPI_FNV_INS	Floating point inverse instructions
PAPI_FP_INS	Floating point instructions
PAPI_FP_OPS	Floating point operations
PAPI_FP_STAL	Cycles the FP unit
PAPI_FPU_IDL	Cycles floating point units are idle
PAPI_FSQ_INS	Floating point square root instructions
Instruction Counting:	
PAPI_FUL_CCY	Cycles with maximum instructions completed
PAPI_FUL_ICY	Cycles with maximum instruction issue
PAPI_FXU_IDL	Cycles integer units are idle
PAPI_HW_INT	Hardware interrupts
PAPI_INT_INS	Integer instructions
PAPI_TOT_CYC	Total cycles
PAPI_TOT_IIS	Instructions issued
PAPI_TOT_INS	Instructions completed
PAPI_VEC_INS	Vector/SIMD instructions
Cache Access:	
PAPI_L1_DCA	L1 data cache accesses
PAPI_L1_DCH	L1 data cache hits
PAPI_L1_DCM	L1 data cache misses
PAPI_L1_DCR	L1 data cache reads
PAPI_L1_DCW	L1 data cache writes
PAPI_L1_ICA	L1 instruction cache accesses
PAPI_L1_ICH	L1 instruction cache hits
PAPI_L1_ICM	L1 instruction cache misses
PAPI_L1_ICR	L1 instruction cache reads
PAPI_L1_ICW	L1 instruction cache writes

PAPI_L1_LDM	L1 load misses
PAPI_L1_STM	L1 store misses
PAPI_L1_TCA	L1 total cache accesses
PAPI_L1_TCH	L1 total cache hits
PAPI_L1_TCM	L1 total cache misses
PAPI_L1_TCR	L1 total cache reads
PAPI_L1_TCW	L1 total cache writes
PAPI_L2_DCA	L2 data cache accesses
PAPI_L2_DCH	L2 data cache hits
PAPI_L2_DCM	L2 data cache misses
PAPI_L2_DCR	L2 data cache reads
PAPI_L2_DCW	L2 data cache writes
PAPI_L2_ICA	L2 instruction cache accesses
PAPI_L2_ICH	L2 instruction cache hits
PAPI_L2_ICM	L2 instruction cache misses
PAPI_L2_ICR	L2 instruction cache reads
PAPI_L2_ICW	L2 instruction cache writes
PAPI_L2_LDM	L2 load misses
PAPI_L2_STM	L2 store misses
PAPI_L2_TCA	L2 total cache accesses
PAPI_L2_TCH	L2 total cache hits
PAPI_L2_TCM	L2 total cache misses
PAPI_L2_TCR	L2 total cache reads
PAPI_L2_TCW	L2 total cache writes
PAPI_L3_DCA	L3 data cache accesses
PAPI_L3_DCH	L3 Data Cache Hits
PAPI_L3_DCM	L3 data cache misses
PAPI_L3_DCR	L3 data cache reads
PAPI_L3_DCW	L3 data cache writes
PAPI_L3_ICA	L3 instruction cache accesses
PAPI_L3_ICH	L3 instruction cache hits
PAPI_L3_ICM	L3 instruction cache misses
PAPI_L3_ICR	L3 instruction cache reads

PAPI_L3_ICW	L3 instruction cache writes
PAPI_L3_LDM	L3 load misses
PAPI_L3_STM	L3 store misses
PAPI_L3_TCA	L3 total cache accesses
PAPI_L3_TCH	L3 total cache hits
PAPI_L3_TCM	L3 cache misses
PAPI_L3_TCR	L3 total cache reads
PAPI_L3_TCW	L3 total cache writes
Data Access:	
PAPI_LD_INS	Load instructions
PAPI_LST_INS	Load/store instructions completed
PAPI_LSU_IDL	Cycles load/store units are idle
PAPI_MEM_RCY	Cycles Stalled Waiting for memory Reads
PAPI_MEM_SCY	Cycles Stalled Waiting for memory accesses
PAPI_MEM_WCY	Cycles Stalled Waiting for memory writes
PAPI_PRF_DM	Data prefetch cache misses
PAPI_RES_STL	Cycles stalled on any resource
PAPI_SR_INS	Store instructions
PAPI_STL_CCY	Cycles with no instructions completed
PAPI_STL_ICY	Cycles with no instruction issue
PAPI_SYC_INS	Synchronization instructions completed
TLB Operations:	
PAPI_TLB_DM	Data translation lookaside buffer misses
PAPI_TLB_IM	Instruction translation lookaside buffer misses
PAPI_TLB_SD	Translation lookaside buffer shootdowns
PAPI_TLB_TL	Total translation lookaside buffer misses

AUTHORS

Nils Smeds <smeds@cs.utk.edu>

BUGS

The exact semantics of an event counter are platform dependent. PAPI preset names are mapped onto available events in a way so as to count as similar types of events as possible on different platforms. Due

to hardware implementation differences it is not necessarily possible to directly compare the counts of a particular PAPI event obtained on different hardwareplatforms.

SEE ALSO

PAPI, PAPI_native, PAPI_enum_event, PAPI_get_event_info, PAPI_event_code_to_name, PAPI_event_name_to_code

NAME

PAPI_native - Accessing PAPI native events

SYNOPSIS

```
#include <papi.h>
```

DESCRIPTION

In addition to the predefined PAPI preset events, the PAPI library also exposes a majority of the events native to each platform. Native events form the basic building blocks for PAPI presets. They can also be used directly to access functions specific to a given platform.
Since native events are *by definition* specific to each platform, the names for these events are unique to each platform. Native events for a given platform can be discovered by combining the PAPI_enum_event and PAPI_event_code_to_name or PAPI_get_event_info functions.

BUGS

Not every native event on every platform can be represented through the native event interface. Occasionally, exotic but valuable events are not represented. There is presently no method for representing these events in a PAPI event set.

SEE ALSO

PAPI, PAPI_presets, PAPI_enum_event, PAPI_get_event_info, PAPI_event_code_to_name, PAPI_event_name_to_code

NAME

papi_avail - provides availability and detail information for PAPI preset events.

SYNOPSIS

papi_avail [*-adht*] [*-e event*]

DESCRIPTION

papi_avail is a PAPI utility program that reports information about the current PAPI installation and supported preset events. Using the -e option, it will also display information about specific native events.

OPTIONS

-a Display only the available PAPI preset events.

-d Display PAPI preset event information in a more detailed format.

-h Display help information about this utility.

-t Display the PAPI preset event information in a tabular format. This is the default.

-e <event>

Display detailed event information for the named event. This event can be either a preset or a native event.

BUGS

There are no known bugs in this utility.
If you find a bug, it should be reported to the PAPI Mailing List at <ptools-perfapi@ptools.org>.

SEE ALSO

PAPI, papi_clockres, papi_command_line, papi_cost, papi_decode, papi_event_chooser, papi_mem_info, papi_native_avail

NAME

papi_clockres - measures and reports clock latency and resolution for PAPI timers.

SYNOPSIS

papi_clockres

DESCRIPTION

papi_clockres is a PAPI utility program that measures and reports the latency and resolution of the four PAPI timer functions: **PAPI_get_real_cyc(), PAPI_get_virt_cyc(), PAPI_get_real_usec() and PAPI_get_virt_usec().**

OPTIONS

This utility has no command line options.

BUGS

There are no known bugs in this utility.
If you find a bug, it should be reported to the PAPI Mailing List at <ptools-perfapi@ptools.org>.

SEE ALSO

PAPI, papi_avail, papi_command_line, papi_cost, papi_decode, papi_event_chooser, papi_mem_info, papi_native_avail

NAME

papi_cost - computes execution time costs for basic PAPI operations.

SYNOPSIS

papi_cost [*-dhs*] [*-b bins*] [*-t threshold*]

DESCRIPTION

papi_cost is a PAPI utility program that computes the min / max / mean / std. deviation of execution times for PAPI start/stop pairs and for PAPI reads. This information provides the basic operating cost to a user's program for collecting hardware counter data. Command line options control display capabilities.

OPTIONS

-b <bins>

Define the number of bins into which the results are partitioned for display. The default is 100.

-d Display a graphical distribution of costs in a vertical histogram.

-h Display help information about this utility.

-s Show the number of iterations in each of the first 10 standard deviations above the mean.

-t <threshold>

Set the threshold for the number of iterations to measure costs. The default is 100,000.

BUGS

There are no known bugs in this utility.
If you find a bug, it should be reported to the PAPI Mailing List at <ptools-perfapi@ptools.org>.

SEE ALSO

PAPI, papi_avail, papi_clockres, papi_command_line, papi_decode, papi_event_chooser, papi_mem_info, papi_native_avail

NAME

papi_command_line - executes PAPI preset or native events from the command line.

SYNOPSIS

papi_command_line <event> <event> ...

DESCRIPTION

papi_command_line is a PAPI utility program that adds named events from the command line to a PAPI EventSet and does some work with that EventSet. This serves as a handy way to see if events can be counted together, and if they give reasonable results for known work.

OPTIONS

This utility has no command line options.

BUGS

There are no known bugs in this utility.
If you find a bug, it should be reported to the PAPI Mailing List at <ptools-perfapi@ptools.org>.

SEE ALSO

PAPI, papi_avail, papi_clockres, papi_cost, papi_decode, papi_event_chooser, papi_mem_info, papi_native_avail

NAME

papi_decode - provides availability and detail information for PAPI preset events.

SYNOPSIS

papi_decode [*-ah*]

DESCRIPTION

papi_decode is a PAPI utility program that converts the PAPI presets for the existing library into a comma separated value format that can then be viewed or modified in spreadsheet applications or text editors, and can be supplied to PAPI_encode_events as a way of adding or modifying event definitions for specialized applications. The format for the csv output consists of a line of field names, followed by a blank line, followed by one line of comma separated values for each event contained in the preset table. A portion of this output (for Pentium 4) is shown below:

```
      name,derived,postfix,short_descr,long_descr,note,[native,...]

PAPI_L1_ICM,NOT_DERIVED,,"L1I cache misses","Level 1 instruction cache
misses",,BPU_fetch_request_TCMISS
PAPI_L2_TCM,NOT_DERIVED,,"L2 cache misses","Level 2 cache
misses",,BSQ_cache_reference_RD_2ndL_MISS_WR_2ndL_MISS
PAPI_TLB_DM,NOT_DERIVED,,"Data TLB misses","Data translation lookaside buffer
misses",,page_walk_type_DTMISS
```

OPTIONS

-a Convert only the available PAPI preset events.

-h Display help information about this utility.

BUGS

There are no known bugs in this utility.
If you find a bug, it should be reported to the PAPI Mailing List at <ptools-perfapi@ptools.org>.

SEE ALSO

PAPI, papi_avail, papi_clockres, papi_command_line, papi_cost, papi_event_chooser, papi_mem_info, papi_native_avail

NAME

papi_event_chooser - given a list of named events, lists other events that can be counted with them.

SYNOPSIS

papi_event_chooser NATIVE | PRESET <event> <event> ...

DESCRIPTION

papi_event_chooser is a PAPI utility program that reports information about the current PAPI installation and supported preset events.

OPTIONS

This utility has no command line options.

BUGS

There are no known bugs in this utility.
If you find a bug, it should be reported to the PAPI Mailing List at <ptools-perfapi@ptools.org>.

SEE ALSO

PAPI, papi_avail, papi_clockres, papi_command_line, papi_cost, papi_decode, papi_mem_info, papi_native_avail

NAME

papi_mem_info - provides information on the memory architecture of the current processor.

SYNOPSIS

papi_mem_info

DESCRIPTION

papi_mem_info is a PAPI utility program that reports information about the cache memory architecture of the current processor, including number, types, sizes and associativities of instruction and data caches and Translation Lookaside Buffers.

OPTIONS

This utility has no command line options.

BUGS

There are no known bugs in this utility.
If you find a bug, it should be reported to the PAPI Mailing List at <ptools-perfapi@ptools.org>.

SEE ALSO

PAPI, papi_avail, papi_clockres, papi_command_line, papi_cost, papi_decode, papi_event_chooser, papi_native_avail

NAME

papi_native_avail - provides detailed information for PAPI native events.

SYNOPSIS

papi_native_avail

DESCRIPTION

papi_native_avail is a PAPI utility program that reports information about the native events available on the current platform. A native event is an event specific to a specific hardware platform. On many platforms, a specific native event may have a number of optional settings. In such cases, the native event and the valid settings are presented, rather than every possible combination of those settings. For each native event, a name, a description, and specific bit patterns are provided.

OPTIONS

This utility has no command line options.

BUGS

There are no known bugs in this utility.
If you find a bug, it should be reported to the PAPI Mailing List at <ptools-perfapi@ptools.org>.

SEE ALSO

PAPI, papi_avail, papi_clockres, papi_command_line, papi_cost, papi_decode, papi_event_chooser, papi_mem_info

NAME

PAPI_read - read hardware counters from an event set
PAPI_accum - accumulate and reset counters in an event set

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_read(int EventSet, long_long *values);
int PAPI_accum(int EventSet, long_long *values);
```

Fortran Interface

```
#include fpapi.h
PAPIF_read(C_INT EventSet, C_LONG_LONG(*) values, C_INT check)
PAPIF_accum(C_INT EventSet, C_LONG_LONG(*) values, C_INT check)
```

DESCRIPTION

These calls assume an initialized PAPI library and a properly added event set.
PAPI_read() copies the counters of the indicated event set into the array *values*. The counters continue counting after the read.
PAPI_accum() adds the counters of the indicated event set into the array *values*. The counters are zeroed and continue counting after the operation.
Note the differences between PAPI_read() and PAPI_accum(), specifically that PAPI_accum() resets the values array to zero.

ARGUMENTS

EventSet -- an integer handle for a PAPI Event Set as created by PAPI_create_eventset
**values* -- an array to hold the counter values of the counting events

RETURN VALUES

On success, these functions return **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ESYS
A system or C library call failed inside PAPI, see the *errno* variable.

PAPI_ENOEVST
The event set specified does not exist.

EXAMPLES

```
do_100events();
if (PAPI_read(EventSet, values) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 100 */
do_100events();
if (PAPI_accum(EventSet, values) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 200 */
values[0] = -100;
do_100events();
if (PAPI_accum(EventSet, values) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 0 */
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_set_opt, PAPI_reset, PAPI_start, PAPI, PAPIF

NAME

PAPI_read_counters - PAPI High Level: read and reset counters
PAPI_accum_counters - PAPI High Level: accumulate and reset counters

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_read_counters(long_long *values, int array_len);
int PAPI_accum_counters(long_long *values, int array_len);
```

Fortran Interface

```
#include fpapi.h
PAPIF_read_counters(C_LONG_LONG(*) values, C_INT array_len, C_INT
check)
PAPIF_accum_counters(C_LONG_LONG(*) values, C_INT array_len, C_INT
check)
```

DESCRIPTION

PAPI_read_counters() copies the event counters into the array *values* .
The counters are reset and left running after the call.
PAPI_accum_counters() adds the event counters into the array *values* .
The counters are reset and left running after the call.
These calls assume an initialized PAPI library and a properly added event set.

ARGUMENTS

**values* -- an array to hold the counter values of the counting events
array_len -- the number of items in the *events array

RETURN VALUES

On success, these functions return **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ESYS

A system or C library call failed inside PAPI, see the *errno* variable.

EXAMPLES

```
do_100events();
if (PAPI_read_counters(values, num_hwcntrs) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 100 */
do_100events();
if (PAPI_accum_counters(values, num_hwcntrs) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 200 */
values[0] = -100;
do_100events();
if (PAPI_accum_counters(values, num_hwcntrs) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 0 */
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_start_counters, PAPI_set_opt, PAPI, PAPIF

NAME

```
PAPI_add_event  - add PAPI preset or native hardware event to an event
set
PAPI_add_events - add PAPI presets or native hardware events to an
event set
```

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_add_event(int EventSet, int EventCode);
int PAPI_add_events(int EventSet, int *EventCodes, int number);
```

Fortran Interface

```
#include fpapi.h
PAPIF_add_event(C_INT EventSet, C_INT EventCode, C_INT check)
PAPIF_add_events(C_INT EventSet, C_INT(*) EventCodes, C_INT number,
C_INT check)
```

DESCRIPTION

```
PAPI_add_event()  adds one event to a PAPI Event Set.
PAPI_add_events() does the same, but for an array of events.
```

A hardware event can be either a PAPI preset or a native hardware event code. For a list of PAPI preset events, see PAPI_presets or run the *avail* test case in the PAPI distribution. PAPI presets can be passed to PAPI_query_event to see if they exist on the underlying architecture. For a list of native events available on current platform, run native_avail test case in the PAPI distribution. For the encoding of native events, see PAPI_event_name_to_code to learn how to generate native code for the supported native event on the underlying architecture.

ARGUMENTS

EventSet -- an integer handle for a PAPI Event Set as created by PAPI_create_eventset
EventCode -- a defined event such as PAPI_TOT_INS.
**EventCode* -- an array of defined events
number -- an integer indicating the number of events in the array **EventCode*
It should be noted that **PAPI_add_events** can partially succeed, exactly like **PAPI_remove_events**.

RETURN VALUES

On success, these functions return **PAPI_OK.**
On error, a less than zero error code is returned or the the number of elements that succeeded before the error.

ERRORS

Positive integer

The number of consecutive elements that succeeded before the error.

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ENOMEM

Insufficient memory to complete the operation.

PAPI_ENOEVST

The event set specified does not exist.

PAPI_EISRUN

The event set is currently counting events.

PAPI_ECNFLCT

The underlying counter hardware can not count this event and other events in the event set simultaneously.

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

PAPI_EBUG

Internal error, please send mail to the developers.

EXAMPLES

```
      int EventSet = PAPI_NULL;
      unsigned int native = 0x0;

      if (PAPI_create_eventset(&EventSet) != PAPI_OK)
        handle_error(1);

/* Add Total Instructions Executed to our EventSet */

if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
  handle_error(1);

/* Add native event PM_CYC to EventSet */

if (PAPI_event_name_to_code("PM_CYC",&native) != PAPI_OK)
  handle_error(1);

if (PAPI_add_event(EventSet, native) != PAPI_OK)
  handle_error(1);
```

BUGS

The vector function should take a pointer to a length argument so a proper return value can be set upon partial success.

SEE ALSO

PAPI_presets, PAPI_native, PAPI_remove_event, PAPI_remove_events, PAPI_query_event, PAPI_cleanup_eventset, PAPI_destroy_eventset, PAPI_event_code_to_name

NAME

```
PAPI_add_event  - add PAPI preset or native hardware event to an event
set
PAPI_add_events - add PAPI presets or native hardware events to an
event set
```

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_add_event(int EventSet, int EventCode);
int PAPI_add_events(int EventSet, int *EventCodes, int number);
```

Fortran Interface

```
#include fpapi.h
PAPIF_add_event(C_INT EventSet, C_INT EventCode, C_INT check)
PAPIF_add_events(C_INT EventSet, C_INT(*) EventCodes, C_INT number,
C_INT check)
```

DESCRIPTION

```
PAPI_add_event()  adds one event to a PAPI Event Set.
PAPI_add_events() does the same, but for an array of events.
```

A hardware event can be either a PAPI preset or a native hardware event code. For a list of PAPI preset events, see PAPI_presets or run the *avail* test case in the PAPI distribution. PAPI presets can be passed to PAPI_query_event to see if they exist on the underlying architecture. For a list of native events available on current platform, run native_avail test case in the PAPI distribution. For the encoding of native events, see PAPI_event_name_to_code to learn how to generate native code for the supported native event on the underlying architecture.

ARGUMENTS

EventSet -- an integer handle for a PAPI Event Set as created by PAPI_create_eventset
EventCode -- a defined event such as PAPI_TOT_INS.
**EventCode* -- an array of defined events
number -- an integer indicating the number of events in the array **EventCode*
It should be noted that **PAPI_add_events** can partially succeed, exactly like **PAPI_remove_events**.

RETURN VALUES

On success, these functions return **PAPI_OK.**
On error, a less than zero error code is returned or the the number of elements that succeeded before the error.

ERRORS

Positive integer

The number of consecutive elements that succeeded before the error.

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ENOMEM

Insufficient memory to complete the operation.

PAPI_ENOEVST

The event set specified does not exist.

PAPI_EISRUN

The event set is currently counting events.

PAPI_ECNFLCT

The underlying counter hardware can not count this event and other events in the event set simultaneously.

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

PAPI_EBUG

Internal error, please send mail to the developers.

EXAMPLES

```
      int EventSet = PAPI_NULL;
      unsigned int native = 0x0;

      if (PAPI_create_eventset(&EventSet) != PAPI_OK)
        handle_error(1);

/* Add Total Instructions Executed to our EventSet */

if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
  handle_error(1);

/* Add native event PM_CYC to EventSet */

if (PAPI_event_name_to_code("PM_CYC",&native) != PAPI_OK)
  handle_error(1);

if (PAPI_add_event(EventSet, native) != PAPI_OK)
  handle_error(1);
```

BUGS

The vector function should take a pointer to a length argument so a proper return value can be set upon partial success.

SEE ALSO

PAPI_presets, PAPI_native, PAPI_remove_event, PAPI_remove_events, PAPI_query_event, PAPI_cleanup_eventset, PAPI_destroy_eventset, PAPI_event_code_to_name

NAME

PAPI_attach - attach PAPI event set to the specified thread id
PAPI_detach - detach PAPI event set from previously specified thread id and restore to executing thread

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_attach(int EventSet, unsigned long tid);
int PAPI_detach(int EventSet);
```

Fortran Interface

```
<none>
```

DESCRIPTION

PAPI_attach() and **PAPI_detach()** are wrapper functions that access PAPI_set_opt() to allow PAPI to monitor performance counts on a thread other than the one currently executing. This is sometimes referred to as third party monitoring. **PAPI_attach()** connects the specified EventSet to the specifed thread; **PAPI_detach()** breaks that connection and restores the EventSet to the original executing thread.

ARGUMENTS

EventSet -- an integer handle for a PAPI Event Set as created by PAPI_create_eventset
tid -- a thread id as obtained from, for example, PAPI_list_threads or PAPI_thread_id.

RETURN VALUES

On success, these functions return **PAPI_OK.** On error, a negative error code is returned.

ERRORS

PAPI_ESBSTR
This feature is unsupported on this substrate.

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOEVST
The event set specified does not exist.

PAPI_EISRUN
The event set is currently counting events.

EXAMPLES

```
      int EventSet = PAPI_NULL;
      unsigned long pid;

pid = fork();
if (pid <= 0)
  exit(1);

if (PAPI_create_eventset(&EventSet) != PAPI_OK)
  exit(1);

/* Add Total Instructions Executed to our EventSet */

if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
  exit(1);

/* Attach this EventSet to the forked process */

if (PAPI_attach(EventSet, pid) != PAPI_OK)
  exit(1);
```

BUGS

There are no known bugs in these functions.

SEE ALSO

PAPI_list_threads, PAPI_thread_id, PAPI_thread_init, PAPI_set_opt

NAME

PAPI_destroy_eventset, PAPI_cleanup_eventset -empty and destroy an EventSet

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_cleanup_eventset(int EventSet);
int PAPI_destroy_eventset(int *EventSet);
```

Fortran Interface

```
#include fpapi.h
PAPIF_cleanup_eventset(C_INT EventSet, C_INT check)
PAPIF_destroy_eventset(C_INT EventSet, C_INT check)
```

DESCRIPTION

PAPI_cleanup_eventset() removes all events from a PAPI event set and turns off profiling and overflow for all events in the eventset. This can not be called if the EventSet is not stopped.
PAPI_destroy_eventset() deallocates the memory associated with an empty PAPI event set.

ARGUMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset
**EventSet* -- a pointer to the integer handle for a PAPI event set as created by PAPI_create_eventset. The value pointed to by EventSet is then set to PAPI_NULL on success.

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid. Attempting to destroy a non-empty event set or passing in a null pointer to be destroyed.

PAPI_ENOEVST
The EventSet specified does not exist.

PAPI_EISRUN
The EventSet is currently counting events.

PAPI_EBUG
Internal error, send mail to ptools-perfapi@ptools.org and complain.

EXAMPLES

```
      /* Remove all events in the eventset */

if (PAPI_cleanup_eventset(EventSet) != PAPI_OK)
  handle_error(1);

/* Free all memory and data structures, EventSet must be empty. */

if (PAPI_destroy_eventset(&EventSet) != PAPI_OK)
  handle_error(1);
```

BUGS

If the user has set profile on an event with the **PAPI_profil** (3) call, then when destroying the EventSet the memory allocated by **PAPI_profil** (3) will not be freed. The user should turn off profiling on the Events before destroying the EventSet to prevent this behavior.

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_stop, PAPI_profil

NAME

PAPI_create_eventset - create an EventSet

SYNOPSIS

C Interface

```
#include <papi.h>
PAPI_create_eventset (int *EventSet);
```

Fortran Interface

```
#include fpapi.h
PAPIF_create_eventset(C_INT EventSet, C_INT check)
```

DESCRIPTION

PAPI_create_eventset() creates a new EventSet pointed to by *EventSet,* which must be initialized to *PAPI_NULL* before calling this routine. The user may then add hardware events to the event set by calling PAPI_add_event or similar routines.

ARGUMENTS

EventSet -- Address of an integer location to store the new EventSet handle

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL

The argument *handle* has not been initialized to *PAPI_NULL* or the argument is a NULL pointer.

PAPI_ENOMEM

Insufficient memory to complete the operation.

EXAMPLES

```
        int EventSet = PAPI_NULL;

  if (PAPI_create_eventset(&EventSet) != PAPI_OK)
    handle_error(1);

  /* Add Total Instructions Executed to our EventSet */
```

```
  if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
    handle_error(1);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_destroy_eventset, PAPI_cleanup_eventset, PAPI_add_event

NAME

PAPI_destroy_eventset, PAPI_cleanup_eventset -empty and destroy an EventSet

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_cleanup_eventset(int EventSet);
int PAPI_destroy_eventset(int *EventSet);
```

Fortran Interface

```
#include fpapi.h
PAPIF_cleanup_eventset(C_INT EventSet, C_INT check)
PAPIF_destroy_eventset(C_INT EventSet, C_INT check)
```

DESCRIPTION

PAPI_cleanup_eventset() removes all events from a PAPI event set and turns off profiling and overflow for all events in the eventset. This can not be called if the EventSet is not stopped.
PAPI_destroy_eventset() deallocates the memory associated with an empty PAPI event set.

ARGUMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset
**EventSet* -- a pointer to the integer handle for a PAPI event set as created by PAPI_create_eventset. The value pointed to by EventSet is then set to PAPI_NULL on success.

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid. Attempting to destroy a non-empty event set or passing in a null pointer to be destroyed.

PAPI_ENOEVST
The EventSet specified does not exist.

PAPI_EISRUN
The EventSet is currently counting events.

PAPI_EBUG
Internal error, send mail to ptools-perfapi@ptools.org and complain.

EXAMPLES

```
      /* Remove all events in the eventset */

if (PAPI_cleanup_eventset(EventSet) != PAPI_OK)
  handle_error(1);

/* Free all memory and data structures, EventSet must be empty. */

if (PAPI_destroy_eventset(&EventSet) != PAPI_OK)
  handle_error(1);
```

BUGS

If the user has set profile on an event with the **PAPI_profil** (3) call, then when destroying the EventSet the memory allocated by **PAPI_profil** (3) will not be freed. The user should turn off profiling on the Events before destroying the EventSet to prevent this behavior.

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_stop, PAPI_profil

NAME

```
PAPI_attach  - attach PAPI event set to the specified thread id
PAPI_detach  - detach PAPI event set from previously specified thread
id and restore to executing thread
```

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_attach(int EventSet, unsigned long tid);
int PAPI_detach(int EventSet);
```

Fortran Interface

```
<none>
```

DESCRIPTION

PAPI_attach() and **PAPI_detach()** are wrapper functions that access PAPI_set_opt() to allow PAPI to monitor performance counts on a thread other than the one currently executing. This is sometimes referred to as third party monitoring. **PAPI_attach()** connects the specified EventSet to the specifed thread; **PAPI_detach()** breaks that connection and restores the EventSet to the original executing thread.

ARGUMENTS

EventSet -- an integer handle for a PAPI Event Set as created by PAPI_create_eventset
tid -- a thread id as obtained from, for example, PAPI_list_threads or PAPI_thread_id.

RETURN VALUES

On success, these functions return **PAPI_OK.** On error, a negative error code is returned.

ERRORS

PAPI_ESBSTR
This feature is unsupported on this substrate.

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOEVST
The event set specified does not exist.

PAPI_EISRUN
The event set is currently counting events.

EXAMPLES

```
      int EventSet = PAPI_NULL;
      unsigned long pid;

pid = fork();
if (pid <= 0)
  exit(1);

if (PAPI_create_eventset(&EventSet) != PAPI_OK)
  exit(1);

/* Add Total Instructions Executed to our EventSet */

if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
  exit(1);

/* Attach this EventSet to the forked process */

if (PAPI_attach(EventSet, pid) != PAPI_OK)
  exit(1);
```

BUGS

There are no known bugs in these functions.

SEE ALSO

PAPI_list_threads, PAPI_thread_id, PAPI_thread_init, PAPI_set_opt

NAME

PAPI_encode_events - read event definitions from a file and modify the existing PAPI preset table.

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_encode_events(char * event_file, int replace);
```

DESCRIPTION

NOTE: This API has been deprecated in PAPI 3.5 pending a data structure redesign.
This function reads event descriptions from a file where they are stored in comma separated value format and modifies or adds events to the PAPI preset event table. The file format is described below. This function presently works only to define or modify PAPI preset events.

FILE FORMAT

The comma separated value file format is one that can be easily edited in a standard text editor or a csv-aware spreadsheet application, and can be easily parsed. Text strings can contain commas, but only if the strings are enclosed in quotes. Each entry in the file is a separate line, and each field, including empty fields, is separated by a comma from its neighbor. The specific format used in this case consists of a title line for readability, a blank line, and a series of lines containing event definitions. A portion of such a file (for Pentium 4) is shown below:

```
name,derived,postfix,short_descr,long_descr,note,[native,...]

PAPI_L1_ICM,NOT_DERIVED,,"L1I cache misses","Level 1 instruction cache
misses",,BPU_fetch_request_TCMISS
PAPI_L2_TCM,NOT_DERIVED,,"L2 cache misses","Level 2 cache
misses",,BSQ_cache_reference_RD_2ndL_MISS_WR_2ndL_MISS
PAPI_TLB_DM,NOT_DERIVED,,"Data TLB misses","Data translation lookaside buffer
misses",,page_walk_type_DTMISS
MY_PAPI_TLB_DM,NOT_DERIVED,,"Data TLB misses","Data translation lookaside buffer
misses","This is a note for my event",page_walk_type_DTMISS
```

ARGUMENTS

event_file -- string containing the name of the csv event file to be read
replace -- 1 to replace existing events, or 0 to prevent accidental replacement

RETURN VALUES

On success, the function returns **PAPI_OK.** On error, a non-zero error code is returned by the function.

ERRORS

PAPI_EPERM

You are trying to modify an existing event without specifying *replace*.

PAPI_EISRUN

You are trying to modify an event that has been added to an EventSet.

PAPI_EINVAL

One or more of the arguments or fields of the info structure is invalid.

PAPI_ENOTPRESET

The PAPI preset table is full and there is no room for a new event.

PAPI_ENOEVNT

The event specified is not a PAPI preset. Usually because the PAPI_PRESET_MASK bit is not set.

EXAMPLE

```
/* Use the command line utility to create a csv copy of the currently
defined events */
> /papi/utils/decode -a -> current.csv
/* View and modify the events in an editor */
> vi current.csv
/* Load the modified events into the preset table */
if (PAPI_encode_events("./current.csv", 1) != PAPI_OK)
  handle_error(1);
```

BUGS

This function has no known bugs.

SEE ALSO

papi_decode , PAPI, PAPIF, PAPI_get_event_info , PAPI_set_event_info

NAME

```
PAPI_enum_event - enumerate PAPI preset or native events
```

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_enum_event(int *EventCode,int modifer);
```

Fortran Interface

```
#include fpapi.h
PAPIF_enum_event(C_INT EventCode, C_INT modifier, C_INT check)
```

DESCRIPTION

Given a preset or native event code, **PAPI_enum_event()** replaces the event code with the next available event in either the preset or native table. The *modifier* argument affects which events are returned. For all platforms and event types, a value of **PAPI_ENUM_ALL** (zero) directs the function to return all possible events. For preset events, a TRUE (non-zero) value currently directs the function to return event codes only for PAPI preset events available on this platform. This may change in the future. For native events, the effect of the *modifier* argument is different on each platform. See the discussion below for platform-specific definitions.

ARGUMENTS

EventCode -- a defined preset or native event such as PAPI_TOT_INS.
modifier -- modifies the search logic. For preset events, TRUE specifies available events only. For native events, each platform behaves differently. See platform-specific documentation for details

PENTIUM 4

The following values are implemented for *modifier* on Pentium 4: **PAPI_PENT4_ENUM_GROUPS** - 45 groups + custom + user event types **PAPI_PENT4_ENUM_COMBOS** - all combinations of mask bits for given group **PAPI_PENT4_ENUM_BITS** - all individual bits for a given group

ITANIUM

The following values are implemented for *modifier* on Itanium: **PAPI_ITA_ENUM_IARR** - Enumerate IAR (instruction address ranging) events **PAPI_ITA_ENUM_DARR** - Enumerate DAR (data address ranging) events **PAPI_ITA_ENUM_OPCM** - Enumerate OPC (opcode matching) events **PAPI_ITA_ENUM_IEAR** - Enumerate IEAR (instr event address register) events **PAPI_ITA_ENUM_DEAR** - Enumerate DEAR (data event address register) events

POWER 4

The following values are implemented for *modifier* on POWER 4: **PAPI_PWR4_ENUM_GROUPS** - Enumerate groups to which an event belongs

RETURN VALUES

On success, this function returns **PAPI_OK ,** and on error, a non-zero error code is returned.

ERRORS

PAPI_ENOEVNT

The next requested PAPI preset or native event is not available on the underlying hardware.

EXAMPLES

```
      /* Scan for all supported native events on this platform */

   printf("Name                          Code        Description0);
   do {
         retval = PAPI_get_event_info(i, &info);
         if (retval == PAPI_OK) {
            printf("%-30s 0x%-10x0s0, info.symbol, info.event_code,
info.long_descr);
         }
   } while (PAPI_enum_event(&i, PAPI_ENUM_ALL) == PAPI_OK);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_preset, PAPI_native, PAPI_get_event_info, PAPI_event_name_to_codePAPI, PAPIF

NAME

PAPI_event_code_to_name- convert a numeric hardware event code to a name.
PAPI_event_name_to_code- convert a name to a numeric hardware event code.

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_event_code_to_name(int EventCode, char *EventName);
int PAPI_event_name_to_code(char *EventName, int *EventCode);
```

Fortran Interface

```
#include fpapi.h
PAPIF_event_code_to_name(C_INT EventCode, C_STRING EventName, C_INT
check)
PAPIF_event_name_to_code(C_STRING EventName, C_INT EventCode, C_INT
check)
```

DESCRIPTION

PAPI_event_code_to_name() is used to translate a 32-bit integer PAPI event code into an ASCII PAPI event name. Either Preset event codes or Native event codes can be passed to this routine. Native event codes and names differ from platform to platform.
PAPI_event_name_to_code() is used to translate an ASCII PAPI event name into an integer PAPI event code.

ARGUMENTS

EventName -- a string containing the event name as listed in PAPI_presets or discussed in PAPI_native
EventCode -- the numeric code for the event

RETURN VALUES

On success, these functions return **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOTPRESET
The hardware event specified is not a valid PAPI preset.

PAPI_ENOEVNT
The hardware event is not available on the underlying hardware.

EXAMPLES

```
      int EventCode, EventSet = PAPI_NULL;
      char EventCodeStr[PAPI_MAX_STR_LEN];
      char EventDescr[PAPI_MAX_STR_LEN];
      char EventLabel[20];

      /* Convert to integer */

if (PAPI_event_name_to_code("PAPI_TOT_INS",&EventCode) != PAPI_OK)
  handle_error(1);

/* Create the EventSet */

if (PAPI_create_eventset(&EventSet) != PAPI_OK)
  handle_error(1);

/* Add Total Instructions Executed to our EventSet */

if (PAPI_add_event(EventSet, EventCode) != PAPI_OK)
  handle_error(1);
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_native, PAPI_enum_events, PAPI_add_event, PAPI_remove_event, PAPI_get_event_info

NAME

PAPI_event_code_to_name- convert a numeric hardware event code to a name.
PAPI_event_name_to_code- convert a name to a numeric hardware event code.

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_event_code_to_name(int EventCode, char *EventName);
int PAPI_event_name_to_code(char *EventName, int *EventCode);
```

Fortran Interface

```
#include fpapi.h
PAPIF_event_code_to_name(C_INT EventCode, C_STRING EventName, C_INT
check)
PAPIF_event_name_to_code(C_STRING EventName, C_INT EventCode, C_INT
check)
```

DESCRIPTION

PAPI_event_code_to_name() is used to translate a 32-bit integer PAPI event code into an ASCII PAPI event name. Either Preset event codes or Native event codes can be passed to this routine. Native event codes and names differ from platform to platform.
PAPI_event_name_to_code() is used to translate an ASCII PAPI event name into an integer PAPI event code.

ARGUMENTS

EventName -- a string containing the event name as listed in PAPI_presets or discussed in PAPI_native
EventCode -- the numeric code for the event

RETURN VALUES

On success, these functions return **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOTPRESET
The hardware event specified is not a valid PAPI preset.

PAPI_ENOEVNT
The hardware event is not available on the underlying hardware.

EXAMPLES

```
      int EventCode, EventSet = PAPI_NULL;
      char EventCodeStr[PAPI_MAX_STR_LEN];
      char EventDescr[PAPI_MAX_STR_LEN];
      char EventLabel[20];

      /* Convert to integer */

if (PAPI_event_name_to_code("PAPI_TOT_INS",&EventCode) != PAPI_OK)
  handle_error(1);

/* Create the EventSet */

if (PAPI_create_eventset(&EventSet) != PAPI_OK)
  handle_error(1);

/* Add Total Instructions Executed to our EventSet */

if (PAPI_add_event(EventSet, EventCode) != PAPI_OK)
  handle_error(1);
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_native, PAPI_enum_events, PAPI_add_event, PAPI_remove_event, PAPI_get_event_info

NAME

PAPI_flips - PAPI High level: Simplified call to get Mflips/s, real and processor time
PAPI_flops - PAPI High level: Simplified call to get Mflops/s, real and processor time

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_flips (float *rtime, float *ptime, long_long *flpins, float
*mflips);
int PAPI_flops (float *rtime, float *ptime, long_long *flpops, float
*mflops);
```

Fortran Interface

```
#include fpapi.h
PAPIF_flips(C_FLOAT real_time, C_FLOAT proc_time, C_LONG_LONG flpins,
C_FLOAT mflips, C_INT check)
PAPIF_flops(C_FLOAT real_time, C_FLOAT proc_time, C_LONG_LONG flpops,
C_FLOAT mflops, C_INT check)
```

DESCRIPTION

The first call to **PAPI_flips() or PAPI_flops()** will initialize the PAPI High Level interface, set up the counters to monitor PAPI_FP_INS or PAPI_FP_OPS and PAPI_TOT_CYC events and start the counters. Subsequent calls will read the counters and return total real time, total process time, total floating point instructions or operations since the start of the measurement and the Mflip/s or Mflop/s rate since latest call to **PAPI_flips() or PAPI_flops().** A call to **PAPI_stop_counters()** will stop the counters from running and then calls such as **PAPI_start_counters()** can safely be used.

ARGUMENTS

**rtime* -- total realtime since the first PAPI_flops() call
**ptime* -- total process time since the first PAPI_flops() call
**flpins, flpops* -- total floating point instructions or operations since the first call
**mflips, *mflops* -- Mflip/s or Mflop/s achieved since the previous call

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

In addition to the possible errors returned by the various PAPI low level calls, the following errors could also be returned:
PAPI_EINVAL

The counters were already started by something other than: **PAPI_flips() or PAPI_flops().**

PAPI_ENOEVNT

The floating point operations,floating point instruction or total cycles event does not exist.

PAPI_ENOMEM

Insufficient memory to complete the operation.

NOTES

Mflip/s, or millions of floating point instructions per second, is defined in this context as the number of instructions issued to the floating point unit per second. It is usually calculated directly from a counter measurement and may be different from platform to platform. Mflop/s, or millions of floating point operations per second, is intended to represent the number of floating point arithmetic operations per second. Attempts are made to massage the counter values to produce the theoreticallly expected value by, for instance, doubling FMA counts or subtracting floating point loads and stores if necessary.*CAVEAT EMPTOR*

`PAPI_flops()` and `PAPI_flips()` may be called by:

```
        the user application program
```

`PAPI_flops()` contains calls to:

```
        PAPI_perror()
        PAPI_library_init()
        PAPI_get_hardware_info()
        PAPI_create_eventset()
        PAPI_add_event()
        PAPI_start()
        PAPI_get_real_usec()
        PAPI_accum()
        PAPI_shutdown()
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_stop_counters , PAPI_ipc , PAPI_set_opt

NAME

PAPI_flips - PAPI High level: Simplified call to get Mflips/s, real and processor time
PAPI_flops - PAPI High level: Simplified call to get Mflops/s, real and processor time

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_flips (float *rtime, float *ptime, long_long *flpins, float
*mflips);
int PAPI_flops (float *rtime, float *ptime, long_long *flpops, float
*mflops);
```

Fortran Interface

```
#include fpapi.h
PAPIF_flips(C_FLOAT real_time, C_FLOAT proc_time, C_LONG_LONG flpins,
C_FLOAT mflips, C_INT check)
PAPIF_flops(C_FLOAT real_time, C_FLOAT proc_time, C_LONG_LONG flpops,
C_FLOAT mflops, C_INT check)
```

DESCRIPTION

The first call to **PAPI_flips() or PAPI_flops()** will initialize the PAPI High Level interface, set up the counters to monitor PAPI_FP_INS or PAPI_FP_OPS and PAPI_TOT_CYC events and start the counters. Subsequent calls will read the counters and return total real time, total process time, total floating point instructions or operations since the start of the measurement and the Mflip/s or Mflop/s rate since latest call to **PAPI_flips() or PAPI_flops().** A call to **PAPI_stop_counters()** will stop the counters from running and then calls such as **PAPI_start_counters()** can safely be used.

ARGUMENTS

**rtime* -- total realtime since the first PAPI_flops() call
**ptime* -- total process time since the first PAPI_flops() call
**flpins, flpops* -- total floating point instructions or operations since the first call
**mflips, *mflops* -- Mflip/s or Mflop/s achieved since the previous call

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

In addition to the possible errors returned by the various PAPI low level calls, the following errors could also be returned:
PAPI_EINVAL

The counters were already started by something other than: **PAPI_flips() or PAPI_flops().**

PAPI_ENOEVNT

The floating point operations,floating point instruction or total cycles event does not exist.

PAPI_ENOMEM

Insufficient memory to complete the operation.

NOTES

Mflip/s, or millions of floating point instructions per second, is defined in this context as the number of instructions issued to the floating point unit per second. It is usually calculated directly from a counter measurement and may be different from platform to platform. Mflop/s, or millions of floating point operations per second, is intended to represent the number of floating point arithmetic operations per second. Attempts are made to massage the counter values to produce the theoreticallly expected value by, for instance, doubling FMA counts or subtracting floating point loads and stores if necessary.*CAVEAT EMPTOR*

`PAPI_flops()` and `PAPI_flips()` may be called by:

```
        the user application program
```

`PAPI_flops()` contains calls to:

```
        PAPI_perror()
        PAPI_library_init()
        PAPI_get_hardware_info()
        PAPI_create_eventset()
        PAPI_add_event()
        PAPI_start()
        PAPI_get_real_usec()
        PAPI_accum()
        PAPI_shutdown()
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_stop_counters , PAPI_ipc , PAPI_set_opt

NAME

PAPI_get_dmem_info - get information about the dynamic memory usage of the current program

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_get_dmem_info(PAPI_dmem_info_t *dmem);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_dmem_info(C_LONG_LONG(*) dmem, C_INT check)
```

DESCRIPTION

In C, this function takes a pointer to a PAPI_dmem_info_t structure and returns with the structure fields filled in. In Fortran, this function takes a pointer to an array of long_long values and fills in the array on return. A value of PAPI_EINVAL in any field indicates an undefined parameter.

NOTE

This function is currently implemented only for the Linux operating system.

ARGUMENTS

dmem -- Structure (C) or array (Fortran) containing the following values (Fortran values can be accessed using the specified indices):
size [PAPIF_DMEM_VMSIZE] (Size of process image),
resident [PAPIF_DMEM_RESIDENT] (Resident set size),
high_water_mark [PAPIF_DMEM_HIGH_WATER] (High water memory usage),
shared [PAPIF_DMEM_SHARED] (Shared memory),
text [PAPIF_DMEM_TEXT] (Memory allocated to code),
library [PAPIF_DMEM_LIBRARY] (Memory allocated to libraries),
heap [PAPIF_DMEM_HEAP] (Size of the heap),
locked [PAPIF_DMEM_LOCKED] (Locked memory),
stack [PAPIF_DMEM_STACK] (Size of the stack)
pagesize [PAPIF_DMEM_PAGESIZE] (Size of a page in bytes),

RETURN VALUES

On success, this function returns PAPI_OK with the data structure or array values filled in. On error a negative error value is returned.

ERRORS

PAPI_ESBSTR

The funtion is not implemented for the current substrate.

PAPI_EINVAL

Any value in the structure or array may be undefined as indicated by this error value.

PAPI_SYS

A system error occured.

EXAMPLE

```
        int retval;
        PAPI_dmem_info_t dmem;

        if (PAPI_library_init(PAPI_VER_CURRENT) != PAPI_VER_CURRENT)
        exit(1);

retval = PAPI_library_init(PAPI_VER_CURRENT);
if (retval != PAPI_VER_CURRENT)
   handle_error(retval);

PAPI_get_dmem_info(&dmem);
printf("Mem Size:            %lld0,dmem.size);
printf("Mem Resident:                %lld0,dmem.resident);
printf("Mem High Water Mark: %lld0,dmem.high_water_mark);
printf("Mem Shared:          %lld0,dmem.shared);
printf("Mem Text:            %lld0,dmem.text);
printf("Mem Library:         %lld0,dmem.library);
printf("Mem Heap:            %lld0,dmem.heap);
printf("Mem Locked:          %lld0,dmem.locked);
printf("Mem Stack:           %lld0,dmem.stack);
printf("Mem Pagesize:                %lld0,dmem.pagesize);
```

BUGS

If called before **PAPI_library_init()** the behavior of the routine is undefined.

SEE ALSO

PAPI_library_init, PAPI_get_opt, PAPI_get_hardware_info, PAPI_get_executable_info

NAME

PAPI_get_event_info - get the event's name and description info

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_get_event_info(int EventCode, PAPI_event_info_t *info);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_event_info(C_INT EventCode, C_STRING symbol,
        C_STRING long_descr, C_STRING C_INT count,
        C_STRING event_note, C_INT , C_INT check)
```

DESCRIPTION

In C, this function fills the event information into a structure. In Fortran, some fields of the structure are returned explicitly. This function works with existing PAPI preset and native event codes.

ARGUMENTS

The following arguments are implicit in the structure returned by the C function, or explicitly returned by Fortran.
EventCode -- event code(preset or native)
info -- structure with the event information
symbol -- whether the preset is part of the API
long_descr -- detail description about the event
short_descr -- short description about the event
event_note -- notes about the event

RETURN VALUES

On success, the C function returns PAPI_OK, and the Fortran function returns **PAPI_OK.**
On error, a non-zero error code is returned by the function.

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ENOTPRESET

The PAPI preset mask was set, but the hardware event specified is not a valid PAPI preset.

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLE

```
/*Find the event code for PAPI_TOT_INS and its info*/
PAPI_event_name_to_code("PAPI_TOT_INS",&EventCode)
if (PAPI_get_event_info(EventCode, &info) == PAPI_OK)
  handle_error(1);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI, PAPIF, PAPI_set_event_info , PAPI_event_name_to_code

NAME

PAPI_get_executable_info -get the executable's address space info

SYNOPSIS

C Interface

```
#include <papi.h>
const PAPI_exe_info_t *PAPI_get_executable_info(void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_exe_info(C_STRING fullname, C_STRING name,
       C_LONG_LONG text_start, C_LONG_LONG text_end,
       C_LONG_LONG data_start, C_LONG_LONG data_end,
       C_LONG_LONG bss_start,  C_LONG_LONG bss_end, C_INT check)
```

DESCRIPTION

In C, this function returns a pointer to a structure containing information about the current program. In Fortran, the fields of the structure are returned explicitly.

ARGUMENTS

The following arguments are implicit in the structure returned by the C function, or explicitly returned by Fortran.
fullname -- fully qualified path + filename of the executable
name -- filename of the executable with no path information
text_start, text_end -- Start and End addresses of program text segment
data_start, data_end -- Start and End addresses of program data segment
bss_start, bss_end -- Start and End addresses of program bss segment

RETURN VALUES

On success, the C function returns a non-NULL pointer, and the Fortran function returns **PAPI_OK.**
On error, NULL is returned by the C function, and a non-zero error code is returned by the Fortran function.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

EXAMPLE

```
const PAPI_exe_info_t *prginfo = NULL;
```

```
      if ((prginfo = PAPI_get_executable_info()) == NULL)
        exit(1);

printf("Path+Program: %s0,exeinfo->fullname);
printf("Program: %s0,exeinfo->address_info.name);
printf("Text start: %p, Text end: %p0,exeinfo->address_info.text_start,exeinfo-
>address_info.text_end);
printf("Data start: %p, Data end: %p0,exeinfo->address_info.data_start,exeinfo-
>address_info.data_end);
printf("Bss start: %p, Bss end: %p0,exeinfo->address_info.bss_start,exeinfo-
>address_info.bss_end);
```

DATA STRUCTURES

```
         typedef struct _papi_address_map {
            char name[PAPI_HUGE_STR_LEN];
            caddr_t text_start;       /* Start address of program text
      segment */
            caddr_t text_end;         /* End address of program text segment
      */
            caddr_t data_start;       /* Start address of program data
      segment */
            caddr_t data_end;         /* End address of program data segment
      */
            caddr_t bss_start;        /* Start address of program bss segment
      */
            caddr_t bss_end;          /* End address of program bss segment */
         } PAPI_address_map_t;

   typedef struct _papi_program_info {
      char fullname[PAPI_HUGE_STR_LEN];  /* path+name */
      PAPI_address_map_t address_info;
   } PAPI_exe_info_t;
```

BUGS

Only the *text_start* and *text_end* fields are filled on every architecture.

SEE ALSO

PAPI_get_hardware_info, PAPI_get_opt

NAME

PAPI_get_hardware_info- get information about the system hardware

SYNOPSIS

C Interface

```
#include <papi.h>
const PAPI_hw_info_t *PAPI_get_hardware_info(void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_hardware_info(C_INT ncpu, C_INT nnodes,
          C_INT  totalcpus,  C_INT  vendor,
          C_STRING  vendor_string,   C_INT  model,
          C_STRING  model_string,
          C_FLOAT  revision,  C_FLOAT  mhz)
```

DESCRIPTION

In C, this function returns a pointer to a structure containing information about the hardware on which the program runs. In Fortran, the values of the structure are returned explicitly.

NOTE

The C structure contains detailed information about cache and TLB sizes. This information is not available from Fortran.

ARGUMENTS

The following arguments are implicit in the structure returned by the C function, or explicitly returned by Fortran.
ncpu -- number of CPUs in an SMP Node
nnodes -- number of Nodes in the entire system
totalcpus -- total number of CPUs in the entire system
vendor -- vendor id number of CPU
vendor_string -- vendor id string of CPU
model -- model number of CPU
model_string -- model string of CPU
revision -- Revision number of CPU
mhz -- Cycle time of this CPU; *may* be an estimate generated at init time with a quick timing routine
mem_hierarchy -- PAPI memory heirarchy description

RETURN VALUES

On success, the C function returns a non-NULL pointer, and the Fortran function returns **PAPI_OK.**
On error, NULL is returned by the C function, and a non-zero error code is returned by the Fortran function.

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

EXAMPLE

```
      const PAPI_hw_info_t *hwinfo = NULL;

      if (PAPI_library_init(PAPI_VER_CURRENT) != PAPI_VER_CURRENT)
        exit(1);

if ((hwinfo = PAPI_get_hardware_info()) == NULL)
  exit(1);

printf("%d CPU's at %f Mhz.\n",hwinfo->totalcpus,hwinfo->mhz);
```

DATA STRUCTURE

The C data structure returned by this function is found in **papi.h and reproduced below:**

```
        typedef struct _papi_mh_tlb_info {
           int type; /* See papi.h for PAPI_MH definitions. */
           int num_entries;
           int associativity;
        } PAPI_mh_tlb_info_t;

   typedef struct _papi_mh_cache_info {
      int type; /* See papi.h for PAPI_MH definitions. */
      int size;
      int line_size;
      int num_lines;
      int associativity;
   } PAPI_mh_cache_info_t;

   typedef struct _papi_mh_level_info {
      PAPI_mh_tlb_info_t   tlb[2];
      PAPI_mh_cache_info_t cache[2];
   } PAPI_mh_level_t;

   typedef struct _papi_mh_info { /* mh for mem hierarchy maybe? */
      int levels;
      PAPI_mh_level_t level[PAPI_MAX_MEM_HIERARCHY_LEVELS];
   } PAPI_mh_info_t;

   typedef struct _papi_hw_info {
      int ncpu;                     /* Number of CPU's in an SMP Node */
      int nnodes;                   /* Number of Nodes in the entire system */
      int totalcpus;                /* Total number of CPU's in the entire system */
      int vendor;                   /* Vendor number of CPU */
      char vendor_string[PAPI_MAX_STR_LEN];     /* Vendor string of CPU */
      int model;                    /* Model number of CPU */
      char model_string[PAPI_MAX_STR_LEN];      /* Model string of CPU */
```

```
    float revision;                /* Revision of CPU */
    float mhz;                     /* Cycle time of this CPU, *may* be estimated at
                                      init time with a quick timing routine */
  PAPI_mh_info_t mem_hierarchy;    /* PAPI memory heirarchy description */
} PAPI_hw_info_t;
```

BUGS

If called before **PAPI_library_init()** the behavior of the routine is undefined.

SEE ALSO

PAPI_library_init, PAPI_get_dmem_info, PAPI_get_opt, PAPI_get_executable_info

NAME

PAPI_get_multiplex - get the multiplexing status of specified event set PAPI_set_multiplex - convert a standard event set to a multiplexed event set

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_get_multiplex(int EventSet);
int PAPI_set_multiplex(int EventSet);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_multiplex(C_INT EventSet, C_INT check)
PAPIF_set_multiplex(C_INT EventSet, C_INT check)
```

DESCRIPTION

PAPI_get_multiplex tests the state of the *PAPI_MULTIPLEXING* flag in the specified event set, returning *TRUE* if a PAPI event set is multiplexed, or *FALSE* if not.
PAPI_set_multiplex converts a standard PAPI event set created by a call to **PAPI_create_eventset()** into an event set capable of handling multiplexed events. This must be done after calling **PAPI_multiplex_init()**, but prior to calling **PAPI_start().** Events can be added to an event set either before or after converting it into a multiplexed set, but the conversion must be done prior to using it as a multiplexed set.

ARGUMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset

RETURN VALUES

PAPI_get_multiplex returns either *TRUE* (positive non-zero) if multiplexing is enabled for this event set, *FALSE* (zero) if multiplexing is not enabled, or *PAPI_ENOEVST* if the specified event set cannot be found.
On success, **PAPI_get_multiplex** returns *PAPI_OK*. On error, a non-zero error code is returned, as described below.

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid, or the EventSet is already multiplexed.

PAPI_ENOEVST

The EventSet specified does not exist.

PAPI_EISRUN

The EventSet is currently counting events.

PAPI_ENOMEM

Insufficient memory to complete the operation.

EXAMPLES

```
  retval = PAPI_get_multiplex(EventSet);
  if (retval > 0) printf("This event set is ready for multiplexing0")
  if (retval == 0) printf("This event set is not enabled for
multiplexing0")
  if (retval < 0) handle_error(retval);

  retval = PAPI_set_multiplex(EventSet);
  if ((retval == PAPI_EINVAL) && (PAPI_get_multiplex(EventSet) > 0))
    printf("This event set already has multiplexing enabled0);
  else if (retval != PAPI_OK) handle_error(retval);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_multiplex_init, PAPI_set_opt, PAPI_create_eventset

NAME

PAPI_get_opt - get PAPI library or event set options
PAPI_set_opt - set PAPI library or event set options
PAPIF_get_clockrate - get the clockrate (Fortran only)
PAPIF_get_domain - get the counting domain (Fortran only)
PAPIF_get_granularity - get the counting granularity (Fortran only)
PAPIF_get_preload - get the library preload setting (Fortran only)

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_get_opt(int option, PAPI_option_t *ptr);
int PAPI_set_opt(int option, PAPI_option_t *ptr);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_clockrate(C_INT clockrate)
PAPIF_get_domain(C_INT EventSet, C_INT domain, C_INT mode, C_INT check)
PAPIF_get_granularity(C_INT EventSet, C_INT granularity, C_INT mode,
C_INT check)
PAPIF_get_preload(C_STRING preload, C_INT check)
```

DESCRIPTION

PAPI_get_opt() and **PAPI_set_opt()** query or change the options of the PAPI library or a specific event set created by PAPI_create_eventset The C interface for these functions passes a pointer to the *PAPI_option_t* structure. Not all options require or return information in this structure, and not all options are implemented for both get and set.
The Fortran interface is a series of calls implementing various subsets of the C interface. Not all options in C are available in Fortran.
NOTE: Some options, such as PAPI_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.
The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described in the table below. The Fortran calls are implementations of specific options.
ptr -- is a pointer to a structure that acts as both an input and output parameter. It is defined in **papi.h** and below.
EventSet -- input; a reference to an EventSetInfo structure
clockrate -- output; cycle time of this CPU in MHz; *may* be an estimate generated at init time with a quick timing routine
domain -- output; execution domain for which events are counted
granularity -- output; execution granularity for which events are counted
mode -- input; determines if domain or granularity are default or for the current event set

preload -- output; environment variable string for preloading libraries

OPTIONS TABLE

Predefined name	Explanation
	General information requests
PI_CLOCKRATE	Get clockrate in MHz.
PI_MAX_CPUS	Get number of CPUs.
PI_MAX_HWCTRS	Get number of counters.
PI_EXEINFO	Get Executable addresses for text/data/bss.
PI_HWINFO	Get information about the hardware.
PI_SHLIBINFO	Get shared library information used by the program.
PI_SUBSTRATEINFO	Get the PAPI features the substrate supports
PI_LIB_VERSION	Get the full PAPI version of the library
PI_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
	Defaults for the global library
PI_DEFDOM	Get/Set default counting domain for newly created event sets.
PI_DEFGRN	Get/Set default counting granularity.
PI_DEBUG	Get/Set the PAPI debug state and the debug handler. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level. The debug handler is available in ptr->debug.handler. For information regarding the behavior of the handler, please see the man page for PAPI_set_debug.
	Multiplexing control
PI_MULTIPLEX	Get/Set options for multiplexing.
PI_MAX_MPX_CTRS	Get maximum number of multiplexing counters.
PI_DEF_MPX_USEC	Get/Set the sampling time slice in microseconds for multiplexing.
	Manipulating individual event sets
PI_ATTACH	Get thread or process id to which event set is attached. Returns TRUE if currently attached. Set event set specified in ptr->ptr->attach.eventset to be attached to thread or process id specified in in ptr->attach.tid
PI_DETACH	Get thread or process id to which event set is attached. Returns TRUE if currently detached. Set event set specified in ptr->ptr->attach.eventset to be detached from any thread or process id.
PI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PI_GRANUL	Get/Set granularity for a single event set. The event set is specified in ptr->granularity.eventset. Not implemented yet.

Platform specific options	
PAPI_DATA_ADDRESS	Set data address range to restrict event counting for event set specified in ptr->addr.eventset. Starting and ending addresses are specified in ptr->addr.start and ptr->addr.end, respectively. If exact addresses cannot be instantiated, offsets are returned in ptr->addr.start_off and ptr->addr.end_off. Currently implemented on Itanium only.
PAPI_INSTR_ADDRESS	Set instruction address range as described above. Itanium only.

The **option_t *** *ptr* structure is defined in **papi.h** and looks something like the following example from the source tree. Users should use the definition in **papi.h** which is in synch with the library used.

```
typedef union {
  PAPI_preload_option_t preload;
  PAPI_debug_option_t debug;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_attach_option_t attach;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_shlib_info_t *shlib_info;
  PAPI_exe_info_t *exe_info;
  PAPI_substrate_info_t *sub_info;
  PAPI_overflow_option_t ovf_info;
  PAPI_addr_range_option_t addr;
} PAPI_option_t;
```

RETURN VALUES

On success, these functions return *PAPI_OK*. On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOEVST
The event set specified does not exist.

PAPI_EISRUN
The event set is currently counting events.

EXAMPLES

```
PAPI_option_t options;

if ((num = PAPI_get_opt(PAPI_MAX_HWCTRS,NULL)) <= 0)
  handle_error();

printf("This machine has %d counters.0,num);
```

```
/* Set the domain of this EventSet
   to counter user and kernel modes for this
   process */

memset(&options,0x0,sizeof(options));
options.domain.eventset = EventSet;
options.domain.domain = PAPI_DOM_ALL;
if (PAPI_set_opt(PAPI_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

BUGS

The granularity functions are not yet implemented. The domain functions are only implemented on some platforms. There are no known bugs in these functions.

SEE ALSO

PAPI_set_debug, PAPI_set_multiplex, PAPI_set_domain

NAME

PAPI_get_overflow_event_index - converts an overflow vector into an array of indexes to overflowing events

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_get_overflow_event_index(int EventSet, long_long
overflow_vector, int *array, int *number);
```

Fortran Interface

```
Not implemented
```

DESCRIPTION

PAPI_get_overflow_event_index decomposes an overflow_vector into an event index array in which the first element corresponds to the least significant set bit in overflow_vector and so on. Based on overflow_vector, the user can only tell which physical counters overflowed. Using this function, the user can map overflowing counters to specific events in the event set. An array is used in this function to support the possibility of multiple simultaneous overflow events.

ARGUMENTS

EventSet -- an integer handle to a PAPI event set as created by PAPI_create_eventset
overflow_vector -- a vector with bits set for each counter that overflowed. This vector is passed by the system to the overflow handler routine.
**array* -- an array of indexes for events in *EventSet*. No more than **number* indexes will be stored into the *array*.
**number* -- On input the variable determines the size of the *array*.
On output the variable contains the number of indexes in the *array*.
Note that if the given **array* is too short to hold all the indexes correspond to the set bits in the overflow_vector the **number* variable will be set to the size of *array*.

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid. This could occur if the *overflow_vector* is empty (zero), if the *array* or *number* pointers are NULL, if the value of *number* is less than one, or if the *EventSet* is empty.

PAPI_ENOEVST

The *EventSet* specified does not exist.

EXAMPLES

Create a user defined overflow handlerroutine that prints diagnostic information about the overflow:

```
void handler(int EventSet, void *address, long_long overflow_vector,
void *context)
{
  int Events[4], number, i;
  int total = 0, retval;

   printf("Overflow #%d0 Handler(%d) Overflow at %p! vector=0x%llx0,
     total, EventSet, address, overflow_vector);
   total++;
   number = 4;
   retval = PAPI_get_overflow_event_index(EventSet,
                overflow_vector, Events, &number);
   if(retval == PAPI_OK)
     for(i=0; i<number; i++) printf("Event index[%d] = %d", i,
Events[i]);
}
```

BUGS

This function may not return all overflowing events if used with software-driven overflow of multiple derived events.

SEE ALSO

PAPI_overflow

NAME

PAPI_get_real_cyc - get real time counter value in clock cycles
PAPI_get_real_usec - get real time counter value in microseconds

SYNOPSIS

C Interface

```
#include <papi.h>
long_long PAPI_get_real_cyc(void);
long_long PAPI_get_real_usec(void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_real_usec(C_LONG_LONG time)
PAPIF_get_real_cyc(C_LONG_LONG real_cyc)
```

DESCRIPTION

Both of these functions return the total real time passed since some arbitrary starting point. The time is returned in clock cycles or microseconds respectively. These calls are equivalent to wall clock time.

ERRORS

These functions always succeed.

EXAMPLE

```
s = PAPI_get_real_cyc();
your_slow_code();
e = PAPI_get_real_cyc();
printf("Wallclock cycles: %lld\n",e-s);
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_virt_cyc, PAPI_get_virt_usec, PAPI, PAPIF

NAME

PAPI_get_real_cyc - get real time counter value in clock cycles
PAPI_get_real_usec - get real time counter value in microseconds

SYNOPSIS

C Interface

```
#include <papi.h>
long_long PAPI_get_real_cyc(void);
long_long PAPI_get_real_usec(void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_real_usec(C_LONG_LONG time)
PAPIF_get_real_cyc(C_LONG_LONG real_cyc)
```

DESCRIPTION

Both of these functions return the total real time passed since some arbitrary starting point. The time is returned in clock cycles or microseconds respectively. These calls are equivalent to wall clock time.

ERRORS

These functions always succeed.

EXAMPLE

```
s = PAPI_get_real_cyc();
your_slow_code();
e = PAPI_get_real_cyc();
printf("Wallclock cycles: %lld\n",e-s);
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_virt_cyc, PAPI_get_virt_usec, PAPI, PAPIF

NAME

PAPI_get_shared_lib_info - get address info about the shared libraries used by the process

SYNOPSIS

C Interface

```
#include <papi.h>
const PAPI_shlib_info_t *PAPI_get_shared_lib_info(void);
```

DESCRIPTION

In C, this function returns a pointer to a structure containing information about the shared library used by the program. There is no Fortran equivalent call.

NOTE

This data will be incorporated into the **PAPI_get_executable_info** call in the future. will be deprecated and should be used with caution.

RETURN VALUES

On success, the function returns a non-NULL pointer.
On error, NULL is returned.

DATA STRUCTURE

```
   typedef struct _papi_address_map {
      char name[PAPI_MAX_STR_LEN];
      caddr_t text_start;       /* Start address of program text
segment */
      caddr_t text_end;         /* End address of program text segment
*/
      caddr_t data_start;       /* Start address of program data
segment */
      caddr_t data_end;         /* End address of program data segment
*/
      caddr_t bss_start;        /* Start address of program bss segment
*/
      caddr_t bss_end;          /* End address of program bss segment */
   } PAPI_address_map_t;

typedef struct _papi_shared_lib_info {
   PAPI_address_map_t *map;
   int count;
} PAPI_shlib_info_t;
```

BUGS

If called before **PAPI_library_init()** the behavior of the routine is undefined.

SEE ALSO

PAPI_library_init, PAPI_get_opt, PAPI_get_dmem_info, PAPI_get_executable_info, PAPI_get_hardware_info

NAME

PAPI_get_substrate_info - get information about the software substrate

SYNOPSIS

C Interface

```
#include <papi.h>
const PAPI_substrate_info_t *PAPI_get_substrate_info(void);
```

Fortran Interface

```
<none>
```

DESCRIPTION

This function returns a pointer to a structure containing detailed information about the software substrate on which the program runs. This includes versioning information, preset and native event information, details on event multiplexing, and more. For full details, see the structure listing below.

RETURN VALUES

On success, the function returns a non-NULL pointer. On error, a NULL pointer is returned.

ERRORS

<none>

EXAMPLE

```
const PAPI_substrate_info_t *sbinfo = NULL;

if (PAPI_library_init(PAPI_VER_CURRENT) != PAPI_VER_CURRENT)
  exit(1);

if ((sbinfo = PAPI_get_substrate_info()) == NULL)
  exit(1);

printf("This substrate supports %d Preset Events and %d Native events.0,
   sbinfo->num_preset_events, sbinfo->num_native_events);
```

DATA STRUCTURE

The C data structure returned by this function is found in **papi.h** and reproduced below.

```
typedef struct _papi_substrate_option {
```

```
    char name[PAPI_MAX_STR_LEN];           /* Name of the substrate
we're using, usually CVS RCS Id */
    char version[PAPI_MIN_STR_LEN];        /* Version of this
substrate, usually CVS Revision */
    char support_version[PAPI_MIN_STR_LEN]; /* Version of the support
library */
    char kernel_version[PAPI_MIN_STR_LEN];  /* Version of the kernel
PMC support driver */
    int num_cntrs;              /* Number of hardware counters the
substrate supports */
    int num_mpx_cntrs;          /* Number of hardware counters the
substrate or PAPI can multiplex supports */
    int num_preset_events;      /* Number of preset events the
substrate supports */
    int num_native_events;      /* Number of native events the
substrate supports */
    int default_domain;         /* The default domain when this
substrate is used */
    int available_domains;      /* Available domains */
    int default_granularity;    /* The default granularity when this
substrate is used */
    int available_granularities; /* Available granularities */
    int multiplex_timer_sig;    /* Signal number used by the
multiplex timer, 0 if not */
    int multiplex_timer_num;    /* Number of the itimer or POSIX 1
timer used by the multiplex timer */
    int multiplex_timer_us;   /* uS between switching of sets */
    int hardware_intr_sig;      /* Signal used by hardware to deliver
PMC events */
    int opcode_match_width;     /* Width of opcode matcher if exists,
0 if not */
    int reserved_ints[4];
    unsigned int hardware_intr:1;         /* hw overflow intr, does
not need to be emulated in software*/
    unsigned int precise_intr:1;          /* Performance interrupts
happen precisely */
    unsigned int posix1b_timers:1;        /* Using POSIX 1b interval
timers (timer_create) instead of setitimer */
    unsigned int kernel_profile:1;        /* Has kernel profiling
support (buffered interrupts or sprofil-like) */
    unsigned int kernel_multiplex:1;      /* In kernel multiplexing */
    unsigned int data_address_range:1;    /* Supports data address
range limiting */
    unsigned int instr_address_range:1;   /* Supports instruction
address range limiting */
    unsigned int fast_counter_read:1;     /* Supports a user level PMC
read instruction */
    unsigned int fast_real_timer:1;       /* Supports a fast real
timer */
    unsigned int fast_virtual_timer:1;    /* Supports a fast virtual
timer */
    unsigned int attach:1;                /* Supports attach */
    unsigned int attach_must_ptrace:1;    /* Attach must first ptrace
and stop the thread/process*/
    unsigned int edge_detect:1;           /* Supports edge detection
on events */
    unsigned int invert:1;                /* Supports invert detection
on events */
    unsigned int profile_ear:1;           /* Supports data/instr/tlb
miss address sampling */
```

```
        unsigned int grouped_cntrs:1;         /* Underlying hardware uses
    counter groups */
        unsigned int reserved_bits:16;
      } PAPI_substrate_info_t;
```

BUGS

If called before **PAPI_library_init()** the behavior of the routine is undefined.

SEE ALSO

PAPI_library_init, PAPI_get_opt, PAPI_get_dmem_info, PAPI_get_hardware_info, PAPI_get_executable_info

NAME

PAPI_get_thr_specific, PAPI_set_thr_specific - Store or retrieve a pointer to a thread specific data structure

SYNOPSIS

```
#include <papi.h>
int PAPI_get_thr_specific(int tag, void **ptr);
int PAPI_set_thr_specific(int tag, void *ptr);
```

DESCRIPTION

In C, PAPI_set_thr_specific will save ptr into an array indexed by tag. PAPI_get_thr_specific will retrieve the pointer from the array with index tag. There are 2 user available locations and tag can be either PAPI_USR1_TLS or PAPI_USR2_TLS. The array mentioned above is managed by PAPI and allocated to each thread which has called PAPI_thread_init. There are no Fortran equivalent functions.

ARGUMENTS

tag -- An identifier, the value of which is either PAPI_USR1_TLS or PAPI_USR2_TLS. This identifier indicates which of several data structures associated with this thread is to be accessed.
ptr -- A pointer to the memory containing the data structure.

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a negative error value is returned.

ERRORS

PAPI_EINVAL

The *tag* argument is out of range.

EXAMPLE

```
      HighLevelInfo *state = NULL;

if (retval = PAPI_thread_init(pthread_self) != PAPI_OK)
   handle_error(retval);

/*
 * Do we have the thread specific data setup yet?
 */
if ((retval = PAPI_get_thr_specific(PAPI_USR1_TLS, (void *) &state))
    != PAPI_OK || state == NULL) {
   state = (HighLevelInfo *) malloc(sizeof(HighLevelInfo));
   if (state == NULL)
```

```
        return (PAPI_ESYS);

    memset(state, 0, sizeof(HighLevelInfo));
    state->EventSet = PAPI_NULL;

    if ((retval = PAPI_create_eventset(&state->EventSet)) != PAPI_OK)
        return (PAPI_ESYS);

    if ((retval=PAPI_set_thr_specific(PAPI_USR1_TLS, state))!=PAPI_OK)
        return (retval);
}
```

BUGS

There are no known bugs in these functions.

SEE ALSO

PAPI_thread_init, .BR PAPI_thread_id(3), PAPI_register_thread

NAME

PAPI_get_virt_cyc - get virtual time counter value in clock cycles
PAPI_get_virt_usec - get virtual time counter values in microseconds

SYNOPSIS

C Interface

```
#include <papi.h>
long_long PAPI_get_virt_cyc(void);
long_long PAPI_get_virt_usec(void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_virt_usec(C_LONG_LONG time)
PAPIF_get_virt_cyc(C_LONG_LONG virt_cyc)
```

DESCRIPTION

Both of these functions return the total number of virtual units from some arbitrary starting point. Virtual units accrue every time the process is running in user-mode on behalf of the process. Like the real time counters, these are guaranteed to exist on every platform PAPI supports. However on some platforms, the resolution can be as bad as 1/Hz as defined by the operating system.

ERRORS

The functions returns **PAPI_ECNFLCT** if there is no master event set. This will happen if the library has not been initialized, or for threaded applications, if there has been no thread id function defined by the **PAPI_thread_init** function.
For threaded applications, if there has not yet been any thread specific master event created for the current thread, and if the allocation of such an event set fails, the call will return **PAPI_ENOMEM** or **PAPI_ESYS**.

EXAMPLE

```
s = PAPI_get_virt_cyc();
your_slow_code();
e = PAPI_get_virt_cyc();
printf("Process has run for cycles: %lld\n",e-s);
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_real_usec, PAPI_get_real_cyc, PAPI, PAPIF

NAME

PAPI_get_virt_cyc - get virtual time counter value in clock cycles
PAPI_get_virt_usec - get virtual time counter values in microseconds

SYNOPSIS

C Interface

```
#include <papi.h>
long_long PAPI_get_virt_cyc(void);
long_long PAPI_get_virt_usec(void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_virt_usec(C_LONG_LONG time)
PAPIF_get_virt_cyc(C_LONG_LONG virt_cyc)
```

DESCRIPTION

Both of these functions return the total number of virtual units from some arbitrary starting point. Virtual units accrue every time the process is running in user-mode on behalf of the process. Like the real time counters, these are guaranteed to exist on every platform PAPI supports. However on some platforms, the resolution can be as bad as 1/Hz as defined by the operating system.

ERRORS

The functions returns **PAPI_ECNFLCT** if there is no master event set. This will happen if the library has not been initialized, or for threaded applications, if there has been no thread id function defined by the **PAPI_thread_init** function.
For threaded applications, if there has not yet been any thread specific master event created for the current thread, and if the allocation of such an event set fails, the call will return **PAPI_ENOMEM** or **PAPI_ESYS**.

EXAMPLE

```
s = PAPI_get_virt_cyc();
your_slow_code();
e = PAPI_get_virt_cyc();
printf("Process has run for cycles: %lld\n",e-s);
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_real_usec, PAPI_get_real_cyc, PAPI, PAPIF

NAME

PAPI_get_opt - get PAPI library or event set options
PAPI_set_opt - set PAPI library or event set options
PAPIF_get_clockrate - get the clockrate (Fortran only)
PAPIF_get_domain - get the counting domain (Fortran only)
PAPIF_get_granularity - get the counting granularity (Fortran only)
PAPIF_get_preload - get the library preload setting (Fortran only)

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_get_opt(int option, PAPI_option_t *ptr);
int PAPI_set_opt(int option, PAPI_option_t *ptr);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_clockrate(C_INT clockrate)
PAPIF_get_domain(C_INT EventSet, C_INT domain, C_INT mode, C_INT check)
PAPIF_get_granularity(C_INT EventSet, C_INT granularity, C_INT mode,
C_INT check)
PAPIF_get_preload(C_STRING preload, C_INT check)
```

DESCRIPTION

PAPI_get_opt() and **PAPI_set_opt()** query or change the options of the PAPI library or a specific event set created by PAPI_create_eventset The C interface for these functions passes a pointer to the *PAPI_option_t* structure. Not all options require or return information in this structure, and not all options are implemented for both get and set.
The Fortran interface is a series of calls implementing various subsets of the C interface. Not all options in C are available in Fortran.
NOTE: Some options, such as PAPI_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.
The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described in the table below. The Fortran calls are implementations of specific options.
ptr -- is a pointer to a structure that acts as both an input and output parameter. It is defined in **papi.h** and below.
EventSet -- input; a reference to an EventSetInfo structure
clockrate -- output; cycle time of this CPU in MHz; *may* be an estimate generated at init time with a quick timing routine
domain -- output; execution domain for which events are counted
granularity -- output; execution granularity for which events are counted
mode -- input; determines if domain or granularity are default or for the current event set

preload -- output; environment variable string for preloading libraries

OPTIONS TABLE

Predefined name	Explanation
General information requests	
PAPI_CLOCKRATE	Get clockrate in MHz.
PAPI_MAX_CPUS	Get number of CPUs.
PAPI_MAX_HWCTRS	Get number of counters.
PAPI_EXEINFO	Get Executable addresses for text/data/bss.
PAPI_HWINFO	Get information about the hardware.
PAPI_SHLIBINFO	Get shared library information used by the program.
PAPI_SUBSTRATEINFO	Get the PAPI features the substrate supports
PAPI_LIB_VERSION	Get the full PAPI version of the library
PAPI_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
Defaults for the global library	
PAPI_DEFDOM	Get/Set default counting domain for newly created event sets.
PAPI_DEFGRN	Get/Set default counting granularity.
PAPI_DEBUG	Get/Set the PAPI debug state and the debug handler. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level. The debug handler is available in ptr->debug.handler. For information regarding the behavior of the handler, please see the man page for PAPI_set_debug.
Multiplexing control	
PAPI_MULTIPLEX	Get/Set options for multiplexing.
PAPI_MAX_MPX_CTRS	Get maximum number of multiplexing counters.
PAPI_DEF_MPX_USEC	Get/Set the sampling time slice in microseconds for multiplexing.
Manipulating individual event sets	
PAPI_ATTACH	Get thread or process id to which event set is attached. Returns TRUE if currently attached. Set event set specified in ptr->ptr->attach.eventset to be attached to thread or process id specified in in ptr->attach.tid
PAPI_DETACH	Get thread or process id to which event set is attached. Returns TRUE if currently detached. Set event set specified in ptr->ptr->attach.eventset to be detached from any thread or process id.
PAPI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_GRANUL	Get/Set granularity for a single event set. The event set is specified in

	ptr->granularity.eventset. Not implementedyet.
Platform specific options	
PAPI_DATA_ADDRESS	Set data address range to restrict event counting for event set specified in ptr->addr.eventset. Starting and ending addresses are specified in ptr->addr.start and ptr->addr.end, respectively. If exact addresses cannot be instantiated, offsets are returned in ptr->addr.start_off and ptr->addr.end_off. Currently implemented on Itanium only.
PAPI_INSTR_ADDRESS	Set instruction address range as described above. Itanium only.

The **option_t *_ptr_** structure is defined in **papi.h** and looks something like the following example from the source tree. Users should use the definition in **papi.h** which is in synch with the library used.

```
typedef union {
  PAPI_preload_option_t preload;
  PAPI_debug_option_t debug;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_attach_option_t attach;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_shlib_info_t *shlib_info;
  PAPI_exe_info_t *exe_info;
  PAPI_substrate_info_t *sub_info;
  PAPI_overflow_option_t ovf_info;
  PAPI_addr_range_option_t addr;
} PAPI_option_t;
```

RETURN VALUES

On success, these functions return *PAPI_OK*. On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOEVST
The event set specified does not exist.

PAPI_EISRUN
The event set is currently counting events.

EXAMPLES

```
PAPI_option_t options;

if ((num = PAPI_get_opt(PAPI_MAX_HWCTRS,NULL)) <= 0)
  handle_error();
```

```
printf("This machine has %d counters.0,num);

/* Set the domain of this EventSet
   to counter user and kernel modes for this
   process */

memset(&options,0x0,sizeof(options));
options.domain.eventset = EventSet;
options.domain.domain = PAPI_DOM_ALL;
if (PAPI_set_opt(PAPI_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

BUGS

The granularity functions are not yet implemented. The domain functions are only implemented on some platforms. There are no known bugs in these functions.

SEE ALSO

PAPI_set_debug, PAPI_set_multiplex, PAPI_set_domain

NAME

PAPI_get_opt - get PAPI library or event set options
PAPI_set_opt - set PAPI library or event set options
PAPIF_get_clockrate - get the clockrate (Fortran only)
PAPIF_get_domain - get the counting domain (Fortran only)
PAPIF_get_granularity - get the counting granularity (Fortran only)
PAPIF_get_preload - get the library preload setting (Fortran only)

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_get_opt(int option, PAPI_option_t *ptr);
int PAPI_set_opt(int option, PAPI_option_t *ptr);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_clockrate(C_INT clockrate)
PAPIF_get_domain(C_INT EventSet, C_INT domain, C_INT mode, C_INT check)
PAPIF_get_granularity(C_INT EventSet, C_INT granularity, C_INT mode,
C_INT check)
PAPIF_get_preload(C_STRING preload, C_INT check)
```

DESCRIPTION

PAPI_get_opt() and **PAPI_set_opt()** query or change the options of the PAPI library or a specific event set created by PAPI_create_eventset The C interface for these functions passes a pointer to the *PAPI_option_t* structure. Not all options require or return information in this structure, and not all options are implemented for both get and set.
The Fortran interface is a series of calls implementing various subsets of the C interface. Not all options in C are available in Fortran.
NOTE: Some options, such as PAPI_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.
The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described in the table below. The Fortran calls are implementations of specific options.
ptr -- is a pointer to a structure that acts as both an input and output parameter. It is defined in **papi.h** and below.
EventSet -- input; a reference to an EventSetInfo structure
clockrate -- output; cycle time of this CPU in MHz; *may* be an estimate generated at init time with a quick timing routine
domain -- output; execution domain for which events are counted
granularity -- output; execution granularity for which events are counted
mode -- input; determines if domain or granularity are default or for the current event set

preload -- output; environment variable string for preloading libraries

OPTIONS TABLE

Predefined name	Explanation
General information requests	
PAPI_CLOCKRATE	Get clockrate in MHz.
PAPI_MAX_CPUS	Get number of CPUs.
PAPI_MAX_HWCTRS	Get number of counters.
PAPI_EXEINFO	Get Executable addresses for text/data/bss.
PAPI_HWINFO	Get information about the hardware.
PAPI_SHLIBINFO	Get shared library information used by the program.
PAPI_SUBSTRATEINFO	Get the PAPI features the substrate supports
PAPI_LIB_VERSION	Get the full PAPI version of the library
PAPI_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
Defaults for the global library	
PAPI_DEFDOM	Get/Set default counting domain for newly created event sets.
PAPI_DEFGRN	Get/Set default counting granularity.
PAPI_DEBUG	Get/Set the PAPI debug state and the debug handler. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level. The debug handler is available in ptr->debug.handler. For information regarding the behavior of the handler, please see the man page for PAPI_set_debug.
Multiplexing control	
PAPI_MULTIPLEX	Get/Set options for multiplexing.
PAPI_MAX_MPX_CTRS	Get maximum number of multiplexing counters.
PAPI_DEF_MPX_USEC	Get/Set the sampling time slice in microseconds for multiplexing.
Manipulating individual event sets	
PAPI_ATTACH	Get thread or process id to which event set is attached. Returns TRUE if currently attached. Set event set specified in ptr->ptr->attach.eventset to be attached to thread or process id specified in in ptr->attach.tid
PAPI_DETACH	Get thread or process id to which event set is attached. Returns TRUE if currently detached. Set event set specified in ptr->ptr->attach.eventset to be detached from any thread or process id.
PAPI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_GRANUL	Get/Set granularity for a single event set. The event set is specified in

	ptr->granularity.eventset. Not implementedyet.
Platform specific options	
PAPI_DATA_ADDRESS	Set data address range to restrict event counting for event set specified in ptr->addr.eventset. Starting and ending addresses are specified in ptr->addr.start and ptr->addr.end, respectively. If exact addresses cannot be instantiated, offsets are returned in ptr->addr.start_off and ptr->addr.end_off. Currently implemented on Itanium only.
PAPI_INSTR_ADDRESS	Set instruction address range as described above. Itanium only.

The **option_t *_ptr_** structure is defined in **papi.h** and looks something like the following example from the source tree. Users should use the definition in **papi.h** which is in synch with the library used.

```
typedef union {
  PAPI_preload_option_t preload;
  PAPI_debug_option_t debug;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_attach_option_t attach;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_shlib_info_t *shlib_info;
  PAPI_exe_info_t *exe_info;
  PAPI_substrate_info_t *sub_info;
  PAPI_overflow_option_t ovf_info;
  PAPI_addr_range_option_t addr;
} PAPI_option_t;
```

RETURN VALUES

On success, these functions return *PAPI_OK*. On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOEVST
The event set specified does not exist.

PAPI_EISRUN
The event set is currently counting events.

EXAMPLES

```
PAPI_option_t options;

if ((num = PAPI_get_opt(PAPI_MAX_HWCTRS,NULL)) <= 0)
  handle_error();
```

```
printf("This machine has %d counters.0,num);

/* Set the domain of this EventSet
   to counter user and kernel modes for this
   process */

memset(&options,0x0,sizeof(options));
options.domain.eventset = EventSet;
options.domain.domain = PAPI_DOM_ALL;
if (PAPI_set_opt(PAPI_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

BUGS

The granularity functions are not yet implemented. The domain functions are only implemented on some platforms. There are no known bugs in these functions.

SEE ALSO

PAPI_set_debug, PAPI_set_multiplex, PAPI_set_domain

NAME

PAPI_get_executable_info -get the executable's address space info

SYNOPSIS

C Interface

```
#include <papi.h>
const PAPI_exe_info_t *PAPI_get_executable_info(void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_exe_info(C_STRING fullname, C_STRING name,
       C_LONG_LONG text_start, C_LONG_LONG text_end,
       C_LONG_LONG data_start, C_LONG_LONG data_end,
       C_LONG_LONG bss_start,  C_LONG_LONG bss_end, C_INT check)
```

DESCRIPTION

In C, this function returns a pointer to a structure containing information about the current program. In Fortran, the fields of the structure are returned explicitly.

ARGUMENTS

The following arguments are implicit in the structure returned by the C function, or explicitly returned by Fortran.
fullname -- fully qualified path + filename of the executable
name -- filename of the executable with no path information
text_start, text_end -- Start and End addresses of program text segment
data_start, data_end -- Start and End addresses of program data segment
bss_start, bss_end -- Start and End addresses of program bss segment

RETURN VALUES

On success, the C function returns a non-NULL pointer, and the Fortran function returns **PAPI_OK.**
On error, NULL is returned by the C function, and a non-zero error code is returned by the Fortran function.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

EXAMPLE

```
const PAPI_exe_info_t *prginfo = NULL;
```

```
      if ((prginfo = PAPI_get_executable_info()) == NULL)
        exit(1);

printf("Path+Program: %s0,exeinfo->fullname);
printf("Program: %s0,exeinfo->address_info.name);
printf("Text start: %p, Text end: %p0,exeinfo->address_info.text_start,exeinfo-
>address_info.text_end);
printf("Data start: %p, Data end: %p0,exeinfo->address_info.data_start,exeinfo-
>address_info.data_end);
printf("Bss start: %p, Bss end: %p0,exeinfo->address_info.bss_start,exeinfo-
>address_info.bss_end);
```

DATA STRUCTURES

```
        typedef struct _papi_address_map {
           char name[PAPI_HUGE_STR_LEN];
           caddr_t text_start;       /* Start address of program text
     segment */
           caddr_t text_end;         /* End address of program text segment
     */
           caddr_t data_start;       /* Start address of program data
     segment */
           caddr_t data_end;         /* End address of program data segment
     */
           caddr_t bss_start;        /* Start address of program bss segment
     */
           caddr_t bss_end;          /* End address of program bss segment */
        } PAPI_address_map_t;

   typedef struct _papi_program_info {
      char fullname[PAPI_HUGE_STR_LEN];  /* path+name */
      PAPI_address_map_t address_info;
   } PAPI_exe_info_t;
```

BUGS

Only the *text_start* and *text_end* fields are filled on every architecture.

SEE ALSO

PAPI_get_hardware_info, PAPI_get_opt

NAME

PAPI_get_opt - get PAPI library or event set options
PAPI_set_opt - set PAPI library or event set options
PAPIF_get_clockrate - get the clockrate (Fortran only)
PAPIF_get_domain - get the counting domain (Fortran only)
PAPIF_get_granularity - get the counting granularity (Fortran only)
PAPIF_get_preload - get the library preload setting (Fortran only)

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_get_opt(int option, PAPI_option_t *ptr);
int PAPI_set_opt(int option, PAPI_option_t *ptr);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_clockrate(C_INT clockrate)
PAPIF_get_domain(C_INT EventSet, C_INT domain, C_INT mode, C_INT check)
PAPIF_get_granularity(C_INT EventSet, C_INT granularity, C_INT mode,
C_INT check)
PAPIF_get_preload(C_STRING preload, C_INT check)
```

DESCRIPTION

PAPI_get_opt() and **PAPI_set_opt()** query or change the options of the PAPI library or a specific event set created by PAPI_create_eventset The C interface for these functions passes a pointer to the *PAPI_option_t* structure. Not all options require or return information in this structure, and not all options are implemented for both get and set.
The Fortran interface is a series of calls implementing various subsets of the C interface. Not all options in C are available in Fortran.
NOTE: Some options, such as PAPI_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.
The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described in the table below. The Fortran calls are implementations of specific options.
ptr -- is a pointer to a structure that acts as both an input and output parameter. It is defined in **papi.h** and below.
EventSet -- input; a reference to an EventSetInfo structure
clockrate -- output; cycle time of this CPU in MHz; *may* be an estimate generated at init time with a quick timing routine
domain -- output; execution domain for which events are counted
granularity -- output; execution granularity for which events are counted
mode -- input; determines if domain or granularity are default or for the current event set

preload -- output; environment variable string for preloading libraries

OPTIONS TABLE

Predefined name	Explanation
General information requests	
PAPI_CLOCKRATE	Get clockrate in MHz.
PAPI_MAX_CPUS	Get number of CPUs.
PAPI_MAX_HWCTRS	Get number of counters.
PAPI_EXEINFO	Get Executable addresses for text/data/bss.
PAPI_HWINFO	Get information about the hardware.
PAPI_SHLIBINFO	Get shared library information used by the program.
PAPI_SUBSTRATEINFO	Get the PAPI features the substrate supports
PAPI_LIB_VERSION	Get the full PAPI version of the library
PAPI_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
Defaults for the global library	
PAPI_DEFDOM	Get/Set default counting domain for newly created event sets.
PAPI_DEFGRN	Get/Set default counting granularity.
PAPI_DEBUG	Get/Set the PAPI debug state and the debug handler. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level. The debug handler is available in ptr->debug.handler. For information regarding the behavior of the handler, please see the man page for PAPI_set_debug.
Multiplexing control	
PAPI_MULTIPLEX	Get/Set options for multiplexing.
PAPI_MAX_MPX_CTRS	Get maximum number of multiplexing counters.
PAPI_DEF_MPX_USEC	Get/Set the sampling time slice in microseconds for multiplexing.
Manipulating individual event sets	
PAPI_ATTACH	Get thread or process id to which event set is attached. Returns TRUE if currently attached. Set event set specified in ptr->ptr->attach.eventset to be attached to thread or process id specified in in ptr->attach.tid
PAPI_DETACH	Get thread or process id to which event set is attached. Returns TRUE if currently detached. Set event set specified in ptr->ptr->attach.eventset to be detached from any thread or process id.
PAPI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_GRANUL	Get/Set granularity for a single event set. The event set is specified in

	ptr->granularity.eventset. Not implementedyet.
Platform specific options	
PAPI_DATA_ADDRESS	Set data address range to restrict event counting for event set specified in ptr->addr.eventset. Starting and ending addresses are specified in ptr->addr.start and ptr->addr.end, respectively. If exact addresses cannot be instantiated, offsets are returned in ptr->addr.start_off and ptr->addr.end_off. Currently implemented on Itanium only.
PAPI_INSTR_ADDRESS	Set instruction address range as described above. Itanium only.

The **option_t *** *ptr* structure is defined in **papi.h** and looks something like the following example from the source tree. Users should use the definition in **papi.h** which is in synch with the library used.

```
typedef union {
  PAPI_preload_option_t preload;
  PAPI_debug_option_t debug;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_attach_option_t attach;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_shlib_info_t *shlib_info;
  PAPI_exe_info_t *exe_info;
  PAPI_substrate_info_t *sub_info;
  PAPI_overflow_option_t ovf_info;
  PAPI_addr_range_option_t addr;
} PAPI_option_t;
```

RETURN VALUES

On success, these functions return *PAPI_OK*. On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ENOEVST

The event set specified does not exist.

PAPI_EISRUN

The event set is currently counting events.

EXAMPLES

```
PAPI_option_t options;

if ((num = PAPI_get_opt(PAPI_MAX_HWCTRS,NULL)) <= 0)
  handle_error();
```

```
printf("This machine has %d counters.0,num);

/* Set the domain of this EventSet
   to counter user and kernel modes for this
   process */

memset(&options,0x0,sizeof(options));
options.domain.eventset = EventSet;
options.domain.domain = PAPI_DOM_ALL;
if (PAPI_set_opt(PAPI_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

BUGS

The granularity functions are not yet implemented. The domain functions are only implemented on some platforms. There are no known bugs in these functions.

SEE ALSO

PAPI_set_debug, PAPI_set_multiplex, PAPI_set_domain

NAME

PAPI_get_opt - get PAPI library or event set options
PAPI_set_opt - set PAPI library or event set options
PAPIF_get_clockrate - get the clockrate (Fortran only)
PAPIF_get_domain - get the counting domain (Fortran only)
PAPIF_get_granularity - get the counting granularity (Fortran only)
PAPIF_get_preload - get the library preload setting (Fortran only)

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_get_opt(int option, PAPI_option_t *ptr);
int PAPI_set_opt(int option, PAPI_option_t *ptr);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_clockrate(C_INT clockrate)
PAPIF_get_domain(C_INT EventSet, C_INT domain, C_INT mode, C_INT check)
PAPIF_get_granularity(C_INT EventSet, C_INT granularity, C_INT mode,
C_INT check)
PAPIF_get_preload(C_STRING preload, C_INT check)
```

DESCRIPTION

PAPI_get_opt() and **PAPI_set_opt()** query or change the options of the PAPI library or a specific event set created by PAPI_create_eventset The C interface for these functions passes a pointer to the *PAPI_option_t* structure. Not all options require or return information in this structure, and not all options are implemented for both get and set.
The Fortran interface is a series of calls implementing various subsets of the C interface. Not all options in C are available in Fortran.
NOTE: Some options, such as PAPI_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.
The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described in the table below. The Fortran calls are implementations of specific options.
ptr -- is a pointer to a structure that acts as both an input and output parameter. It is defined in **papi.h** and below.
EventSet -- input; a reference to an EventSetInfo structure
clockrate -- output; cycle time of this CPU in MHz; *may* be an estimate generated at init time with a quick timing routine
domain -- output; execution domain for which events are counted
granularity -- output; execution granularity for which events are counted
mode -- input; determines if domain or granularity are default or for the current event set

preload -- output; environment variable string for preloading libraries

OPTIONS TABLE

Predefined name	Explanation
General information requests	
PAPI_CLOCKRATE	Get clockrate in MHz.
PAPI_MAX_CPUS	Get number of CPUs.
PAPI_MAX_HWCTRS	Get number of counters.
PAPI_EXEINFO	Get Executable addresses for text/data/bss.
PAPI_HWINFO	Get information about the hardware.
PAPI_SHLIBINFO	Get shared library information used by the program.
PAPI_SUBSTRATEINFO	Get the PAPI features the substrate supports
PAPI_LIB_VERSION	Get the full PAPI version of the library
PAPI_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
Defaults for the global library	
PAPI_DEFDOM	Get/Set default counting domain for newly created event sets.
PAPI_DEFGRN	Get/Set default counting granularity.
PAPI_DEBUG	Get/Set the PAPI debug state and the debug handler. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level. The debug handler is available in ptr->debug.handler. For information regarding the behavior of the handler, please see the man page for PAPI_set_debug.
Multiplexing control	
PAPI_MULTIPLEX	Get/Set options for multiplexing.
PAPI_MAX_MPX_CTRS	Get maximum number of multiplexing counters.
PAPI_DEF_MPX_USEC	Get/Set the sampling time slice in microseconds for multiplexing.
Manipulating individual event sets	
PAPI_ATTACH	Get thread or process id to which event set is attached. Returns TRUE if currently attached. Set event set specified in ptr->ptr->attach.eventset to be attached to thread or process id specified in in ptr->attach.tid
PAPI_DETACH	Get thread or process id to which event set is attached. Returns TRUE if currently detached. Set event set specified in ptr->ptr->attach.eventset to be detached from any thread or process id.
PAPI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_GRANUL	Get/Set granularity for a single event set. The event set is specified in

	ptr->granularity.eventset. Not implementedyet.
Platform specific options	
PAPI_DATA_ADDRESS	Set data address range to restrict event counting for event set specified in ptr->addr.eventset. Starting and ending addresses are specified in ptr->addr.start and ptr->addr.end, respectively. If exact addresses cannot be instantiated, offsets are returned in ptr->addr.start_off and ptr->addr.end_off. Currently implemented on Itanium only.
PAPI_INSTR_ADDRESS	Set instruction address range as described above. Itanium only.

The **option_t *_ptr_** structure is defined in **papi.h** and looks something like the following example from the source tree. Users should use the definition in **papi.h** which is in synch with the library used.

```
typedef union {
  PAPI_preload_option_t preload;
  PAPI_debug_option_t debug;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_attach_option_t attach;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_shlib_info_t *shlib_info;
  PAPI_exe_info_t *exe_info;
  PAPI_substrate_info_t *sub_info;
  PAPI_overflow_option_t ovf_info;
  PAPI_addr_range_option_t addr;
} PAPI_option_t;
```

RETURN VALUES

On success, these functions return *PAPI_OK*. On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOEVST
The event set specified does not exist.

PAPI_EISRUN
The event set is currently counting events.

EXAMPLES

```
PAPI_option_t options;

if ((num = PAPI_get_opt(PAPI_MAX_HWCTRS,NULL)) <= 0)
  handle_error();
```

```
printf("This machine has %d counters.0,num);

/* Set the domain of this EventSet
   to counter user and kernel modes for this
   process */

memset(&options,0x0,sizeof(options));
options.domain.eventset = EventSet;
options.domain.domain = PAPI_DOM_ALL;
if (PAPI_set_opt(PAPI_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

BUGS

The granularity functions are not yet implemented. The domain functions are only implemented on some platforms. There are no known bugs in these functions.

SEE ALSO

PAPI_set_debug, PAPI_set_multiplex, PAPI_set_domain

NAME

PAPI_ipc - PAPI High level: Simplified call to get instructions per cycle, real and processor time

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_ipc (float *rtime, float *ptime, long_long *ins, float *ipc);
```

Fortran Interface

```
#include fpapi.h
PAPIF_ipc(C_FLOAT real_time, C_FLOAT proc_time, C_LONG_LONG ins,
C_FLOAT ipc, C_INT check)
```

DESCRIPTION

The first call to **PAPI_ipc()** will initialize the PAPI High Level interface, set up the counters to monitor PAPI_TOT_INS and PAPI_TOT_CYC events and start the counters. Subsequent calls will read the counters and return total real time, total process time, total instructions since the start of the measurement and the instructions per cycle rate since latest call to **PAPI_ipc().** A call to **PAPI_stop_counters()** will stop the counters from running and then calls such as **PAPI_start_counters()** can safely be used.

ARGUMENTS

**rtime* -- total realtime since the first PAPI_ipc() call
**ptime* -- total process time since the first PAPI_ipc() call
**ins* -- total instructions since the first call
**ipc* -- instructions per cycle achieved since the previous call

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

In addition to the possible errors returned by the various PAPI low level calls, the following errors could also be returned:

PAPI_EINVAL

The counters were already started by something other than: **PAPI_ipc()**

PAPI_ENOEVNT

The total instructions or total cycles event does not exist.

PAPI_ENOMEM

Insufficient memory to complete the operation.

PAPI_ipc() may be called by: **the user application program**

PAPI_ipc() contains calls to:

```
PAPI_perror()
PAPI_library_init()
PAPI_get_hardware_info()
PAPI_create_eventset()
PAPI_add_event()
PAPI_start()
PAPI_get_real_usec()
PAPI_accum()
PAPI_shutdown()
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_stop_counters , PAPI_set_opt , PAPI_flips , PAPI_flops

NAME

```
PAPI_library_init   - initialize the PAPI library.
PAPI_is_initialized - check for initialization.
```

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_library_init(int version);
int PAPI_is_initialized(void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_library_init(C_INT check)
PAPIF_is_initialized(C_INT check)
```

DESCRIPTION

PAPI_library_init() initializes the PAPI library. It must be called before any low level PAPI functions can be used. If your application is making use of threads PAPI_thread_init must also be called prior to making any calls to the library other than **PAPI_library_init()**.
PAPI_is_initialized() returns the status of the PAPI library. The PAPI library can be in one of three states, as described under RETURN VALUES.

ARGUMENTS

version -- upon initialization, PAPI checks the argument against the internal value of **PAPI_VER_CURRENT** when the library was compiled. This guards against portability problems when updating the PAPI shared libraries on your system.

RETURN VALUES

PAPI_library_init : On success, this function returns **PAPI_VER_CURRENT.** A positive return code other than **PAPI_VER_CURRENT** indicates a library version mis-match. A negative error code indicates an initialization error.
PAPI_is_initialized :
PAPI_NOT_INITED
-- PAPI has not been initialized
PAPI_LOW_LEVEL_INITED
-- PAPI_library_init has been called
PAPI_HIGH_LEVEL_INITED
-- a high level PAPI function has been called

ERRORS

PAPI_is_initialized never returns an error.

PAPI_library_init can return the following:

PAPI_EINVAL

papi.h is different from the version used to compile the PAPIlibrary.

PAPI_ENOMEM

Insufficient memory to complete the operation.

PAPI_ESBSTR

This substrate does not support the underlying hardware.

PAPI_ESYS

A system or C library call failed inside PAPI, see the *errno* variable.

EXAMPLES

```
      int retval;

/* Initialize the library */

retval = PAPI_library_init(PAPI_VER_CURRENT);

if (retval != PAPI_VER_CURRENT && retval > 0) {
  fprintf(stderr,"PAPI library version mismatch!\n");
  exit(1); }

if (retval < 0)
  handle_error(retval);

retval = PAPI_is_initialized();

if (retval != PAPI_LOW_LEVEL_INITED)
  handle_error(retval);
```

BUGS

If you don't call this before using any of the low level PAPI calls, your application could core dump.

SEE ALSO

PAPI_thread_init, PAPI

NAME

```
PAPI_library_init   - initialize the PAPI library.
PAPI_is_initialized - check for initialization.
```

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_library_init(int version);
int PAPI_is_initialized(void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_library_init(C_INT check)
PAPIF_is_initialized(C_INT check)
```

DESCRIPTION

PAPI_library_init() initializes the PAPI library. It must be called before any low level PAPI functions can be used. If your application is making use of threads PAPI_thread_init must also be called prior to making any calls to the library other than **PAPI_library_init()**.
PAPI_is_initialized() returns the status of the PAPI library. The PAPI library can be in one of three states, as described under RETURN VALUES.

ARGUMENTS

version -- upon initialization, PAPI checks the argument against the internal value of **PAPI_VER_CURRENT** when the library was compiled. This guards against portability problems when updating the PAPI shared libraries on your system.

RETURN VALUES

PAPI_library_init : On success, this function returns **PAPI_VER_CURRENT.** A positive return code other than **PAPI_VER_CURRENT** indicates a library version mis-match. A negative error code indicates an initialization error.
PAPI_is_initialized :
PAPI_NOT_INITED
-- PAPI has not been initialized
PAPI_LOW_LEVEL_INITED
-- PAPI_library_init has been called
PAPI_HIGH_LEVEL_INITED
-- a high level PAPI function has been called

ERRORS

PAPI_is_initialized never returns an error.

PAPI_library_init can return the following:

PAPI_EINVAL

papi.h is different from the version used to compile the PAPI library.

PAPI_ENOMEM

Insufficient memory to complete the operation.

PAPI_ESBSTR

This substrate does not support the underlying hardware.

PAPI_ESYS

A system or C library call failed inside PAPI, see the *errno* variable.

EXAMPLES

```
      int retval;

/* Initialize the library */

retval = PAPI_library_init(PAPI_VER_CURRENT);

if (retval != PAPI_VER_CURRENT && retval > 0) {
  fprintf(stderr,"PAPI library version mismatch!\n");
  exit(1); }

if (retval < 0)
  handle_error(retval);

retval = PAPI_is_initialized();

if (retval != PAPI_LOW_LEVEL_INITED)
  handle_error(retval);
```

BUGS

If you don't call this before using any of the low level PAPI calls, your application could core dump.

SEE ALSO

PAPI_thread_init, PAPI

NAME

PAPI_list_events - list the events in an eventset

SYNOPSIS

C Interface

```
#include  papi.h
int PAPI_list_events(int EventSet, int *Events, int *number);
```

Fortran Interface

```
#include  fpapi.h
PAPIF_list_events(C_INT EventSet, C_INT(*) Events, C_INT number, C_INT
check)
```

DESCRIPTION

PAPI_list_events() decomposes an event set into the hardware events it contains.
This call assumes an initialized PAPI library and a successfully added event set.

ARGUMENTS

EventSet -- An integer handle for a PAPI event set as created by PAPI_create_eventset
**Events* -- An array of codes for events, such as PAPI_INT_INS. No more than **number* codes will be stored into the array.
**number* -- On input the variable determines the size of the *Events* array. On output the variable contains the number of counters in the event set.
Note that if the given array *Events* is too short to hold all the counters in the event set the **number* variable will be greater than the actually stored number of counter codes.

RETURN VALUES

PAPI_OK
The call returned successfully.

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOEVST
The EventSet specified does not exist.

EXAMPLES

```
/* Convert an event name to an event code */
if (PAPI_event_name_to_code("PAPI_TOT_INS",&EventCode) != PAPI_OK)
  exit(1);
```

```
  /* Add Total Instructions Executed to our EventSet */
  if (PAPI_add_event(EventSet, EventCode) != PAPI_OK)
    exit(1);

  /* Convert a second event name to an event code */
  if (PAPI_event_name_to_code("PAPI_L1_LDM",&EventCode) != PAPI_OK)
    exit(1);

  /* Add L1 Load Misses to our EventSet */
  if (PAPI_add_event(EventSet, EventCode) != PAPI_OK)
    exit(1);

  /* List the events in our EventSet */
  number = 4;
  if(PAPI_list_events(EventSet, Events, &number);
    exit(1);

  if(number != 2)
    exit(1);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_event_name_to_code, PAPI_add_event, PAPI_create_eventset, PAPI_event_code_to_name, PAPI,

NAME

PAPI_list_threads - list the registered thread ids

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_list_threads(PAPI_thread_id_t *id, int *number);
```

Fortran Interface

```
<none>
```

DESCRIPTION

PAPI_list_threads() returns to the caller a list of all thread ID's known to PAPI.
This call assumes an initialized PAPI library.

ARGUMENTS

**id* -- A pointer to a preallocated array. This may be NULL to only return a count of threads. No more than **number* codes will be stored in the array.
**number* -- An input and output parameter, input specifies the number of allocated elements in **id* (if non-NULL) and output specifies the number of threads.

RETURN VALUES

PAPI_OK
The call returned successfully.

PAPI_EINVAL
One or more of the arguments is invalid.

EXAMPLES

```
/* Reserved for example usage */
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_thread_init, PAPI_thread_id, PAPI_register_thread, PAPI_unregister_thread, PAPI_get_thr_specific, PAPI_set_thr_specific, PAPI

NAME

PAPI_lock - Lock one of two mutex variables defined in papi.h
PAPI_unlock - Unlock one of the mutex variables defined in papi.h

SYNOPSIS

C Interface

```
#include <papi.h>
void PAPI_lock(int lock);
void PAPI_unlock(int lock);
```

Fortran Interface

```
#include fpapi.h
PAPIF_lock(C_INT lock)
PAPIF_unlock(C_INT lock)
```

DESCRIPTION

PAPI_lock() Grabs access to one of the two PAPI mutex variables. This function is provided to the user to have a platform independent call to (hopefully) efficiently implemented mutex.
PAPI_unlock() unlocks the mutex acquired by a call to **PAPI_lock**.

ARGUMENT

lock -- an integer value specifying one of the two user locks: **PAPI_USR1_LOCK or PAPI_USR2_LOCK**

RETURN VALUES

There are no return values for these calls. Upon return from **PAPI_lock** the current thread has acquired exclusive access to the specified PAPI mutex.

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_thread_init

NAME

PAPI_multiplex_init - initialize multiplex support in the PAPI library

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_multiplex_init (void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_multiplex_init(C_INT check)
```

DESCRIPTION

PAPI_multiplex_init enables and initializes multiplex support in the PAPI library. Multiplexing allows a user to count more events than total physical counters by time sharing the existing counters at some loss in precision. Applications that make no use of multiplexing do not need to call this routine.

RETURN VALUES

This function always returns **PAPI_OK.**

ERRORS

No errors are reported.

EXAMPLES

```
retval = PAPI_multiplex_init();
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_set_multiplex, PAPI_get_multiplex

NAME

PAPI_num_events - return the number of events in an event set

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_num_events(int EventSet);
```

Fortran Interface

```
#include fpapi.h
PAPIF_num_events(C_INT EventSet, C_INT count)
```

DESCRIPTION

PAPI_num_events() returns the number of preset events contained in an event set. The event set should be created by PAPI_create_eventset

ARGUMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset
**count* -- On output the variable contains the number of events in the event set.

RETURN VALUES

On success, this function returns the positive number of events in the event set. On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
The event count is zero; only if code is compiled with debug enabled.

PAPI_ENOEVST
The EventSet specified does not exist.

EXAMPLES

```
/* Count the events in our EventSet */
printf("%d events found in EventSet.0, PAPI_num_events(EventSet));
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_add_event, PAPI_create_eventset, PAPI, PAPIF

NAME

PAPI_num_counters - PAPI High Level: return the number of hardware counters available on the system

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_num_counters(void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_num_counters(C_INT number)
```

DESCRIPTION

PAPI_num_counters() returns the optimal length of the values array for the high level functions. This value corresponds to the number of hardware counters supported by the current substrate.
PAPI_num_counters() initializes the library to **PAPI_HIGH_LEVEL_INITED** if necessary.

RETURN VALUES

On success, this function returns the number of hardware counters available.
On error, a negative error code is returned.

ERRORS

PAPI_EINVAL
papi.h is different from the version used to compile the PAPI library.

PAPI_ENOMEM
Insufficient memory to complete the operation.

PAPI_ESYS
A system or C library call failed inside PAPI, see the *errno* variable.

EXAMPLES

```
        int num_hwcntrs;
        /*  The installation does not support PAPI */
        if ((num_hwcntrs = PAPI_num_counters()) < 0 )
          handle_error(1);

  /*  The installation supports PAPI, but has no counters */
  if ((num_hwcntrs = PAPI_num_counters()) == 0 )
    fprintf(stderr,"Info:: This machine does not provide hardware counters.0);
```

BUGS

If you don't call this function, your application could core dump.

SEE ALSO

PAPI, PAPIF

NAME

PAPI_num_hwctrs - return the number of hardware counters

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_num_hwctrs();
```

Fortran Interface

```
#include fpapi.h
PAPIF_num_hwctrs(C_INT num)
```

DESCRIPTION

PAPI_num_hwctrs() returns the number of physical hardware counters present in the processor. This count does not include any special purpose registers or performancehardware. PAPI_library_init must be called in order for this function to return anything greater than 0.

ARGUMENTS

This function takes no arguments.

RETURN VALUES

On success, this function returns a value greater than zero.
A zero result usually means the library has not been initialized.

EXAMPLES

```
/* Query the substrate for our resources. */
printf("%d hardware counters found.0, PAPI_num_hwctrs());
```

BUGS

None.

SEE ALSO

PAPI_init_library, PAPI, PAPI_F

NAME

PAPI_overflow - set up an event set to begin registering overflows
_papi_overflow_handler - user defined function to process overflow events

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_overflow
      (int EventSet, int EventCode, int threshold, int flags,
PAPI_overflow_handler_t handler);
(*PAPI_overflow_handler_t) _papi_overflow_handler
      (int EventSet, void * address, long_long overflow_vector, void *
context);
```

Fortran Interface

```
Not implemented
```

DESCRIPTION

PAPI_overflow() marks a specific *EventCode* in an *EventSet* to generate an overflow signal after every *threshold* events are counted. More than one event in an event set can be used to trigger overflows. In such cases, the user must call this function once for each overflowing event. To turn off overflow on a specified event, call this function with a *threshold* value of 0.
Overflows can be implemented in either software or hardware, but the scope is the entire eventset. PAPI defaults to hardware overflow if it is available. In the case of software overflow, a periodic timer interrupt causes PAPI to compare the event counts against the *threshold* values and call the overflow handlerif one or more events have exceeded their *threshold.* In the case of hardware overflow, the counters are typically set to the negative of the *threshold* value and count up to 0. This zero-crossing triggers a hardware interrupt that calls the overflow handler.Because of this counter interrupt, the counter values for overflowing counters may be very small or even negative numbers, and cannot be relied upon as accurate. In such cases the overflow handler canapproximate the counts by supplying the *threshold* value whenever an overflow occurs.
_papi_overflow_handler() is a placeholder for a user-defined function to process overflow events. A pointer to this function is passed to the **PAPI_overflow** routine, where it is invoked whenever a software or hardware overflow occurs. This handler receives the *EventSet* of the overflowing event, the Program Counter *address* when the interrupt occured, an *overflow_vector* that can be processed to determined which event(s) caused the overflow, and a pointer to the machine *context,* which can be used in a platform-specific manor to extract register information about what was happening when the overflow occured.

ARGUMENTS

EventSet -- an integer handle to a PAPI event set as created by PAPI_create_eventset
EventCode -- the preset or native event code to be set for overflow detection. This event must have already been added to the *EvenSet.*
threshold -- the overflow threshold value for this *EventCode.*

flags -- bit map that controls the overflow mode of operation. Set to PAPI_OVERFLOW_FORCE_SW to force software overflowing, even if hardware overflow support is available. If hardware overflow support is available on a given system, it will be the default mode of operation. There are situations where it is advantageous to use software overflow instead. Although software overflow is inherently less accurate, with more latency and processing overhead, it does allow for overflowing on derived events, and for the accurate recording of overflowing event counts. These two features are typically not available with hardware overflow. Only one type of overflow is allowed per event set, so setting one event to hardware overflow and another to forced software overflow will result in an error being returned.
handler -- pointer to the user supplied handler function to call upon overflow
address -- the Program Counter address at the time of the overflow
overflow_vector -- a long_long word containing flag bits to indicate which hardware counter(s) caused the overflow
**context* -- pointer to a machine specific structure that defines the register context at the time of overflow. This parameter is often unused and can be ignored in the user function.

RETURN VALUES

On success, **PAPI_overflow** returns **PAPI_OK.** On error, a non-zero error code is returned.
_papi_overflow_handler is a void function and returns nothing.

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid. Specifically, a bad threshold value.

PAPI_ENOMEM

Insufficient memory to complete the operation.

PAPI_ENOEVST

The EventSet specified does not exist.

PAPI_EISRUN

The EventSet is currently counting events.

PAPI_ECNFLCT

The underlying counter hardware cannot count this event and other events in the EventSet simultaneously. Or you are trying to overflow both by hardware and by forced software at the same time.

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLES

Define a simple overflow handler:

```
void handler(int EventSet, void *address, long_long overflow_vector,
void *context)
{
  fprintf(stderr,"Overflow at %p! bit=0x%llx \n",
          address,overflow_vector);
}
```

Call PAPI_overflow for an event set containing the PAPI_TOT_INS event, setting the threshold to 100000. Use the handler defined above.

```
retval = PAPI_overflow(EventSet, PAPI_TOT_INS, 100000, 0, handler);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_get_overflow_event_index

NAME

```
PAPI_perror   - convert PAPI error codes to strings, and print error
message to stderr.
PAPI_strerror - convert PAPI error codes to strings, and return the
error string to user.
```

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_perror(int code, char *destination, int length);
char *PAPI_strerror(int code);
```

Fortran Interface

```
#include fpapi.h
PAPIF_perror(C_INT code, C_STRING destination, C_INT check)
```

DESCRIPTION

PAPI_perror() fills the string *destination* with the error message corresponding to the error code *code*. The function copies *length* worth of the error description string corresponding to *code* into destination. The resulting string is always null terminated. If length is 0, then the string is printed on stderr. **PAPI_strerror()** returns a pointer to the error message corresponding to the error code *code*. If the call fails the function returns the NULL pointer. This function is not implemented in Fortran.

ARGUMENTS

code -- the error code to interpret
**destination* -- "the error message in quotes"
length -- either 0 or strlen(destination)

RETURN VALUES

On success **PAPI_perror()** returns **PAPI_OK.** and **PAPI_strerror()** returns a non-NULL pointer.

ERRORS

PAPI_EINVAL
One or more of the arguments to **PAPI_perror()** is invalid.

NULL The input error code to **PAPI_strerror()** is invalid.

EXAMPLE

```
int EventSet = PAPI_NULL;
int native = 0x0;
```

```
      char error_str[PAPI_MAX_STR_LEN];

      if ((retval = PAPI_create_eventset(&EventSet)) != PAPI_OK)
        {
           fprintf(stderr, "PAPI error %d:
      %s\n",retval,PAPI_strerror(retval));
           exit(1);
        }

/* Add Total Instructions Executed to our EventSet */

if ((retval = PAPI_add_event(EventSet, PAPI_TOT_INS)) != PAPI_OK)
  {
     PAPI_perror(retval,error_str,PAPI_MAX_STR_LEN);
     fprintf(stderr,"PAPI_error %d: %s\n",retval,error_str);
     exit(1);
  }


/* Start counting */

if ((retval = PAPI_start(EventSet)) != PAPI_OK)
  handle_error(retval);
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_set_debug, PAPI_set_opt, PAPI_get_opt, PAPI_shutdown,

NAME

PAPI_profil - generate a histogram of hardware counter overflows vs. PC addresses

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_profil(void * buf, unsigned bufsiz, unsigned long offset,
                unsigned scale, int EventSet, int EventCode,  int
threshold,
                int flags);
```

Fortran Interface

```
The profiling routines have no Fortran interface.
```

DESCRIPTION

PAPI_profil() provides hardware event statistics by profiling the occurence of specified hardware counter events. It is designed to mimic the UNIX SVR4 profil call. The statistics are generated by creating a histogram of hardware counter event overflows vs. program counter addresses for the current process. The histogram is defined for a specific region of program code to be profiled, and the identified region is logically broken up into a set of equal size subdivisions, each of which corresponds to a count in the histogram. With each hardware event overflow, the current subdivision is identified and its corresponding histogram count is incremented. These counts establish a relative measure of how many hardware counter events are occuring in each code subdivision. The resulting histogram counts for a profiled region can be used to identify those program addresses that generate a disproportionately high percentage of the event of interest.
Events to be profiled are specified with the *EventSet* and *EventCode* parameters. More than one event can be simultaneously profiled by calling **PAPI_profil()** several times with different *EventCode* values. Profiling can be turned off for a given event by calling **PAPI_profil()** with a *threshold* value of 0.

ARGUMENTS

**buf* -- pointer to a buffer of *bufsiz* bytes in which the histogram counts are stored in an array of unsigned short, unsigned int, or unsigned long long values, or 'buckets'. The size of the buckets is determined by values in the *flags* argument.
bufsiz -- the size of the histogram buffer in bytes. It is computed from the length of the code region to be profiled, the size of the buckets, and the scale factor as discussed below.
offset -- the start address of the region to be profiled.
scale -- broadly and historically speaking, a contraction factor that indicates how much smaller the histogram buffer is than the region to be profiled. More precisely, scale is interpreted as an unsigned 16-bit fixed-point fraction with the decimal point implied on the left. Its value is the reciprocal of the number of addresses in a subdivision, per counter of histogram buffer. Below is a table of representative values for scale:

Representative values for the scale variable		
HEX	DECIMAL	DEFININTION

0x20000	131072	Maps precisely one instruction address to a unique bucket in buf.
0x10000	65536	Maps precisely two instruction addresses to a unique bucket in buf.
0xFFFF	65535	Maps approximately two instruction addresses to a unique bucket in buf.
0x8000	32768	Maps every four instruction addresses to a bucket in buf.
0x4000	16384	Maps every eight instruction addresses to a bucket in buf.
0x0002	2	Maps all instruction addresses to the same bucket in buf.
0x0001	1	Undefined.
0x0000	0	Undefined.

Historically, the scale factor was introduced to allow the allocation of buffers smaller than the code size to be profiled. Data and instruction sizes were assumed to be multiples of 16-bits. These assumptions are no longer necessarily true. **PAPI_profil** has preserved the traditional definition of scale where appropriate, but deprecated the definitions for 0 and 1 (disable scaling) and extended the range of scale to include 65536 and 131072 to allow for exactly two addresses and exactly one address per profiling bucket.
The value of bufsiz is computed as follows:
bufsiz = (end - start)(bucket_size/2)*(scale/65536)* where
bufsiz - the size of the buffer in bytes
end, start - the ending and starting addresses of the profiled region
bucket_size - the size of each bucket in bytes; 2, 4, or 8 as defined in *flags*
scale - as defined above
EventSet -- The PAPI EventSet to profile. This EventSet is marked as profiling-ready, but profiling doesn't actually start until a **PAPI_start()** call is issued.
EventCode -- Code of the Event in the EventSet to profile. This event must already be a member of the EventSet.
threshold -- minimum number of events that must occur before the PC is sampled. If hardware overflow is supported for your substrate, this threshold will trigger an interrupt when reached. Otherwise, the counters will be sampled periodically and the PC will be recorded for the first sample that exceeds the threshold. If the value of threshold is 0, profiling will be disabled for this event.
flags -- bit pattern to control profiling behavior. Defined values are shown in the table below.

Defined bits for the flags variable	
PAPI_PROFIL_POSIX	Default type of profiling, similar to
PAPI_PROFIL_RANDOM	Drop a random 25% of the samples.
PAPI_PROFIL_WEIGHTED	Weight the samples by their value.
PAPI_PROFIL_COMPRESS	Ignore samples as values in the hash buckets get big.
PAPI_PROFIL_BUCKET_16	Use unsigned short (16 bit) buckets, This is the default bucket.
PAPI_PROFIL_BUCKET_32	Use unsigned int (32 bit) buckets.
PAPI_PROFIL_BUCKET_64	Use unsigned long long (64 bit) buckets.
PAPI_PROFIL_FORCE_SW	Force software overflow in profiling.

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ENOMEM

Insufficient memory to complete the operation.

PAPI_ENOEVST

The EventSet specified does not exist.

PAPI_EISRUN

The EventSet is currently counting events.

PAPI_ECNFLCT

The underlying counter hardware can not count this event and other events in the EventSet simultaneously.

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLES

```
      int retval;
      unsigned long length;
      PAPI_exe_info_t *prginfo;
      unsigned short *profbuf;

if ((prginfo = PAPI_get_executable_info()) == NULL)
  handle_error(1);

length = (unsigned long)(prginfo->text_end - prginfo->text_start);

profbuf = (unsigned short *)malloc(length);
if (profbuf == NULL)
  handle_error(1);
memset(profbuf,0x00,length);
 .
 .
 .
if ((retval = PAPI_profil(profbuf, length, start, 65536, EventSet,
                PAPI_FP_INS, 1000000, PAPI_PROFIL_POSIX | PAPI_PROFIL_BUCKET_16)) !
= PAPI_OK)
  handle_error(retval);
```

BUGS

If you call PAPI_profil, PAPI allocates buffer space that will not be freed if you call PAPI_shutdown or PAPI_cleanup_eventset. To clean all memory, you must call PAPI_profil on the Events with a 0 threshold.

SEE ALSO

PAPI_sprofil, PAPI_overflow, PAPI_get_executable_info

NAME

```
PAPI_query_event - query if PAPI event exists
```

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_query_event(int EventCode);
```

Fortran Interface

```
#include fpapi.h
PAPIF_query_event(C_INT EventCode, C_INT check)
```

DESCRIPTION

PAPI_query_event() asks the PAPI library if the PAPI Preset event can be counted on this architecture. If the event CAN be counted, the function returns PAPI_OK. If the event CANNOT be counted, the function returns an error code. This function also can be used to check the syntax of a native event.

ARGUMENTS

EventCode -- a defined event such as PAPI_TOT_INS.

RETURN VALUES

On success, **PAPI_query_event returns PAPI_OK**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOTPRESET
The hardware event specified is not a valid PAPI preset.

PAPI_ENOEVNT
The PAPI preset is not available on the underlying hardware.

EXAMPLES

```
int retval;

/* Initialize the library */
```

```
retval = PAPI_library_init(PAPI_VER_CURRENT);

if (retval != PAPI_VER_CURRENT) {
  fprintf(stderr,"PAPI library init error!\n");
  exit(1); }

if (PAPI_query_event(PAPI_TOT_INS) != PAPI_OK) {
  fprintf(stderr,"No instruction counter? How lame.\n");
  exit(1);
  }
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_native, PAPI_remove_event, PAPI_remove_events,

NAME

PAPI_read - read hardware counters from an event set
PAPI_accum - accumulate and reset counters in an event set

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_read(int EventSet, long_long *values);
int PAPI_accum(int EventSet, long_long *values);
```

Fortran Interface

```
#include fpapi.h
PAPIF_read(C_INT EventSet, C_LONG_LONG(*) values, C_INT check)
PAPIF_accum(C_INT EventSet, C_LONG_LONG(*) values, C_INT check)
```

DESCRIPTION

These calls assume an initialized PAPI library and a properly added event set.
PAPI_read() copies the counters of the indicated event set into the array *values*. The counters continue counting after the read.
PAPI_accum() adds the counters of the indicated event set into the array *values*. The counters are zeroed and continue counting after the operation.
Note the differences between PAPI_read() and PAPI_accum(), specifically that PAPI_accum() resets the values array to zero.

ARGUMENTS

EventSet -- an integer handle for a PAPI Event Set as created by PAPI_create_eventset
**values* -- an array to hold the counter values of the counting events

RETURN VALUES

On success, these functions return **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ESYS
A system or C library call failed inside PAPI, see the *errno* variable.

PAPI_ENOEVST
The event set specified does not exist.

EXAMPLES

```
do_100events();
if (PAPI_read(EventSet, values) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 100 */
do_100events();
if (PAPI_accum(EventSet, values) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 200 */
values[0] = -100;
do_100events();
if (PAPI_accum(EventSet, values) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 0 */
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_set_opt, PAPI_reset, PAPI_start, PAPI, PAPIF

NAME

PAPI_read_counters - PAPI High Level: read and reset counters
PAPI_accum_counters - PAPI High Level: accumulate and reset counters

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_read_counters(long_long *values, int array_len);
int PAPI_accum_counters(long_long *values, int array_len);
```

Fortran Interface

```
#include fpapi.h
PAPIF_read_counters(C_LONG_LONG(*) values, C_INT array_len, C_INT
check)
PAPIF_accum_counters(C_LONG_LONG(*) values, C_INT array_len, C_INT
check)
```

DESCRIPTION

PAPI_read_counters() copies the event counters into the array *values* .
The counters are reset and left running after the call.
PAPI_accum_counters() adds the event counters into the array *values* .
The counters are reset and left running after the call.
These calls assume an initialized PAPI library and a properly added event set.

ARGUMENTS

**values* -- an array to hold the counter values of the counting events
array_len -- the number of items in the *events array

RETURN VALUES

On success, these functions return **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ESYS

A system or C library call failed inside PAPI, see the *errno* variable.

EXAMPLES

```
do_100events();
if (PAPI_read_counters(values, num_hwcntrs) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 100 */
do_100events();
if (PAPI_accum_counters(values, num_hwcntrs) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 200 */
values[0] = -100;
do_100events();
if (PAPI_accum_counters(values, num_hwcntrs) != PAPI_OK)
  handle_error(1);
/* values[0] now equals 0 */
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_start_counters, PAPI_set_opt, PAPI, PAPIF

NAME

PAPI_register_thread, PAPI_unregister_thread - Inform PAPI of thread status

SYNOPSIS

```
#include <papi.h>
int PAPI_register_thread (void);
int PAPI_unregister_thread (void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_register_thread(C_INT check)
PAPIF_unregister_thread(C_INT check)
```

DESCRIPTION

PAPI_register_thread should be called when the user wants to force PAPI to initialize a thread that PAPI has not seen before. Usually this is not necessary as PAPI implicitly detects the thread when an eventset is created or other thread local PAPI functions are called. However, it can be useful for debugging and performance enhancements in the run-time systems of performance tools.
PAPI_unregister_thread should be called when the user wants to shutdown a particular thread and free the associated thread ID. THIS IS IMPORTANT IF YOUR THREAD LIBRARY REUSES THE SAME THREAD ID FOR A NEW KERNEL LWP. OpenMP does this. OpenMP parallel regions, if separated by a call to omp_set_num_threads() will often kill off the underlying kernel LWPs and then start new ones for the next region. However, omp_get_thread_id() does not reflect this, as the thread IDs for the new LWPs will be the same as the old LWPs. PAPI needs to know that the underlying LWP has changed so it can set up the counters for that new thread. This is accomplished by calling this function.

ARGUMENTS

None.

RETURN VALUES

On success, this function returns **PAPI_OK.** On error, a non-zero error code is returned.

ERRORS

PAPI_ENOMEM
Space could not be allocated to store the new thread information.

PAPI_ESYS
A system or C library call failed inside PAPI, see the *errno* variable.

PAPI_ESBSTR
Hardware counters for this thread could not be initialized.

EXAMPLES

None.

SEE ALSO

PAPI_thread_init, PAPI_thread_id

NAME

```
PAPI_remove_event  - remove PAPI preset or native hardware event from
an EventSet
PAPI_remove_events - remove PAPI presets or native hardware events from
an EventSet
```

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_remove_event(int EventSet, int EventCode);
int PAPI_remove_events(int EventSet, int *EventCode, int number);
```

Fortran Interface

```
#include fpapi.h
PAPIF_remove_event(C_INT EventSet, C_INT EventCode, C_INT check)
PAPIF_remove_events(C_INT EventSet, C_INT(*) EventCode, C_INT number,
C_INT check)
```

DESCRIPTION

PAPI_remove_event() removes a hardware event to a PAPI event set. **PAPI_remove_events()** does the same, but for an array of hardware event codes.
A hardware event can be either a PAPI Preset or a native hardware event code. For a list of PAPI preset events, see PAPI_presets or run the *avail* test case in the PAPI distribution. PAPI Presets can be passed to PAPI_query_event to see if they exist on the underlying architecture. For a list of native events available on current platform, run native_avail test case in the PAPI distribution. For the encoding of native events, see PAPI_event_name_to_code to learn how to generate native code for the supported native event on the underlying architecture."
It should be noted that **PAPI_remove_events** can partially succeed, exactly like **PAPI_add_events**.

ARGUMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset
EventCode -- a defined event such as PAPI_TOT_INS or a native event.
**EventCode* -- an array of defined events
number -- an integer indicating the number of events in the array **EventCode*

RETURN VALUES

On success, these functions return **PAPI_OK.** On error, a less than zero error code is returned or the the number of elements that succeeded before the error.

ERRORS

Positive integer

The number of consecutive elements that succeeded before the error.

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ENOEVST

The EventSet specified does not exist.

PAPI_EISRUN

The EventSet is currently counting events.

PAPI_ECNFLCT

The underlying counter hardware can not count this event and other events in the EventSet simultaneously.

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLES

```
      int EventSet = PAPI_NULL;
      unsigned int native = 0x0;

      if (PAPI_create_eventset(&EventSet) != PAPI_OK)
        handle_error(1);

/* Add Total Instructions Executed to our EventSet */

if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
  handle_error(1);

/* Start counting */

if (PAPI_start(EventSet) != PAPI_OK)
  handle_error(1);

/* Stop counting, ignore values */

if (PAPI_stop(EventSet, NULL) != PAPI_OK)
  handle_error(1);

/* Remove event */

if (PAPI_remove_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
  handle_error(1);
```

BUGS

The vector function should take a pointer to a length argument so a proper return value can be set upon partial success.

SEE ALSO

PAPI_preset,
PAPI_add_event(3),
PAPI_add_events(3),
PAPI_cleanup_eventset, PAPI_destroy_eventset, PAPI_event_name_to_code

NAME

```
PAPI_remove_event  - remove PAPI preset or native hardware event from
an EventSet
PAPI_remove_events - remove PAPI presets or native hardware events from
an EventSet
```

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_remove_event(int EventSet, int EventCode);
int PAPI_remove_events(int EventSet, int *EventCode, int number);
```

Fortran Interface

```
#include fpapi.h
PAPIF_remove_event(C_INT EventSet, C_INT EventCode, C_INT check)
PAPIF_remove_events(C_INT EventSet, C_INT(*) EventCode, C_INT number,
C_INT check)
```

DESCRIPTION

PAPI_remove_event() removes a hardware event to a PAPI event set. **PAPI_remove_events()** does the same, but for an array of hardware event codes.
A hardware event can be either a PAPI Preset or a native hardware event code. For a list of PAPI preset events, see PAPI_presets or run the *avail* test case in the PAPI distribution. PAPI Presets can be passed to PAPI_query_event to see if they exist on the underlying architecture. For a list of native events available on current platform, run native_avail test case in the PAPI distribution. For the encoding of native events, see PAPI_event_name_to_code to learn how to generate native code for the supported native event on the underlying architecture."
It should be noted that **PAPI_remove_events** can partially succeed, exactly like **PAPI_add_events**.

ARGUMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset
EventCode -- a defined event such as PAPI_TOT_INS or a native event.
**EventCode* -- an array of defined events
number -- an integer indicating the number of events in the array **EventCode*

RETURN VALUES

On success, these functions return **PAPI_OK.** On error, a less than zero error code is returned or the the number of elements that succeeded before the error.

ERRORS

Positive integer

The number of consecutive elements that succeeded before the error.

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ENOEVST

The EventSet specified does not exist.

PAPI_EISRUN

The EventSet is currently counting events.

PAPI_ECNFLCT

The underlying counter hardware can not count this event and other events in the EventSet simultaneously.

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLES

```
      int EventSet = PAPI_NULL;
      unsigned int native = 0x0;

      if (PAPI_create_eventset(&EventSet) != PAPI_OK)
        handle_error(1);

/* Add Total Instructions Executed to our EventSet */

if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
  handle_error(1);

/* Start counting */

if (PAPI_start(EventSet) != PAPI_OK)
  handle_error(1);

/* Stop counting, ignore values */

if (PAPI_stop(EventSet, NULL) != PAPI_OK)
  handle_error(1);

/* Remove event */

if (PAPI_remove_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
  handle_error(1);
```

BUGS

The vector function should take a pointer to a length argument so a proper return value can be set upon partial success.

SEE ALSO

PAPI_preset,
PAPI_add_event(3),
PAPI_add_events(3),
PAPI_cleanup_eventset, PAPI_destroy_eventset, PAPI_event_name_to_code

NAME

PAPI_reset - reset the hardware event counts in an event set

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_reset (int EventSet);
```

Fortran Interface

```
#include fpapi.h
PAPIF_reset(C_INT EventSet, C_INT check)
```

DESCRIPTION

PAPI_reset() zeroes the values of the counters contained in *EventSet*. This call assumes an initialized PAPI library and a properly added event set.

ARGUMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_ESYS
A system or C library call failed inside PAPI, see the *errno* variable.

PAPI_ENOEVST
The EventSet specified does not exist.

EXAMPLES

```
if (PAPI_reset(EventSet) != PAPI_OK)
  handle_error(1);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI, PAPIF

NAME

PAPI_set_debug - set the current debug level for PAPI

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_set_debug(int debuglevel);
```

Fortran Interface

```
#include fpapi.h
PAPIF_set_debug(C_INT debug, C_INT check)
```

DESCRIPTION

PAPI_set_debug sets the debug level for error output from the PAPI library.

ARGUMENTS

debuglevel -- one of the constants shown in the table below and defined in the papi.h header file. The current debug level is used by both the internal error and debug message handler subroutines. The debug handler is only used if the library was compiled with -DDEBUG. The debug handler is called when there is an error upon a call to the PAPI API. The error handler is always active and it's behavior cannot be modified except for whether or not it prints anything. NOTE: This is the ONLY function that may be called BEFORE **PAPI_library_init().**
The PAPI error handler prints out messages in the following form:
PAPI Error: message.
The default PAPI debug handler prints out messages in the following form:
PAPI Error: Error Code code,symbol,description
If the error was caused from a system call and the return code is PAPI_ESYS, the message will have a colon space and the error string as reported by strerror() appended to the end.
The possible debug levels for debugging are shown in the table below.

PAPI_QUIET	Do not print anything, just return the error code
PAPI_VERB_ECONT	Print error message and continue
PAPI_VERB_ESTOP	Print error message and exit

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL

The debuglevel is invalid.

EXAMPLES

```
if ( PAPI_set_debug(PAPI_VERB_ECONT) != PAPI_OK )
   handle_error();
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_set_opt, PAPI_get_opt, PAPI_library_init

NAME

PAPI_set_domain - set the default execution domain for new event sets

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_set_domain(int domain);
```

Fortran Interface

```
#include fpapi.h
PAPIF_set_domain(C_INT domain, C_INT check)
PAPIF_set_event_domain(C_INT EventSet, C_INT domain, C_INT check)
```

DESCRIPTION

PAPI_set_domain sets the default execution domain for all new event sets created by PAPI_create_eventset in all threads. Event sets that are already in existance are not affected. To change the domain of an existing event set, please see the PAPI_set_opt man page. The reader should note that the domain of an event set affects only which mode the counter continue to run. Counts are still aggregated for the current process, and not for any other processes in the system. Thus when requesting **PAPI_DOM_KERNEL**, the user is asking for events that occur on behalf of the process, inside the kernel.

ARGUMENTS

domain -- one of the following constants as defined in the papi.h header file:

PAPI_DOM_USER	User context counted
PAPI_DOM_KERNEL	Kernel/OS context counted
PAPI_DOM_OTHER	Exception/transient mode counted
PAPI_DOM_SUPERVISOR	Supervisor/hypervisor context counted
PAPI_DOM_ALL	All above contexts counted
PAPI_DOM_MIN	The smallest available context
PAPI_DOM_MAX	The largest available context

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOEVST
The event set specified does not exist.

PAPI_EISRUN
The event set is currently counting events.

EXAMPLES

```
      int retval;

/* Initialize the library */

retval = PAPI_library_init(PAPI_VER_CURRENT);

if (retval > 0 && retval != PAPI_VER_CURRENT) {
  fprintf(stderr,"PAPI library version mismatch!0);
  exit(1); }

if (retval < 0)
  handle_error(retval);

if ((retval = PAPI_set_domain(PAPI_DOM_KERNEL)) != PAPI_OK)
  handle_error(retval);

if ((retval = PAPI_create_eventset(&EventSet)) != PAPI_OK)
  handle_error(retval);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_set_opt

NAME

PAPI_set_event_info - set an event's name, description and definition info

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_set_event_info(PAPI_event_info_t *info, int *EventCode, int
replace);
```

DESCRIPTION

NOTE: This API has been deprecated in PAPI 3.5 pending a data structure redesign.
This function modifies or adds an event to the PAPI preset event table based on the contents of an even info structure. This function presently works only to define or modify PAPI preset events.

ARGUMENTS

The following arguments are explicitly passed, or are implicit in the info structure.
EventCode -- event code returned by the function on success
replace -- 1 to replace an existing event, or 0 to prevent accidental replacement
info -- structure containing the event information. Relevant fields in this structure are discussed below.
event_code -- although the value of this event code is not used, the PAPI_PRESET_MASK bit must be set to indicate that the following event description is for a preset event.
symbol -- name of the preset event. If the event name is found in the table and *replace* is non-zero, the event definition will be replaced. If the names do not match a new entry will be created.
derived -- a string value indicating whether and how native event terms are combined to form a preset event. Possible values include:
NOT_DERIVED: Do nothing; only one native event,
DERIVED_ADD: Add all native events,
DERIVED_CMPD: Event lives in first counter but takes 2 or more native codes,
DERIVED_SUB: Subtract all events from the first event specified,
DERIVED_POSTFIX: Process events based on specified postfix string,
postfix -- a string value containing postfix operations used only for DERIVED_POSTFIX events.
short_descr -- short description of the event
long_descr -- detailed description of the event
event_note -- special information or notes about the event
name -- an array of up to 8 names of native events that make up this preset event.

RETURN VALUES

On success, the function returns **PAPI_OK.** The EventCode parameter will also be set to the new event code for this event. On error, a non-zero error code is returned by the function.

ERRORS

PAPI_EPERM

You are trying to modify an existing event without specifying *replace*.

PAPI_EISRUN

You are trying to modify an event that has been added to an EventSet.

PAPI_EINVAL

One or more of the arguments or fields of the info structure is invalid.

PAPI_ENOTPRESET

The PAPI preset table is full and there is no room for a new event.

PAPI_ENOEVNT

The event specified is not a PAPI preset. Usually because the PAPI_PRESET_MASK bit is not set.

EXAMPLE

```
/*Add a note to a custom definition of PAPI_TOT_INS */
PAPI_event_name_to_code("PAPI_TOT_INS",&EventCode)
if (PAPI_get_event_info(EventCode, &info) != PAPI_OK)
  handle_error(1);
strcpy(info.symbol, "MY_TOT_INS");
strcpy(info.note, "This note describes my version of total
instructions.");
if (PAPI_set_event_info(&info, EventCode, 0) != PAPI_OK)
  handle_error(1);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI, PAPIF, PAPI_get_event_info , PAPI_set_event_info , PAPI_event_name_to_code

NAME

PAPI_set_granularity - set the execution granularity for which events are counted

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_set_granularity(int granularity);
```

Fortran Interface

```
#include fpapi.h
PAPIF_set_granularity(C_INT granularity, C_INT check)
```

DESCRIPTION

This function is currently unimplemented.

RETURN VALUES

ERRORS

EXAMPLES

BUGS

This function is currently unimplemented.

SEE ALSO

PAPI_set_domain, PAPI_set_opt, PAPI_get_opt

NAME

PAPI_get_multiplex - get the multiplexing status of specified event set PAPI_set_multiplex - convert a standard event set to a multiplexed event set

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_get_multiplex(int EventSet);
int PAPI_set_multiplex(int EventSet);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_multiplex(C_INT EventSet, C_INT check)
PAPIF_set_multiplex(C_INT EventSet, C_INT check)
```

DESCRIPTION

PAPI_get_multiplex tests the state of the *PAPI_MULTIPLEXING* flag in the specified event set, returning *TRUE* if a PAPI event set is multiplexed, or *FALSE* if not.
PAPI_set_multiplex converts a standard PAPI event set created by a call to **PAPI_create_eventset()** into an event set capable of handling multiplexed events. This must be done after calling **PAPI_multiplex_init()**, but prior to calling **PAPI_start().** Events can be added to an event set either before or after converting it into a multiplexed set, but the conversion must be done prior to using it as a multiplexed set.

ARGUMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset

RETURN VALUES

PAPI_get_multiplex returns either *TRUE* (positive non-zero) if multiplexing is enabled for this event set, *FALSE* (zero) if multiplexing is not enabled, or *PAPI_ENOEVST* if the specified event set cannot be found.
On success, **PAPI_get_multiplex** returns *PAPI_OK*. On error, a non-zero error code is returned, as described below.

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid, or the EventSet is already multiplexed.

PAPI_ENOEVST

The EventSet specified does not exist.

PAPI_EISRUN

The EventSet is currently counting events.

PAPI_ENOMEM

Insufficient memory to complete the operation.

EXAMPLES

```
  retval = PAPI_get_multiplex(EventSet);
  if (retval > 0) printf("This event set is ready for multiplexing0")
  if (retval == 0) printf("This event set is not enabled for
multiplexing0")
  if (retval < 0) handle_error(retval);

  retval = PAPI_set_multiplex(EventSet);
  if ((retval == PAPI_EINVAL) && (PAPI_get_multiplex(EventSet) > 0))
    printf("This event set already has multiplexing enabled0);
  else if (retval != PAPI_OK) handle_error(retval);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_multiplex_init, PAPI_set_opt, PAPI_create_eventset

NAME

PAPI_get_opt - get PAPI library or event set options
PAPI_set_opt - set PAPI library or event set options
PAPIF_get_clockrate - get the clockrate (Fortran only)
PAPIF_get_domain - get the counting domain (Fortran only)
PAPIF_get_granularity - get the counting granularity (Fortran only)
PAPIF_get_preload - get the library preload setting (Fortran only)

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_get_opt(int option, PAPI_option_t *ptr);
int PAPI_set_opt(int option, PAPI_option_t *ptr);
```

Fortran Interface

```
#include fpapi.h
PAPIF_get_clockrate(C_INT clockrate)
PAPIF_get_domain(C_INT EventSet, C_INT domain, C_INT mode, C_INT check)
PAPIF_get_granularity(C_INT EventSet, C_INT granularity, C_INT mode,
C_INT check)
PAPIF_get_preload(C_STRING preload, C_INT check)
```

DESCRIPTION

PAPI_get_opt() and **PAPI_set_opt()** query or change the options of the PAPI library or a specific event set created by PAPI_create_eventset The C interface for these functions passes a pointer to the *PAPI_option_t* structure. Not all options require or return information in this structure, and not all options are implemented for both get and set.
The Fortran interface is a series of calls implementing various subsets of the C interface. Not all options in C are available in Fortran.
NOTE: Some options, such as PAPI_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.
The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described in the table below. The Fortran calls are implementations of specific options.
ptr -- is a pointer to a structure that acts as both an input and output parameter. It is defined in **papi.h** and below.
EventSet -- input; a reference to an EventSetInfo structure
clockrate -- output; cycle time of this CPU in MHz; *may* be an estimate generated at init time with a quick timing routine
domain -- output; execution domain for which events are counted
granularity -- output; execution granularity for which events are counted
mode -- input; determines if domain or granularity are default or for the current event set

preload -- output; environment variable string for preloading libraries

OPTIONS TABLE

Predefined name	Explanation
General information requests	
PAPI_CLOCKRATE	Get clockrate in MHz.
PAPI_MAX_CPUS	Get number of CPUs.
PAPI_MAX_HWCTRS	Get number of counters.
PAPI_EXEINFO	Get Executable addresses for text/data/bss.
PAPI_HWINFO	Get information about the hardware.
PAPI_SHLIBINFO	Get shared library information used by the program.
PAPI_SUBSTRATEINFO	Get the PAPI features the substrate supports
PAPI_LIB_VERSION	Get the full PAPI version of the library
PAPI_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
Defaults for the global library	
PAPI_DEFDOM	Get/Set default counting domain for newly created event sets.
PAPI_DEFGRN	Get/Set default counting granularity.
PAPI_DEBUG	Get/Set the PAPI debug state and the debug handler. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level. The debug handler is available in ptr->debug.handler. For information regarding the behavior of the handler, please see the man page for PAPI_set_debug.
Multiplexing control	
PAPI_MULTIPLEX	Get/Set options for multiplexing.
PAPI_MAX_MPX_CTRS	Get maximum number of multiplexing counters.
PAPI_DEF_MPX_USEC	Get/Set the sampling time slice in microseconds for multiplexing.
Manipulating individual event sets	
PAPI_ATTACH	Get thread or process id to which event set is attached. Returns TRUE if currently attached. Set event set specified in ptr->ptr->attach.eventset to be attached to thread or process id specified in in ptr->attach.tid
PAPI_DETACH	Get thread or process id to which event set is attached. Returns TRUE if currently detached. Set event set specified in ptr->ptr->attach.eventset to be detached from any thread or process id.
PAPI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_GRANUL	Get/Set granularity for a single event set. The event set is specified in

	ptr->granularity.eventset. Not implementedyet.
Platform specific options	
PAPI_DATA_ADDRESS	Set data address range to restrict event counting for event set specified in ptr->addr.eventset. Starting and ending addresses are specified in ptr->addr.start and ptr->addr.end, respectively. If exact addresses cannot be instantiated, offsets are returned in ptr->addr.start_off and ptr->addr.end_off. Currently implemented on Itanium only.
PAPI_INSTR_ADDRESS	Set instruction address range as described above. Itanium only.

The **option_t *** *ptr* structure is defined in **papi.h** and looks something like the following example from the source tree. Users should use the definition in **papi.h** which is in synch with the library used.

```
typedef union {
  PAPI_preload_option_t preload;
  PAPI_debug_option_t debug;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_attach_option_t attach;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_shlib_info_t *shlib_info;
  PAPI_exe_info_t *exe_info;
  PAPI_substrate_info_t *sub_info;
  PAPI_overflow_option_t ovf_info;
  PAPI_addr_range_option_t addr;
} PAPI_option_t;
```

RETURN VALUES

On success, these functions return *PAPI_OK*. On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOEVST
The event set specified does not exist.

PAPI_EISRUN
The event set is currently counting events.

EXAMPLES

```
PAPI_option_t options;

if ((num = PAPI_get_opt(PAPI_MAX_HWCTRS,NULL)) <= 0)
  handle_error();
```

```
printf("This machine has %d counters.0,num);

/* Set the domain of this EventSet
   to counter user and kernel modes for this
   process */

memset(&options,0x0,sizeof(options));
options.domain.eventset = EventSet;
options.domain.domain = PAPI_DOM_ALL;
if (PAPI_set_opt(PAPI_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

BUGS

The granularity functions are not yet implemented. The domain functions are only implemented on some platforms. There are no known bugs in these functions.

SEE ALSO

PAPI_set_debug, PAPI_set_multiplex, PAPI_set_domain

NAME

PAPI_get_thr_specific, PAPI_set_thr_specific - Store or retrieve a pointer to a thread specific data structure

SYNOPSIS

```
#include <papi.h>
int PAPI_get_thr_specific(int tag, void **ptr);
int PAPI_set_thr_specific(int tag, void *ptr);
```

DESCRIPTION

In C, PAPI_set_thr_specific will save ptr into an array indexed by tag. PAPI_get_thr_specific will retrieve the pointer from the array with index tag. There are 2 user available locations and tag can be either PAPI_USR1_TLS or PAPI_USR2_TLS. The array mentioned above is managed by PAPI and allocated to each thread which has called PAPI_thread_init. There are no Fortran equivalent functions.

ARGUMENTS

tag -- An identifier, the value of which is either PAPI_USR1_TLS or PAPI_USR2_TLS. This identifier indicates which of several data structures associated with this thread is to be accessed.
ptr -- A pointer to the memory containing the data structure.

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a negative error value is returned.

ERRORS

PAPI_EINVAL

The *tag* argument is out of range.

EXAMPLE

```
      HighLevelInfo *state = NULL;

if (retval = PAPI_thread_init(pthread_self) != PAPI_OK)
   handle_error(retval);

/*
 * Do we have the thread specific data setup yet?
 */
if ((retval = PAPI_get_thr_specific(PAPI_USR1_TLS, (void *) &state))
    != PAPI_OK || state == NULL) {
   state = (HighLevelInfo *) malloc(sizeof(HighLevelInfo));
   if (state == NULL)
```

```
        return (PAPI_ESYS);

    memset(state, 0, sizeof(HighLevelInfo));
    state->EventSet = PAPI_NULL;

    if ((retval = PAPI_create_eventset(&state->EventSet)) != PAPI_OK)
       return (PAPI_ESYS);

    if ((retval=PAPI_set_thr_specific(PAPI_USR1_TLS, state))!=PAPI_OK)
       return (retval);
}
```

BUGS

There are no known bugs in these functions.

SEE ALSO

PAPI_thread_init, .BR PAPI_thread_id(3), PAPI_register_thread

NAME

PAPI_set_domain - set the default execution domain for new event sets

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_set_domain(int domain);
```

Fortran Interface

```
#include fpapi.h
PAPIF_set_domain(C_INT domain, C_INT check)
PAPIF_set_event_domain(C_INT EventSet, C_INT domain, C_INT check)
```

DESCRIPTION

PAPI_set_domain sets the default execution domain for all new event sets created by PAPI_create_eventset in all threads. Event sets that are already in existance are not affected. To change the domain of an existing event set, please see the PAPI_set_opt man page. The reader should note that the domain of an event set affects only which mode the counter continue to run. Counts are still aggregated for the current process, and not for any other processes in the system. Thus when requesting **PAPI_DOM_KERNEL**, the user is asking for events that occur on behalf of the process, inside the kernel.

ARGUMENTS

domain -- one of the following constants as defined in the papi.h header file:

PAPI_DOM_USER	User context counted
PAPI_DOM_KERNEL	Kernel/OS context counted
PAPI_DOM_OTHER	Exception/transient mode counted
PAPI_DOM_SUPERVISOR	Supervisor/hypervisor context counted
PAPI_DOM_ALL	All above contexts counted
PAPI_DOM_MIN	The smallest available context
PAPI_DOM_MAX	The largest available context

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOEVST
The event set specified does not exist.

PAPI_EISRUN
The event set is currently counting events.

EXAMPLES

```
      int retval;

/* Initialize the library */

retval = PAPI_library_init(PAPI_VER_CURRENT);

if (retval > 0 && retval != PAPI_VER_CURRENT) {
  fprintf(stderr,"PAPI library version mismatch!0);
  exit(1); }

if (retval < 0)
  handle_error(retval);

if ((retval = PAPI_set_domain(PAPI_DOM_KERNEL)) != PAPI_OK)
  handle_error(retval);

if ((retval = PAPI_create_eventset(&EventSet)) != PAPI_OK)
  handle_error(retval);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_set_opt

NAME

PAPI_shutdown - finish using PAPI and free all related resources

SYNOPSIS

C Interface

```
#include <papi.h>
void PAPI_shutdown (void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_shutdown()
```

DESCRIPTION

PAPI_shutdown() is an exit function used by the PAPI Library to free resources and shut down when certain error conditions arise. It is not necessary for the user to call this function, but doing so allows the user to have the capability to free memory and resources used by the PAPI Library.

BUGS

This function has no known bugs.

SEE ALSO

PAPI_cleanup_eventset, PAPI_destroy_eventset

NAME

PAPI_sprofil - generate PC histogram data from multiple code regions where hardware counter overflow occurs

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_sprofil(PAPI_sprofil_t * prof, int profcnt, int EventSet,
                 int EventCode,  int threshold,  int flags);
```

Fortran Interface

```
The profiling routines have no Fortran interface.
```

DESCRIPTION

PAPI_sprofil() is a structure driven profiler that profiles one or more disjoint regions of code in a single call. It accepts a pointer to a preinitialized array of sprofil structures, and initiates profiling based on the values contained in the array. Each structure in the array defines the profiling parameters that are normally passed to **PAPI_profil().** For more information on profiling, see: PAPI_pofil

STRUCTURE FIELDS

**pr_base* -- pointer to the base address of the buffer.
pr_size -- the size of the histogram buffer in pr_base.
pr_off -- the start address of the region to be profiled.
pr_scale -- the scaling factor applied to the buffer.
These fields are described in greater detail in the documentation for PAPI_pofil

ARGUMENTS

**prof* -- pointer to an array of PAPI_sprofil_t structures.
profcnt -- number of structures in the prof array for hardware profiling.
EventSet -- The PAPI EventSet to profile. This EventSet is marked as profiling-ready, but profiling doesn't actually start until a **PAPI_start()** call is issued.
EventCode -- Code of the Event in the EventSet to profile. This event must already be a member of the EventSet.
threshold -- minimum number of events that must occur before the PC is sampled. If hardware overflow is supported for your substrate, this threshold will trigger an interrupt when reached. Otherwise, the counters will be sampled periodically and the PC will be recorded for the first sample that exceeds the threshold. If the value of threshold is 0, profiling will be disabled for this event.
flags -- bit pattern to control profiling behavior. Defined values are given in a table in the documentation for PAPI_pofil

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

Error returns for **PAPI_sprofil()** are identical to those for PAPI_profil Please refer to that page for further details.

EXAMPLES

```
      int retval;
      unsigned long length;
      PAPI_exe_info_t *prginfo;
      unsigned short *profbuf1, *profbuf2, profbucket;
      PAPI_sprofil_t sprof[3];

if ((prginfo = PAPI_get_executable_info()) == NULL)
  handle_error(1);

length = (unsigned long)(prginfo->text_end - prginfo->text_start);

/* Allocate 2 buffers of equal length */
profbuf1 = (unsigned short *)malloc(length);
profbuf2 = (unsigned short *)malloc(length);
if ((profbuf1 == NULL) || (profbuf2 == NULL))
  handle_error(1);
memset(profbuf1,0x00,length);
memset(profbuf2,0x00,length);

/* First buffer */
sprof[0].pr_base = profbuf1;
sprof[0].pr_size = length;
sprof[0].pr_off = (caddr_t) DO_FLOPS;
sprof[0].pr_scale = 0x10000;

/* Second buffer */
sprof[1].pr_base = profbuf2;
sprof[1].pr_size = length;
sprof[1].pr_off = (caddr_t) DO_READS;
sprof[1].pr_scale = 0x10000;

/* Overflow bucket */
sprof[2].pr_base = profbucket;
sprof[2].pr_size = 1;
sprof[2].pr_off = 0;
sprof[2].pr_scale = 0x0002;

if ((retval = PAPI_sprofil(sprof, EventSet, PAPI_FP_INS, 1000000,
                 PAPI_PROFIL_POSIX | PAPI_PROFIL_BUCKET_16)) != PAPI_OK)
  handle_error(retval);
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_profil, PAPI_get_executable_info, PAPI_overflow

NAME

PAPI_start - start counting hardware events in an event set
PAPI_stop - stop counting hardware events in an event set

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_start(int EventSet);
int PAPI_stop(int EventSet, long_long *values);
```

Fortran Interface

```
#include fpapi.h
PAPIF_start(C_INT EventSet, C_INT check)
PAPIF_stop(C_INT EventSet, C_LONG_LONG(*) values, C_INT check)
```

DESCRIPTION

PAPI_start starts counting all of the hardware events contained in the previously defined EventSet. All counters are implicitly set to zero before counting.
PAPI_stop halts the counting of a previously defined event set and the counter values contained in that EventSet are copied into the values array
These calls assume an initialized PAPI library and a properly added event set.

ARGUMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset
**values* -- an array to hold the counter values of the counting events

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ESYS
A system or C library call failed inside PAPI, see the *errno* variable.

PAPI_ENOEVST
The EventSet specified does not exist.

PAPI_EISRUN
The EventSet is currently counting events. (**PAPI_start()** only)

PAPI_ENOTRUN

The EventSet is currently not running. (**PAPI_stop()** only)

PAPI_ECNFLCT

The underlying counter hardware can not count this event and other events in the EventSet simultaneously. (**PAPI_start()** only)

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLES

```
        if (PAPI_create_eventset(&EventSet) != PAPI_OK)
          handle_error(1);

  /* Add Total Instructions Executed to our EventSet */
  if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
    handle_error(1);

  /* Start counting */
  if (PAPI_start(EventSet) != PAPI_OK)
    handle_error(1);

  poorly_tuned_function();

  if (PAPI_stop(EventSet, values) != PAPI_OK)
    handle_error(1);

  printf("%lld\n",values[0]);
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI, PAPIF

NAME

PAPI_start_counters - PAPI High Level: start counting hardware events

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_start_counters(int *events, int array_len);
```

Fortran Interface

```
#include fpapi.h
PAPIF_start_counters(C_INT(*) events, C_INT array_len, C_INT check)
```

DESCRIPTION

PAPI_start_counters() starts counting the events named in the events array. This function cannot be called if the events array is already running. The user must call PAPI_stop_counters to stop the events explicitly if he/she wants to call this function again. It is the user's responsibility to choose events that can be counted simultaneously by reading the vendor's documentation. The length of the event array should be no longer than the value returned by PAPI_num_counters.

ARGUMENTS

**events* -- an array of codes for events such as PAPI_INT_INS or a native event code
array_len -- the number of items in the *events array

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_EISRUN
Counters already been started, you must call PAPI_stop_counters before you call this function again.

PAPI_ESYS
A system or C library call failed inside PAPI, see the *errno* variable.

PAPI_ENOMEM
Insufficient memory to complete the operation.

PAPI_ECNFLCT

The underlying counter hardware can not count this event and other events in the EventSet simultaneously.

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLES

```
/* Start counting events */
if (PAPI_start_counters(Events, num_hwcntrs) != PAPI_OK)
  handle_error(1);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_stop_counters, PAPI, PAPIF

NAME

PAPI_state - return the counting state of an EventSet

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_state (int EventSet, int *status);
```

Fortran Interface

```
#include fpapi.h
PAPIF_state(C_INT EventSet, C_INT status, C_INT check)
```

DESCRIPTION

PAPI_state() returns the counting state of the specified event set.

ARGUEMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset
status -- an integer containing a boolean combination of one or more of the following nonzero constants as defined in the PAPI header file papi.h:

PAPI_STOPPED	EventSet is stopped
PAPI_RUNNING	EventSet is running
PAPI_PAUSED	EventSet temporarily disabled by the library
PAPI_NOT_INIT	EventSet defined, but not initialized
PAPI_OVERFLOWING	EventSet has overflowing enabled
PAPI_PROFILING	EventSet has profiling enabled
PAPI_MULTIPLEXING	EventSet has multiplexing enabled
PAPI_ACCUMULATING	reserved for future use
PAPI_HWPROFILING	reserved for future use

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ENOEVST

The EventSet specified does not exist.

EXAMPLES

```
      int EventSet = PAPI_NULL;
      int status = 0;

if (PAPI_create_eventset(&EventSet) != PAPI_OK)
  handle_error(1);

/* Add Total Instructions Executed to our EventSet */

if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
  handle_error(1);

/* Start counting */

if (PAPI_state(EventSet, &status) != PAPI_OK)
  handle_error(1);

printf("State is now %d\n",status);

if (PAPI_start(EventSet) != PAPI_OK)
  handle_error(1);

if (PAPI_state(EventSet, &status) != PAPI_OK)
  handle_error(1);

printf("State is now %d\n",status);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_start, PAPI_stop

NAME

PAPI_start - start counting hardware events in an event set
PAPI_stop - stop counting hardware events in an event set

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_start(int EventSet);
int PAPI_stop(int EventSet, long_long *values);
```

Fortran Interface

```
#include fpapi.h
PAPIF_start(C_INT EventSet, C_INT check)
PAPIF_stop(C_INT EventSet, C_LONG_LONG(*) values, C_INT check)
```

DESCRIPTION

PAPI_start starts counting all of the hardware events contained in the previously defined EventSet. All counters are implicitly set to zero before counting.
PAPI_stop halts the counting of a previously defined event set and the counter values contained in that EventSet are copied into the values array
These calls assume an initialized PAPI library and a properly added event set.

ARGUMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset
**values* -- an array to hold the counter values of the counting events

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ESYS
A system or C library call failed inside PAPI, see the *errno* variable.

PAPI_ENOEVST
The EventSet specified does not exist.

PAPI_EISRUN
The EventSet is currently counting events. (**PAPI_start()** only)

PAPI_ENOTRUN

The EventSet is currently not running. (**PAPI_stop()** only)

PAPI_ECNFLCT

The underlying counter hardware can not count this event and other events in the EventSet simultaneously. (**PAPI_start()** only)

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLES

```
        if (PAPI_create_eventset(&EventSet) != PAPI_OK)
          handle_error(1);

  /* Add Total Instructions Executed to our EventSet */
  if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
    handle_error(1);

  /* Start counting */
  if (PAPI_start(EventSet) != PAPI_OK)
    handle_error(1);

  poorly_tuned_function();

  if (PAPI_stop(EventSet, values) != PAPI_OK)
    handle_error(1);

  printf("%lld\n",values[0]);
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI, PAPIF

NAME

PAPI_stop_counters - PAPI High Level: stop counting hardware events and reset values to zero

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_stop_counters(long_long *values, int array_len);
```

Fortran Interface

```
PAPIF_stop_counters(C_LONG_LONG(*) values, C_INT array_len, C_INT
check)
#include fpapi.h
```

DESCRIPTION

PAPI_stop_counters()
This function stops the counters and copies the counts into the values array. The counters must have been started by a previous call to PAPI_start_counters().After this function is called, the values are reset to zero.

ARGUMENTS

**values* -- an array where to put the counter values
array_len -- the number of items in the *values array

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_EINVAL
One or more of the arguments is invalid.

PAPI_ENOTRUN
The eventset is not started yet.

PAPI_ENOEVST
The eventset has not been added yet.

EXAMPLES

```
int Events[2] = { PAPI_TOT_CYC, PAPI_TOT_INS };
long_long values[2];
```

```
/* Start counting events */
if (PAPI_start_counters(Events, 2) != PAPI_OK)
  handle_error(1);
your_slow_code();
/* Stop counting events */
if (PAPI_stop_counters(values, 2) != PAPI_OK)
  handle_error(1);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_start_counters, PAPI_set_opt, PAPI_read_counters, PAPI, PAPIF

NAME

```
PAPI_perror   - convert PAPI error codes to strings, and print error
message to stderr.
PAPI_strerror - convert PAPI error codes to strings, and return the
error string to user.
```

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_perror(int code, char *destination, int length);
char *PAPI_strerror(int code);
```

Fortran Interface

```
#include fpapi.h
PAPIF_perror(C_INT code, C_STRING destination, C_INT check)
```

DESCRIPTION

PAPI_perror() fills the string *destination* with the error message corresponding to the error code *code*. The function copies *length* worth of the error description string corresponding to *code* into destination. The resulting string is always null terminated. If length is 0, then the string is printed on stderr. **PAPI_strerror()** returns a pointer to the error message corresponding to the error code *code*. If the call fails the function returns the NULL pointer. This function is not implemented in Fortran.

ARGUMENTS

code -- the error code to interpret
**destination* -- "the error message in quotes"
length -- either 0 or strlen(destination)

RETURN VALUES

On success **PAPI_perror()** returns **PAPI_OK.** and **PAPI_strerror()** returns a non-NULL pointer.

ERRORS

PAPI_EINVAL
One or more of the arguments to **PAPI_perror()** is invalid.

NULL The input error code to **PAPI_strerror()** is invalid.

EXAMPLE

```
int EventSet = PAPI_NULL;
int native = 0x0;
```

```
      char error_str[PAPI_MAX_STR_LEN];

      if ((retval = PAPI_create_eventset(&EventSet)) != PAPI_OK)
        {
           fprintf(stderr, "PAPI error %d:
      %s\n",retval,PAPI_strerror(retval));
           exit(1);
        }

/* Add Total Instructions Executed to our EventSet */

if ((retval = PAPI_add_event(EventSet, PAPI_TOT_INS)) != PAPI_OK)
  {
     PAPI_perror(retval,error_str,PAPI_MAX_STR_LEN);
     fprintf(stderr,"PAPI_error %d: %s\n",retval,error_str);
     exit(1);
  }


/* Start counting */

if ((retval = PAPI_start(EventSet)) != PAPI_OK)
  handle_error(retval);
```

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_set_debug, PAPI_set_opt, PAPI_get_opt, PAPI_shutdown,

NAME

PAPI_thread_id - get the thread identifier of the current thread

SYNOPSIS

C Interface

```
#include <papi.h>
unsigned long PAPI_thread_id(void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_thread_id(C_INT id)
```

DESCRIPTION

This function returns a valid thread identifier. It calls the function registered with PAPI through a call to **PAPI_thread_init().**

ARGUMENTS

None.

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a negative error value is returned.

ERRORS

PAPI_EMISC
is returned if there are no threads registered.

-1 is returned if the thread id function returns an error.

EXAMPLE

```
unsigned long tid;

if ((tid = PAPI_thread_id()) == (unsigned long int)-1)
  exit(1);

printf("Initial thread id is: %lu\n",tid);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_thread_init

NAME

PAPI_thread_init - initialize thread support in the PAPI library

SYNOPSIS

C Interface

```
#include  papi.h
int PAPI_thread_init (unsigned long int (*handle)());
```

Fortran Interface

```
#include  fpapi.h
PAPIF_thread_init(C_INT FUNCTION handle, C_INT check)
```

DESCRIPTION

PAPI_thread_init initializes thread support in the PAPI library. Applications that make no use of threads do not need to call this routine. This function MUST return a UNIQUE thread ID for every new thread/LWP created. The OpenMP call **omp_get_thread_num()** violates this rule, as the underlying LWPs may have been killed off by the run-time system or by a call to **omp_set_num_threads()**. In that case, it may still possible to use **omp_get_thread_num()** in conjunction with **PAPI_unregister_thread()** when the OpenMP thread has finished. However it is much better to use the underlying thread subsystem's call, which is **pthread_self()** on Linux platforms.

ARGUMENTS

handle -- Pointer to a function that returns current thread ID.

RETURN VALUES

PAPI_OK
The call returned successfully.

PAPI_EINVAL
One or more of the arguments is invalid.

EXAMPLES

```
if (PAPI_thread_init(pthread_self) != PAPI_OK)
  exit(1);
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_thread_id, PAPI_list_threads, PAPI_get_thr_specific, PAPI_set_thr_specific, PAPI_register_thread, PAPI_unregister_thread (3),PAPI

NAME

PAPI_lock - Lock one of two mutex variables defined in papi.h
PAPI_unlock - Unlock one of the mutex variables defined in papi.h

SYNOPSIS

C Interface

```
#include <papi.h>
void PAPI_lock(int lock);
void PAPI_unlock(int lock);
```

Fortran Interface

```
#include fpapi.h
PAPIF_lock(C_INT lock)
PAPIF_unlock(C_INT lock)
```

DESCRIPTION

PAPI_lock() Grabs access to one of the two PAPI mutex variables. This function is provided to the user to have a platform independent call to (hopefully) efficiently implemented mutex.
PAPI_unlock() unlocks the mutex acquired by a call to **PAPI_lock**.

ARGUMENT

lock -- an integer value specifying one of the two user locks: **PAPI_USR1_LOCK or PAPI_USR2_LOCK**

RETURN VALUES

There are no return values for these calls. Upon return from **PAPI_lock** the current thread has acquired exclusive access to the specified PAPI mutex.

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_thread_init

NAME

PAPI_register_thread, PAPI_unregister_thread - Inform PAPI of thread status

SYNOPSIS

```
#include <papi.h>
int PAPI_register_thread (void);
int PAPI_unregister_thread (void);
```

Fortran Interface

```
#include fpapi.h
PAPIF_register_thread(C_INT check)
PAPIF_unregister_thread(C_INT check)
```

DESCRIPTION

PAPI_register_thread should be called when the user wants to force PAPI to initialize a thread that PAPI has not seen before. Usually this is not necessary as PAPI implicitly detects the thread when an eventset is created or other thread local PAPI functions are called. However, it can be useful for debugging and performance enhancements in the run-time systems of performance tools.
PAPI_unregister_thread should be called when the user wants to shutdown a particular thread and free the associated thread ID. THIS IS IMPORTANT IF YOUR THREAD LIBRARY REUSES THE SAME THREAD ID FOR A NEW KERNEL LWP. OpenMP does this. OpenMP parallel regions, if separated by a call to omp_set_num_threads() will often kill off the underlying kernel LWPs and then start new ones for the next region. However, omp_get_thread_id() does not reflect this, as the thread IDs for the new LWPs will be the same as the old LWPs. PAPI needs to know that the underlying LWP has changed so it can set up the counters for that new thread. This is accomplished by calling this function.

ARGUMENTS

None.

RETURN VALUES

On success, this function returns **PAPI_OK.** On error, a non-zero error code is returned.

ERRORS

PAPI_ENOMEM
Space could not be allocated to store the new thread information.

PAPI_ESYS
A system or C library call failed inside PAPI, see the *errno* variable.

PAPI_ESBSTR
Hardware counters for this thread could not be initialized.

EXAMPLES

None.

SEE ALSO

PAPI_thread_init, PAPI_thread_id

NAME

PAPI_write - Write counter values into counters

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_write(int EventSet, long_long *values);
```

Fortran Interface

```
#include fpapi.h
PAPIF_write(C_INT EventSet, C_LONG_LONG(*) values, C_INT check)
```

DESCRIPTION

PAPI_write() writes the counter values provided in the array *values* into the event set *EventSet*. The virtual counters managed by the PAPI library will be set to the values provided. If the event set is running, an attempt will be made to write the values to the running counters. This operation is not permitted by all substrates and may result in a run-time error.

ARGUMENTS

EventSet -- an integer handle for a PAPI event set as created by PAPI_create_eventset
**values* -- an array to hold the counter values of the counting events

RETURN VALUES

On success, this function returns **PAPI_OK.**
On error, a non-zero error code is returned.

ERRORS

PAPI_ENOEVST

The EventSet specified does not exist.

PAPI_ESBSTR

PAPI_write() is not implemented for this architecture. **PAPI_ESYS** The EventSet is currently counting events and the substrate could not change the values of the running counters.

EXAMPLES

```
/* Yet to be written */
```

BUGS

This function has no known bugs.

SEE ALSO

PAPI_read, PAPI, PAPIF,