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**
and
the **PAPI Software Specification**.

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

High Level Functions

A simple interface for single thread 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_flops  - simplified call to get Mflops/s, 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 when using the high-level interface the use of PAPI_library_init to initialize the library is optional. If, however, explicit initialization is not made, either of PAPI_flops or PAPI_num_counters *must* be called before any other call to a PAPI function.

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_add_event  - add single PAPI preset or native hardware event to an
event set
PAPI_add_events  - add array of PAPI preset or native hardware events
to an event set
PAPI_add_pevent  - reserved for future use
```

PAPI_cleanup_eventset - remove all PAPI events from an event set
PAPI_create_eventset - create a new empty PAPI event set
PAPI_describe_event - return description of event given the name or event code
PAPI_destroy_eventset - deallocates memory associated with an empty PAPI event set
PAPI_event_code_to_name - translate an integer PAPI event code into an ASCII PAPI preset name
PAPI_event_name_to_code - translate an ASCII PAPI preset name into an integer PAPI event code
PAPI_get_executable_info - get the executable's address space information
PAPI_get_hardware_info - get information about the system hardware
PAPI_get_opt - query the option settings of the PAPI library or a specific event set
PAPIF_get_clockrate - get the processor clockrate in MHz. Fortran only.
PAPIF_get_domain - get the domain of the specified eventset. Fortran only.
PAPIF_get_granularity - get the granularity of the specified eventset. Fortran only.
PAPIF_get_preload - get the 'LD_PRELOAD' environment equivalent. Fortran only.
PAPI_get_overflow_address - return the address at which overflow occurred for profiling
PAPI_get_real_cyc - return the total number of cycles since some arbitrary starting point
PAPI_get_real_usec - return the total number of microseconds since some arbitrary starting point
PAPI_get_virt_cyc - return the process cycles since some arbitrary starting point
PAPI_get_virt_usec - return the process microseconds since some arbitrary starting point
PAPI_label_event - return a short label for an event given the event code
PAPI_library_init - initialize the PAPI library
PAPI_list_events - list the events defined in an event set
PAPI_lock - lock the PAPI internal mutex variable
PAPI_multiplex_init - initialize multiplex support in the PAPI library
PAPI_num_hwctrs - return the number of physical hardware counters available
PAPI_overflow - set up an event set to begin registering overflows
PAPI_perror - return a copy of the error message corresponding to a specified error code
PAPI_profil - generate PC histogram data where hardware counter overflow occurs
PAPI_query_all_events_verbose - request detailed information on all PAPI events
PAPI_query_event - query if a PAPI event exists
PAPI_query_event_verbose - request detailed information on a PAPI event
PAPI_read - read hardware events from an event set with no reset
PAPI_rem_event - remove a hardware event from a PAPI event set
PAPI_rem_events - remove an array of hardware events from a PAPI event set
PAPI_rem_pevent - reserved for future use
PAPI_reset - reset the hardware event counts in an EventSet
PAPI_restore - Restore the saved state of the PAPI library
PAPI_save - Restore the saved state of the PAPI library
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_opt - change the option settings of the PAPI library or a specific event set
PAPI_set_multiplex - convert a standard event set to a multiplexed event set
PAPI_shutdown - finish using PAPI and free all related resources
PAPI_sprofil - generate PC histogram data where hardware counter overflow occurs
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 the PAPI internal mutex variable
PAPI_write - write counter values into counters

AUTHORS

Philip J. Mucci <mucci@cs.utk.edu>
Kevin London <london@cs.utk.edu>
Dan Terpstra <terpstra@cs.utk.edu>

SEE ALSO

The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPIF - Performance Application Programming Interface (Fortran)

SYNOPSIS

#include fpapi.h

call PAPIF_function_name(arg1,arg2,...,check)

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.

DIAGNOSTICS

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

SEE ALSO

The PAPI Web Site: http://icl.cs.utk.edu/projects/papi
The PAPI Interface: PAPI

NAME

PAPI_presets - PAPI predefined named events

SYNOPSIS

#include <papi.h>

DESCRIPTION

The PAPI library names a number of predefined 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 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:

Conditional Branching:

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_FLOPS	Floating point instructions per second
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_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_IPS	Instructions per second
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	Level 1 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	Level 1 instruction cache misses
PAPI_L1_ICR	L1 instruction cache reads

PAPI_L1_ICW	L1 instruction cache writes
PAPI_L1_LDM	Level 1 load misses
PAPI_L1_STM	Level 1 store misses
PAPI_L1_TCA	L1 total cache accesses
PAPI_L1_TCH	L1 total cache hits
PAPI_L1_TCM	Level 1 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	Level 2 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	Level 2 instruction cache misses
PAPI_L2_ICR	L2 instruction cache reads
PAPI_L2_ICW	L2 instruction cache writes
PAPI_L2_LDM	Level 2 load misses
PAPI_L2_STM	Level 2 store misses
PAPI_L2_TCA	L2 total cache accesses
PAPI_L2_TCH	L2 total cache hits
PAPI_L2_TCM	Level 2 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	Level 3 Data Cache Hits
PAPI_L3_DCM	Level 3 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	Level 3 instruction cache misses
PAPI_L3_ICR	L3 instruction cache reads

PAPI_L3_ICW	L3 instruction cache writes
PAPI_L3_LDM	Level 3 load misses
PAPI_L3_STM	Level 3 store misses
PAPI_L3_TCA	L3 total cache accesses
PAPI_L3_TCH	L3 total cache hits
PAPI_L3_TCM	Level 3 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

AUTHOR

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 hardware platforms.

SEE ALSO

PAPI, PAPI_query_event The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_read, PAPI_accum - read hardware events, accumulate and reset hardware events from 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

PAPI_read() copies the counters of the indicated event set into the array *values*. The counters are left counting after the read.

PAPI_accum() adds the counters of the indicated event set into the array *values*. The counters are zeroed and left counting after the operation.

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

```
int EventSet = PAPI_NULL;
unsigned int native = 0x0;
long_long values[1] = (long_long) 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_start(EventSet) != PAPI_OK)
  handle_error(1);

poorly_tuned_function();

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

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

poorly_tuned_function();

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

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_add_event, PAPI_reset, PAPI_set_opt, PAPI_rem_event, PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_read_counters, PAPI_accum_counters - PAPI High Level: read counting hardware events

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.

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

```
int Events[2] = { PAPI_TOT_CYC, PAPI_TOT_INS };
long_long values[2];
int num_hwcntrs = 0;

if ((num_hwcntrs = PAPI_num_counters()) != PAPI_OK)
  handle_error(1);

if (num_hwcntrs > 2)
  num_hwcntrs = 2;

/* Start counting events */

if (PAPI_start_counters(Events, num_hwcntrs) != PAPI_OK)
  handle_error(1);

your_slow_code();

/* Start counting events */

if (PAPI_read_counters(values, num_hwcntrs) != PAPI_OK)
  handle_error(1);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_num_counters, PAPI_start_counters, PAPI_stop_counters,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_add_event, PAPI_add_events, PAPI_add_pevent - add PAPI preset or native hardware event 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 *EventCode, int number);
int PAPI_add_pevent(int *EventSet, int code, void * misc);
```

Fortran Interface

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

DESCRIPTION

PAPI_add_event() adds a hardware event to a PAPI Event Set. **PAPI_add_events()** does the same, but for an array of hardware event codes. **PAPI_add_pevent()** is reserved for future use.

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*

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 the encoding of native events, see the <arch>.README file also included the distribution.

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_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.

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 (0xc1 on hardware counter 1) */

native = (0xc1 << 8) | 1;
if (PAPI_add_event(&EventSet, native) != PAPI_OK)
  handle_error(1);

/* Start counting */

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_set_opt, PAPI_start, PAPI_rem_event, PAPI_rem_events,
PAPI_query_event, PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_add_event, PAPI_add_events, PAPI_add_pevent - add PAPI preset or native hardware event 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 *EventCode, int number);
int PAPI_add_pevent(int *EventSet, int code, void * misc);
```

Fortran Interface

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

DESCRIPTION

PAPI_add_event() adds a hardware event to a PAPI Event Set. **PAPI_add_events()** does the same, but for an array of hardware event codes. **PAPI_add_pevent()** is reserved for future use.

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*

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 the encoding of native events, see the <arch>.README file also included the distribution.

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_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.

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 (0xc1 on hardware counter 1) */

native = (0xc1 << 8) | 1;
if (PAPI_add_event(&EventSet, native) != PAPI_OK)
  handle_error(1);

/* Start counting */

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_set_opt, PAPI_start, PAPI_rem_event, PAPI_rem_events,
PAPI_query_event, PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_add_event, PAPI_add_events, PAPI_add_pevent - add PAPI preset or native hardware event 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 *EventCode, int number);
int PAPI_add_pevent(int *EventSet, int code, void * misc);
```

Fortran Interface

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

DESCRIPTION

PAPI_add_event() adds a hardware event to a PAPI Event Set. **PAPI_add_events()** does the same, but for an array of hardware event codes. **PAPI_add_pevent()** is reserved for future use.

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*

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 the encoding of native events, see the <arch>.README file also included the distribution.

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_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.

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 (0xc1 on hardware counter 1) */

native = (0xc1 << 8) | 1;
if (PAPI_add_event(&EventSet, native) != PAPI_OK)
  handle_error(1);

/* Start counting */

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_set_opt, PAPI_start, PAPI_rem_event, PAPI_rem_events,
PAPI_query_event, PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_cleanup_eventset, PAPI_destroy_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.

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

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. Attempting to to destroy a non-empty event set is one such case.

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

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

/* 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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI_query, PAPI_add_event, PAPI_start, PAPI_rem_event, PAPI_rem_events, PAPI_shutdown,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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.*

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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_destroy_eventset, PAPI_cleanup_eventset, PAPI_rem_event, PAPI_rem_events, PAPI_add_event, PAPI_add_events, PAPI_add_pevent,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_describe_event, PAPI_event_code_to_name, PAPI_event_name_to_code, PAPI_label_event - convert a hardware event to a description, label, number or string symbol.

SYNOPSIS

C Interface

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

Fortran Interface

```
#include fpapi.h
PAPIF_describe_event(C_STRING EventName, C_INT EventCode, C_STRING
EventDescr, C_INT check)
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)
PAPIF_label_event(C_INT EventCode, C_STRING EventLabel, C_INT check)
```

DESCRIPTION

PAPI_describe_event() is used to translate either an ASCII PAPI preset name or an integer PAPI preset event code into the corresponding event code or name as well as an ASCII description of that event. If the *EventName* argument is a string of length >0 it is assumed to contain the name to look up and the corresponding event code is returned in the argument *EventCode*. Otherwise the *EventCode* argument is used to look up the event name which is stored in the *EventName* argument. Finally a descriptive string of length less than **PAPI_MAX_STR_LEN** is copied to the argument *EventDescr*.

The functionality of this call is a superset of the following two calls.

PAPI_event_name_to_code() is used to translate an ASCII PAPI preset name into an integer PAPI event code.

PAPI_event_code_to_name() is used to translate an integer PAPI event code into an ASCII PAPI preset name.

PAPI_label_event() is used to translate an integer PAPI event code into a short (<=18 character) ASCII label that is more descriptive than the preset name but shorter than the description. These labels can be used as event identifiers in third party tools.

ARGUMENTS

EventName -- a string containing the event name as listed in PAPI_presets

EventCode -- the numeric code for the event

EventDescr -- a descriptive string for the event of length less than **PAPI_MAX_STR_LEN**.

EventLabel -- a short descriptive label for the event of length less than 18 characters.

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.

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);

/* Describe event from EventCode */
*EventCodeStr=' '; /* Erase the name argument to look up the code */
if (PAPI_describe_event(EventCodeStr,&EventCode,EventDescr) != PAPI_OK)
  handle_error(1);

/* Label event from EventCode */
if (PAPI_label_event(EventCode,EventLabel) != PAPI_OK)
  handle_error(1);

/* Print the differing text strings */
printf("Name: %s0abel: %s0escription:
%s0,EventCodeStr,EventLabel,EventDescr);

/* 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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_add_event, PAPI_set_opt, PAPI_query_event,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_cleanup_eventset, PAPI_destroy_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.

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

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. Attempting to to destroy a non-empty event set is one such case.

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

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

/* 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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI_query, PAPI_add_event, PAPI_start, PAPI_rem_event, PAPI_rem_events, PAPI_shutdown,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_describe_event, PAPI_event_code_to_name, PAPI_event_name_to_code, PAPI_label_event - convert a hardware event to a description, label, number or string symbol.

SYNOPSIS

C Interface

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

Fortran Interface

```
#include fpapi.h
PAPIF_describe_event(C_STRING EventName, C_INT EventCode, C_STRING
EventDescr, C_INT check)
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)
PAPIF_label_event(C_INT EventCode, C_STRING EventLabel, C_INT check)
```

DESCRIPTION

PAPI_describe_event() is used to translate either an ASCII PAPI preset name or an integer PAPI preset event code into the corresponding event code or name as well as an ASCII description of that event. If the *EventName* argument is a string of length >0 it is assumed to contain the name to look up and the corresponding event code is returned in the argument *EventCode*. Otherwise the *EventCode* argument is used to look up the event name which is stored in the *EventName* argument. Finally a descriptive string of length less than **PAPI_MAX_STR_LEN** is copied to the argument *EventDescr*.

The functionality of this call is a superset of the following two calls.

PAPI_event_name_to_code() is used to translate an ASCII PAPI preset name into an integer PAPI event code.

PAPI_event_code_to_name() is used to translate an integer PAPI event code into an ASCII PAPI preset name.

PAPI_label_event() is used to translate an integer PAPI event code into a short (<=18 character) ASCII label that is more descriptive than the preset name but shorter than the description. These labels can be used as event identifiers in third party tools.

ARGUMENTS

EventName -- a string containing the event name as listed in PAPI_presets

EventCode -- the numeric code for the event

EventDescr -- a descriptive string for the event of length less than **PAPI_MAX_STR_LEN**.

EventLabel -- a short descriptive label for the event of length less than 18 characters.

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.

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);

/* Describe event from EventCode */
*EventCodeStr=' '; /* Erase the name argument to look up the code */
if (PAPI_describe_event(EventCodeStr,&EventCode,EventDescr) != PAPI_OK)
  handle_error(1);

/* Label event from EventCode */
if (PAPI_label_event(EventCode,EventLabel) != PAPI_OK)
  handle_error(1);

/* Print the differing text strings */
printf("Name: %s0abel: %s0escription:
%s0,EventCodeStr,EventLabel,EventDescr);

/* 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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_add_event, PAPI_set_opt, PAPI_query_event,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_describe_event, PAPI_event_code_to_name, PAPI_event_name_to_code, PAPI_label_event - convert a hardware event to a description, label, number or string symbol.

SYNOPSIS

C Interface

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

Fortran Interface

```
#include fpapi.h
PAPIF_describe_event(C_STRING EventName, C_INT EventCode, C_STRING
EventDescr, C_INT check)
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)
PAPIF_label_event(C_INT EventCode, C_STRING EventLabel, C_INT check)
```

DESCRIPTION

PAPI_describe_event() is used to translate either an ASCII PAPI preset name or an integer PAPI preset event code into the corresponding event code or name as well as an ASCII description of that event. If the *EventName* argument is a string of length >0 it is assumed to contain the name to look up and the corresponding event code is returned in the argument *EventCode*. Otherwise the *EventCode* argument is used to look up the event name which is stored in the *EventName* argument. Finally a descriptive string of length less than **PAPI_MAX_STR_LEN** is copied to the argument *EventDescr*.

The functionality of this call is a superset of the following two calls.

PAPI_event_name_to_code() is used to translate an ASCII PAPI preset name into an integer PAPI event code.

PAPI_event_code_to_name() is used to translate an integer PAPI event code into an ASCII PAPI preset name.

PAPI_label_event() is used to translate an integer PAPI event code into a short (<=18 character) ASCII label that is more descriptive than the preset name but shorter than the description. These labels can be used as event identifiers in third party tools.

ARGUMENTS

EventName -- a string containing the event name as listed in PAPI_presets

EventCode -- the numeric code for the event

EventDescr -- a descriptive string for the event of length less than **PAPI_MAX_STR_LEN**.

EventLabel -- a short descriptive label for the event of length less than 18 characters.

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.

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);

/* Describe event from EventCode */
*EventCodeStr=' '; /* Erase the name argument to look up the code */
if (PAPI_describe_event(EventCodeStr,&EventCode,EventDescr) != PAPI_OK)
  handle_error(1);

/* Label event from EventCode */
if (PAPI_label_event(EventCode,EventLabel) != PAPI_OK)
  handle_error(1);

/* Print the differing text strings */
printf("Name: %s0abel: %s0escription:
%s0,EventCodeStr,EventLabel,EventDescr);

/* 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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_add_event, PAPI_set_opt, PAPI_query_event,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

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

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

The first call to **PAPI_flops()** will initialize PAPI, set up the counters to monitor PAPI_FP_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 floating point instructions since the start of the measurement and Mflops/s rate since latest call to **PAPI_flops()** . Any call with *flpins* = -1 will reinitialize all counters to 0.

ARGUMENTS

**rtime* -- total realtime since the first PAPI_flops() call

**ptime* -- total process time since the first PAPI_flops() call

**flpins* -- total floating point instructions since the first PAPI_flops() call

**mflops* -- Mflops/s achieved since the latest PAPI_flops() call

RETURN VALUES

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

NOTES

Most platforms are only capable of counting the number of floating point instructions completed. This may or may not translate to your definition of floating point operations. The measured rate is thus Mflips/s, and will in some circumstances count FMA instructions as one operation. Consult the hardware documentation for your system for specifics.

PAPI_flops() 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

This function has no known bugs.

SEE ALSO

PAPI_accum,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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 C_LONG_LONG data_start,
           C_LONG_LONG  data_end,
         C_LONG_LONG C_LONG_LONG bss_end,
         C_STRING 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

lib_preload_env -- environment variable for preloading libraries

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

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

printf("Start of user program is at %p\n",prginfo->text_start);
printf("End of user program is at %p\n",prginfo->text_end);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

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

SEE ALSO

PAPI_library_init, PAPI_get_opt,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_get_hardware_info - get the info 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.

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

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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

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

SEE ALSO

PAPI_library_init, PAPI_get_opt,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_get_opt, PAPI_set_opt - get/set PAPI library or event set options
PAPIF_get_clockrate, PAPIF_get_domain, PAPIF_get_granularity, PAPIF_get_preload - Fortran implementation of some PAPI_get_opt options

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. 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_SET_DOMAIN, 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 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

Predefined name	Explanation
General information requests	
PAPI_GET_CLOCKRATE	Return clockrate in MHz.
PAPI_GET_MAX_CPUS	Return number of CPUs.
PAPI_GET_MAX_HWCTRS	Return number of counters.
PAPI_GET_EXEINFO	Addresses for text/data/bss.
PAPI_GET_HWINFO	Info. about hardware.
PAPI_GET_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
Defaults for the global library	
PAPI_GET_DEFDOM	Return default counting domain for newly created event sets.
PAPI_SET_DEFDOM	Set default counting domain.
PAPI_GET_DEFGRN	Return default counting granularity.
PAPI_SET_DEFGRN	Set default counting granularity.
PAPI_GET_DEBUG	Get the PAPI debug state. The available debug states are defined in The debug state is available in ptr->debug
PAPI_SET_DEBUG	Set the PAPI debug state
Multiplexing control	
PAPI_GET_MULTIPLEX	Get options for multiplexing. Currently not implemented.
PAPI_SET_MULTIPLEX	Set options for multiplexing
Manipulating individual event sets	
PAPI_GET_DOMAIN	Get domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_SET_DOMAIN	Set the domain for a single event set.
PAPI_GET_GRANUL	Get granularity for a single event set. The event set is specified in ptr->granularity.eventset
PAPI_SET_GRANUL	Set the granularity for a single event set.

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_inherit_option_t inherit;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_exe_info_t *exe_info; } PAPI_option_t;
```

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 num, EventSet = PAPI_NULL;
PAPI_option_t options;

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

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

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

/* 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_SET_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

There are no known bugs in these functions.

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_overflow, PAPI_get_overflow_address - set up an event set to begin registering overflows; return the address at which an overflow occured

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_overflow (int EventSet, int EventCode, int threshold, int
flags, PAPI_overflow_handler_t handler);
void *PAPI_get_overflow_address (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. Only one event in an event set can be used as an overflow trigger. Subsequent calls to **PAPI_overflow()** replace earlier calls. To turn off overflow, set the handler to NULL.

PAPI_get_overflow_address() returns the instruction pointer where overflow occured. This routine is often used as part of the overflow handler routine.

ARGUMENTS

EventSet -- a reference to the event set to use

EventCode -- the counter to be used for overflow detection

threshold -- the overflow threshold value to use

flags -- bit map that controls the overflow mode of operation. This is currently not used and should be set to 0.

handler -- the handler function to call upon overflow

context -- a platform dependent structure containing information about the overflow event. This structure is typically returned automatically by the signal handler.

RETURN VALUES

On success, PAPI_overflow returns **PAPI_OK.**
On error, a non-zero error code is returned.
PAPI_get_overflow_address returns the instruction pointer where the overflow occured.
PAPI_get_overflow_address always returns the value at the offset in the *context* structure where the instruction pointer should be. No validity testing of this structure is done. If an invalid context pointer is passed to this function, the results will be undefined.

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

```
void handler(int EventSet, int EventCode, int EventIndex, long_long
*values, int *threshold, void *context)
{
  fprintf(stderr,"Value %lld at %p\n",
          values[EventIndex],PAPI_get_overflow_address(context));
#endif
  total++;
}

  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);

  /* Call handler every 100000 instructions */
```

```
  retval = PAPI_overflow(EventSet, PAPI_TOT_INS, THRESHOLD, 0,
handler);
  if (retval != PAPI_OK)
    exit(1);

  /* Start counting */

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_set_opt, PAPI_start, PAPI_rem_event, PAPI_rem_events, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_get_real_cyc, PAPI_get_real_usec - get real time counter values

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

```
long_long s, e;

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

s = PAPI_get_real_cyc();

your_slow_code();

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_virt_usec, PAPI_get_virt_cyc,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_get_real_cyc, PAPI_get_real_usec - get real time counter values

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

```
long_long s, e;

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

s = PAPI_get_real_cyc();

your_slow_code();

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_virt_usec, PAPI_get_virt_cyc,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_get_virt_cyc, PAPI_get_virt_usec - get virtual time counter values

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)
```

PAPI_get_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

```
long_long s, e;

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

s = PAPI_get_virt_cyc();

your_slow_code();

e = PAPI_get_virt_cyc();
```

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_real_usec, PAPI_get_real_cyc,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_get_virt_cyc, PAPI_get_virt_usec - get virtual time counter values

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)
```

PAPI_get_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

```
long_long s, e;

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

s = PAPI_get_virt_cyc();

your_slow_code();

e = PAPI_get_virt_cyc();
```

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_real_usec, PAPI_get_real_cyc,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_get_opt, PAPI_set_opt - get/set PAPI library or event set options
PAPIF_get_clockrate, PAPIF_get_domain, PAPIF_get_granularity, PAPIF_get_preload - Fortran implementation of some PAPI_get_opt options

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. 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_SET_DOMAIN, 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 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

Predefined name	Explanation
General information requests	
PAPI_GET_CLOCKRATE	Return clockrate in MHz.
PAPI_GET_MAX_CPUS	Return number of CPUs.
PAPI_GET_MAX_HWCTRS	Return number of counters.
PAPI_GET_EXEINFO	Addresses for text/data/bss.
PAPI_GET_HWINFO	Info. about hardware.
PAPI_GET_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
Defaults for the global library	
PAPI_GET_DEFDOM	Return default counting domain for newly created event sets.
PAPI_SET_DEFDOM	Set default counting domain.
PAPI_GET_DEFGRN	Return default counting granularity.
PAPI_SET_DEFGRN	Set default counting granularity.
PAPI_GET_DEBUG	Get the PAPI debug state. The available debug states are defined in The debug state is available in ptr->debug
PAPI_SET_DEBUG	Set the PAPI debug state
Multiplexing control	
PAPI_GET_MULTIPLEX	Get options for multiplexing. Currently not implemented.
PAPI_SET_MULTIPLEX	Set options for multiplexing
Manipulating individual event sets	
PAPI_GET_DOMAIN	Get domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_SET_DOMAIN	Set the domain for a single event set.
PAPI_GET_GRANUL	Get granularity for a single event set. The event set is specified in ptr->granularity.eventset
PAPI_SET_GRANUL	Set the granularity for a single event set.

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_inherit_option_t inherit;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_exe_info_t *exe_info; } PAPI_option_t;
```

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 num, EventSet = PAPI_NULL;
PAPI_option_t options;

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

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

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

/* 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_SET_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

There are no known bugs in these functions.

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_get_opt, PAPI_set_opt - get/set PAPI library or event set options
PAPIF_get_clockrate, PAPIF_get_domain, PAPIF_get_granularity, PAPIF_get_preload - Fortran implementation of some PAPI_get_opt options

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. 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_SET_DOMAIN, 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 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

Predefined name	Explanation
General information requests	
PAPI_GET_CLOCKRATE	Return clockrate in MHz.
PAPI_GET_MAX_CPUS	Return number of CPUs.
PAPI_GET_MAX_HWCTRS	Return number of counters.
PAPI_GET_EXEINFO	Addresses for text/data/bss.
PAPI_GET_HWINFO	Info. about hardware.
PAPI_GET_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
Defaults for the global library	
PAPI_GET_DEFDOM	Return default counting domain for newly created event sets.
PAPI_SET_DEFDOM	Set default counting domain.
PAPI_GET_DEFGRN	Return default counting granularity.
PAPI_SET_DEFGRN	Set default counting granularity.
PAPI_GET_DEBUG	Get the PAPI debug state. The available debug states are defined in The debug state is available in ptr->debug
PAPI_SET_DEBUG	Set the PAPI debug state
Multiplexing control	
PAPI_GET_MULTIPLEX	Get options for multiplexing. Currently not implemented.
PAPI_SET_MULTIPLEX	Set options for multiplexing
Manipulating individual event sets	
PAPI_GET_DOMAIN	Get domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_SET_DOMAIN	Set the domain for a single event set.
PAPI_GET_GRANUL	Get granularity for a single event set. The event set is specified in ptr->granularity.eventset
PAPI_SET_GRANUL	Set the granularity for a single event set.

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_inherit_option_t inherit;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_exe_info_t *exe_info; } PAPI_option_t;
```

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 num, EventSet = PAPI_NULL;
PAPI_option_t options;

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

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

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

/* 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_SET_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

There are no known bugs in these functions.

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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 C_LONG_LONG data_start,
           C_LONG_LONG  data_end,
         C_LONG_LONG C_LONG_LONG bss_end,
         C_STRING 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

lib_preload_env -- environment variable for preloading libraries

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

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

printf("Start of user program is at %p\n",prginfo->text_start);
printf("End of user program is at %p\n",prginfo->text_end);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

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

SEE ALSO

PAPI_library_init, PAPI_get_opt,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_get_opt, PAPI_set_opt - get/set PAPI library or event set options
PAPIF_get_clockrate, PAPIF_get_domain, PAPIF_get_granularity, PAPIF_get_preload - Fortran implementation of some PAPI_get_opt options

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. 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_SET_DOMAIN, 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 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

Predefined name	Explanation
General information requests	
PAPI_GET_CLOCKRATE	Return clockrate in MHz.
PAPI_GET_MAX_CPUS	Return number of CPUs.
PAPI_GET_MAX_HWCTRS	Return number of counters.
PAPI_GET_EXEINFO	Addresses for text/data/bss.
PAPI_GET_HWINFO	Info. about hardware.
PAPI_GET_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
Defaults for the global library	
PAPI_GET_DEFDOM	Return default counting domain for newly created event sets.
PAPI_SET_DEFDOM	Set default counting domain.
PAPI_GET_DEFGRN	Return default counting granularity.
PAPI_SET_DEFGRN	Set default counting granularity.
PAPI_GET_DEBUG	Get the PAPI debug state. The available debug states are defined in The debug state is available in ptr->debug
PAPI_SET_DEBUG	Set the PAPI debug state
Multiplexing control	
PAPI_GET_MULTIPLEX	Get options for multiplexing. Currently not implemented.
PAPI_SET_MULTIPLEX	Set options for multiplexing
Manipulating individual event sets	
PAPI_GET_DOMAIN	Get domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_SET_DOMAIN	Set the domain for a single event set.
PAPI_GET_GRANUL	Get granularity for a single event set. The event set is specified in ptr->granularity.eventset
PAPI_SET_GRANUL	Set the granularity for a single event set.

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_inherit_option_t inherit;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_exe_info_t *exe_info; } PAPI_option_t;
```

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 num, EventSet = PAPI_NULL;
PAPI_option_t options;

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

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

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

/* 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_SET_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

There are no known bugs in these functions.

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_get_opt, PAPI_set_opt - get/set PAPI library or event set options
PAPIF_get_clockrate, PAPIF_get_domain, PAPIF_get_granularity, PAPIF_get_preload - Fortran implementation of some PAPI_get_opt options

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. 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_SET_DOMAIN, 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 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

Predefined name	Explanation
General information requests	
PAPI_GET_CLOCKRATE	Return clockrate in MHz.
PAPI_GET_MAX_CPUS	Return number of CPUs.
PAPI_GET_MAX_HWCTRS	Return number of counters.
PAPI_GET_EXEINFO	Addresses for text/data/bss.
PAPI_GET_HWINFO	Info. about hardware.
PAPI_GET_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
Defaults for the global library	
PAPI_GET_DEFDOM	Return default counting domain for newly created event sets.
PAPI_SET_DEFDOM	Set default counting domain.
PAPI_GET_DEFGRN	Return default counting granularity.
PAPI_SET_DEFGRN	Set default counting granularity.
PAPI_GET_DEBUG	Get the PAPI debug state. The available debug states are defined in The debug state is available in ptr->debug
PAPI_SET_DEBUG	Set the PAPI debug state
Multiplexing control	
PAPI_GET_MULTIPLEX	Get options for multiplexing. Currently not implemented.
PAPI_SET_MULTIPLEX	Set options for multiplexing
Manipulating individual event sets	
PAPI_GET_DOMAIN	Get domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_SET_DOMAIN	Set the domain for a single event set.
PAPI_GET_GRANUL	Get granularity for a single event set. The event set is specified in ptr->granularity.eventset
PAPI_SET_GRANUL	Set the granularity for a single event set.

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_inherit_option_t inherit;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_exe_info_t *exe_info; } PAPI_option_t;
```

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 num, EventSet = PAPI_NULL;
PAPI_option_t options;

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

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

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

/* 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_SET_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

There are no known bugs in these functions.

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_describe_event, PAPI_event_code_to_name, PAPI_event_name_to_code, PAPI_label_event - convert a hardware event to a description, label, number or string symbol.

SYNOPSIS

C Interface

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

Fortran Interface

```
#include fpapi.h
PAPIF_describe_event(C_STRING EventName, C_INT EventCode, C_STRING
EventDescr, C_INT check)
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)
PAPIF_label_event(C_INT EventCode, C_STRING EventLabel, C_INT check)
```

DESCRIPTION

PAPI_describe_event() is used to translate either an ASCII PAPI preset name or an integer PAPI preset event code into the corresponding event code or name as well as an ASCII description of that event. If the *EventName* argument is a string of length >0 it is assumed to contain the name to look up and the corresponding event code is returned in the argument *EventCode*. Otherwise the *EventCode* argument is used to look up the event name which is stored in the *EventName* argument. Finally a descriptive string of length less than **PAPI_MAX_STR_LEN** is copied to the argument *EventDescr*.

The functionality of this call is a superset of the following two calls.

PAPI_event_name_to_code() is used to translate an ASCII PAPI preset name into an integer PAPI event code.

PAPI_event_code_to_name() is used to translate an integer PAPI event code into an ASCII PAPI preset name.

PAPI_label_event() is used to translate an integer PAPI event code into a short (<=18 character) ASCII label that is more descriptive than the preset name but shorter than the description. These labels can be used as event identifiers in third party tools.

ARGUMENTS

EventName -- a string containing the event name as listed in PAPI_presets

EventCode -- the numeric code for the event

EventDescr -- a descriptive string for the event of length less than **PAPI_MAX_STR_LEN**.

EventLabel -- a short descriptive label for the event of length less than 18 characters.

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.

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);

/* Describe event from EventCode */
*EventCodeStr=' '; /* Erase the name argument to look up the code */
if (PAPI_describe_event(EventCodeStr,&EventCode,EventDescr) != PAPI_OK)
  handle_error(1);

/* Label event from EventCode */
if (PAPI_label_event(EventCode,EventLabel) != PAPI_OK)
  handle_error(1);

/* Print the differing text strings */
printf("Name: %s0abel: %s0escription:
%s0,EventCodeStr,EventLabel,EventDescr);

/* 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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_add_event, PAPI_set_opt, PAPI_query_event,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_library_init - initialize the PAPI library

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

PAPI_library_init() initializes the PAPI library. It must be called before any 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()**.

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

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.

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_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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

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

SEE ALSO

PAPI_thread_init, PAPI_preset, PAPI_low_level, PAPI_high_level, PAPI,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_list_events - list the events defined in an event set

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.

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 larger than the actually stored number of counter codes.

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.

PAPI_ENOTPRESET

The hardware event specified is not a PAPI preset for this architecture.

EXAMPLES

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

  /* Convert to integer */

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

  /* Query Total Instructions Executed to our EventSet */

  if (PAPI_query(EventCode) != PAPI_OK)
    handle_error(1);

  /* Describe event from EventCode */

  if (PAPI_describe_event(EventCodeStr,&EventCode,EventDescr) !=
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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_preset, PAPI_set_opt, PAPI_start, PAPI_rem_event, PAPI_rem_events, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_lock - Lock the PAPI internal mutex variable
PAPI_unlock - Unlock the PAPI internal mutex variable

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

PAPI_lock() Grabs access to the PAPI mutex variable. This function is provided to the user to have a platform independent call to an (hopefully) efficiently implemented mutex.

PAPI_unlock() unlocks the mutex acquired by a call to **PAPI_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 PAPI mutex.

NOTES

Care must be taken since the mutex variable used is the internal PAPI mutex variable. The thread that has acquired the mutex should be designed so as not to call any PAPI library functions until a call to **PAPI_unlock** has been made. Also, other threads that call PAPI library functions may be blocked until the thread that has grabbed the mutex has released it.

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

Due to its possible interference with internal PAPI and that there is only one mutex available through this interface some care must be observed by the programmer using these calls.

SEE ALSO

The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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. This allows a user to count more events than there are 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

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.

EXAMPLES

```
/* Initialize the library */

retval = PAPI_library_init(PAPI_VER_CURRENT);
if (retval != PAPI_VER_CURRENT)
  handle_error("PAPI_library_init",__LINE__,retval);

/* Enable multiplexing support */

retval = PAPI_multiplex_init();
if (retval != PAPI_OK)
  handle_error("PAPI_multiplex_init",__LINE__,retval);

/* Turn on automatic error reporting */
```

```
  retval = PAPI_set_debug(PAPI_VERB_ECONT);
  if (retval != PAPI_OK)
    handle_error("PAPI_set_debug",__LINE__,retval);

  /* Turn on thread support in PAPI */

  if (PAPI_thread_init((unsigned long (*)(void))(pthread_self), 0) !=
PAPI_OK)
    handle_error("PAPI_thread_init",__LINE__,retval);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function causes the application to exit if it is called more than once.

SEE ALSO

PAPI_set_multiplex
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_num_counters - PAPI High Level: get 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()** initialises the library using **PAPI_library_init** 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 Events[2] = { PAPI_TOT_CYC, PAPI_TOT_INS };
int num_hwcntrs = 0;

if ((num_hwcntrs = PAPI_num_counters()) != PAPI_OK)
```

```
    handle_error(1);

  if (num_hwcntrs > 2)
    num_hwcntrs = 2;

  /* Start counting events */

  if (PAPI_start_counters(Events, num_hwcntrs) != PAPI_OK)
    handle_error(1);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

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

SEE ALSO

PAPI_start_counters, PAPI_read_counters, PAPI_stop_counters,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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 performance hardware. 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

```
int retval, num;

/* Initialize the library */

retval = PAPI_library_init(PAPI_VER_CURRENT);

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

/* Query the substrate for our resources. */

num = PAPI_num_hwctrs();
printf("%d hardware counters found.0,num);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

None.

SEE ALSO

PAPI_get_opt, PAPI,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_overflow, PAPI_get_overflow_address - set up an event set to begin registering overflows; return the address at which an overflow occured

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_overflow (int EventSet, int EventCode, int threshold, int
flags, PAPI_overflow_handler_t handler);
void *PAPI_get_overflow_address (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. Only one event in an event set can be used as an overflow trigger. Subsequent calls to **PAPI_overflow()** replace earlier calls. To turn off overflow, set the handler to NULL.

PAPI_get_overflow_address() returns the instruction pointer where overflow occured. This routine is often used as part of the overflow handler routine.

ARGUMENTS

EventSet -- a reference to the event set to use

EventCode -- the counter to be used for overflow detection

threshold -- the overflow threshold value to use

flags -- bit map that controls the overflow mode of operation. This is currently not used and should be set to 0.

handler -- the handler function to call upon overflow

context -- a platform dependent structure containing information about the overflow event. This structure is typically returned automatically by the signal handler.

RETURN VALUES

On success, PAPI_overflow returns **PAPI_OK.**
On error, a non-zero error code is returned.
PAPI_get_overflow_address returns the instruction pointer where the overflow occured.
PAPI_get_overflow_address always returns the value at the offset in the *context* structure where the instruction pointer should be. No validity testing of this structure is done. If an invalid context pointer is passed to this function, the results will be undefined.

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

```
void handler(int EventSet, int EventCode, int EventIndex, long_long
*values, int *threshold, void *context)
{
  fprintf(stderr,"Value %lld at %p\n",
          values[EventIndex],PAPI_get_overflow_address(context));
#endif
  total++;
}

  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);

  /* Call handler every 100000 instructions */
```

```
  retval = PAPI_overflow(EventSet, PAPI_TOT_INS, THRESHOLD, 0,
handler);
  if (retval != PAPI_OK)
    exit(1);

  /* Start counting */

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_set_opt, PAPI_start, PAPI_rem_event, PAPI_rem_events, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_perror, PAPI_strerror - convert PAPI error codes to strings

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.

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);
  }

/* Add native event (0xc1 on hardware counter 1) */

native = (0xc1 << 8) | 1;
if ((retval = PAPI_add_event(&EventSet, native)) != PAPI_OK)
  {
     /* Dump error string directly to stderr. */

      PAPI_perror(retval,NULL,NULL);
     exit(1);
  }

/* Start counting */

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_set_debug, PAPI_set_opt, PAPI_get_opt, PAPI_shutdown,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_profil, PAPI_sprofil - generate PC histogram data where hardware counter overflow occurs

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_profil(unsigned short * buf, unsigned bufsiz, unsigned long
offset,
        unsigned  int EventSet,
             int  EventCode,  int  threshold,  int  flags );
int PAPI_sprofil(PAPI_sprofil_t * prof, int profcnt, int EventSet,
        int int threshold, int flags );
```

Fortran Interface

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

DESCRIPTION

PAPI_profil() uses its first four parameters to create the data structures needed by PAPI_sprofil and then calls PAPI_sprofil to do the work.

PAPI_sprofil() assumes a preinitialized sprofil structure, and initiates profiling based on its values.

ARGUMENTS

**buf* -- pointer to profile buffer array.

bufsiz -- number of entries in *buf.

offset -- starting value of lowest memory address to profile.

scale -- scaling factor for bin values.

EventSet -- The PAPI EventSet to profile when it is started.

EventCode -- Code of the Event in the EventSet to profile.

threshold -- threshold value for the Event triggers the handler.

flags -- bit pattern to control profiling behavior. Defined values are shown in the table below.

**prof* -- pointer to PAPI_sprofil_t structure.

profcnt -- number of buffers for hardware profiling (*reserved*)

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 if hash buckets get big.

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;
int EventSet = PAPI_NULL;
unsigned long start, end, length;
PAPI_exe_info_t *prginfo;
unsigned short *profbuf;

retval = PAPI_library_init(PAPI_VER_CURRENT);
```

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

if (retval < 0)
  handle_error(retval);

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

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

profbuf = (unsigned short *)malloc(length*sizeof(unsigned short));
if (profbuf == NULL)
  handle_error(1);
memset(profbuf,0x00,length*sizeof(unsigned short));

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

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

if ((retval = PAPI_add_event(&EventSet, PAPI_FP_INS)) != PAPI_OK)
  handle_error(retval);

if ((retval = PAPI_profil(profbuf, length, start, 65536, EventSet,
                PAPI_FP_INS, 1000000, PAPI_PROFIL_POSIX)) != PAPI_OK)
  handle_error(retval);

/* Start counting */

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_get_executable_info, PAPI_set_opt, PAPI_start, PAPI_rem_event, PAPI_rem_events, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_query_event - query if PAPI event exists
PAPI_query_event_verbose - query details of a PAPI event
PAPI_query_all_events_verbose - acquire details of all PAPI events

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_query_event(int EventCode);
int PAPI_query_event_verbose(int EventCode, PAPI_preset_info_t *info);
const PAPI_preset_info_t* PAPI_query_all_events_verbose();
```

Fortran Interface

```
#include fpapi.h
PAPIF_query_event(C_INT EventCode, C_INT check)
PAPIF_query_event_verbose(C_INT EventCode, C_STRING EventName,
          C_STRING EventDescr,  C_STRING EventLabel,  C_INT  avail,
          C_STRING EventNote,  C_INT  flags,  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. On some platforms, this function also can be used to check the syntax of a native event.

PAPI_query_event_verbose() asks the PAPI library for a copy of an event descriptor. This descriptor can then be used to investigate the details about the event. In Fortran, the individual fields in the descriptor are returned as parameters.

PAPI_query_all_events_verbose() is implemented only in C. It asks the PAPI library to return a pointer to an array of event descriptors. The number of objects in the array is **PAPI_MAX_PRESET_EVENTS** and each object is a descriptor as returned by **PAPI_query_event_verbose().**

ARGUMENTS

EventCode -- a defined event such as PAPI_TOT_INS.

EventName -- the event name as described in PAPI_presets .

EventDescr -- a descriptive string for the event of length less than **PAPI_MAX_STR_LEN**.

EventLabel -- a short descriptive label for the event of length less than 18 characters.

avail -- zero if the event CANNOT be counted.

EventNote -- additional text information about an event if available.

flags -- provides additional information about an event, e.g., **PAPI_DERIVED** for an event derived from 2 or more other events.

RETURN VALUES

On success, **PAPI_query_event** and **PAPI_query_event_verbose** return **PAPI_OK ,** and on error, a non-zero error code is returned.

PAPI_query_all_events_verbose returns a pointer to an array of *PAPI_preset_info_t* structures on success, and a NULL pointer on error.

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 EventSet = PAPI_NULL;
unsigned int native = 0x0;
int retval,i;
PAPI_preset_info_t info;
PAPI_preset_info_t *infostructs;

/* 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);
  }

if (PAPI_query_event_verbose(PAPI_TOT_INS,&info) != PAPI_OK) {
```

```
    fprintf(stderr,"No instruction counter? How lame.\n");
    exit(1);
    }

  if(info.avail)
    printf("This event is available on this hardware.\n");

  if(info.flags & PAPI_DERIVED)
    printf("This event is a derived event on this hardware.\n");

  retval=0;
  infostructs=PAPI_query_all_events_verbose();
  if(infostructs)
    for(i=0;i<PAPI_MAX_PRESET_EVENTS;i++)
      if(infostructs[i].avail)
        retval += 1;
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_native, PAPI_set_opt, PAPI_start, PAPI_rem_event, PAPI_rem_events, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset, The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_query_event - query if PAPI event exists
PAPI_query_event_verbose - query details of a PAPI event
PAPI_query_all_events_verbose - acquire details of all PAPI events

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_query_event(int EventCode);
int PAPI_query_event_verbose(int EventCode, PAPI_preset_info_t *info);
const PAPI_preset_info_t* PAPI_query_all_events_verbose();
```

Fortran Interface

```
#include fpapi.h
PAPIF_query_event(C_INT EventCode, C_INT check)
PAPIF_query_event_verbose(C_INT EventCode, C_STRING EventName,
          C_STRING EventDescr,  C_STRING EventLabel,  C_INT  avail,
          C_STRING EventNote,  C_INT  flags,  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. On some platforms, this function also can be used to check the syntax of a native event.

PAPI_query_event_verbose() asks the PAPI library for a copy of an event descriptor. This descriptor can then be used to investigate the details about the event. In Fortran, the individual fields in the descriptor are returned as parameters.

PAPI_query_all_events_verbose() is implemented only in C. It asks the PAPI library to return a pointer to an array of event descriptors. The number of objects in the array is **PAPI_MAX_PRESET_EVENTS** and each object is a descriptor as returned by **PAPI_query_event_verbose().**

ARGUMENTS

EventCode -- a defined event such as PAPI_TOT_INS.

EventName -- the event name as described in PAPI_presets .

EventDescr -- a descriptive string for the event of length less than **PAPI_MAX_STR_LEN**.

EventLabel -- a short descriptive label for the event of length less than 18 characters.

avail -- zero if the event CANNOT be counted.

EventNote -- additional text information about an event if available.

flags -- provides additional information about an event, e.g., **PAPI_DERIVED** for an event derived from 2 or more other events.

RETURN VALUES

On success, **PAPI_query_event** and **PAPI_query_event_verbose** return **PAPI_OK ,** and on error, a non-zero error code is returned.

PAPI_query_all_events_verbose returns a pointer to an array of *PAPI_preset_info_t* structures on success, and a NULL pointer on error.

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 EventSet = PAPI_NULL;
unsigned int native = 0x0;
int retval,i;
PAPI_preset_info_t info;
PAPI_preset_info_t *infostructs;

/* 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);
  }

if (PAPI_query_event_verbose(PAPI_TOT_INS,&info) != PAPI_OK) {
```

```
    fprintf(stderr,"No instruction counter? How lame.\n");
    exit(1);
    }

  if(info.avail)
    printf("This event is available on this hardware.\n");

  if(info.flags & PAPI_DERIVED)
    printf("This event is a derived event on this hardware.\n");

  retval=0;
  infostructs=PAPI_query_all_events_verbose();
  if(infostructs)
    for(i=0;i<PAPI_MAX_PRESET_EVENTS;i++)
      if(infostructs[i].avail)
        retval += 1;
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_native, PAPI_set_opt, PAPI_start, PAPI_rem_event, PAPI_rem_events, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset, The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_query_event - query if PAPI event exists
PAPI_query_event_verbose - query details of a PAPI event
PAPI_query_all_events_verbose - acquire details of all PAPI events

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_query_event(int EventCode);
int PAPI_query_event_verbose(int EventCode, PAPI_preset_info_t *info);
const PAPI_preset_info_t* PAPI_query_all_events_verbose();
```

Fortran Interface

```
#include fpapi.h
PAPIF_query_event(C_INT EventCode, C_INT check)
PAPIF_query_event_verbose(C_INT EventCode, C_STRING EventName,
          C_STRING EventDescr,  C_STRING EventLabel,  C_INT  avail,
          C_STRING EventNote,  C_INT  flags,  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. On some platforms, this function also can be used to check the syntax of a native event.

PAPI_query_event_verbose() asks the PAPI library for a copy of an event descriptor. This descriptor can then be used to investigate the details about the event. In Fortran, the individual fields in the descriptor are returned as parameters.

PAPI_query_all_events_verbose() is implemented only in C. It asks the PAPI library to return a pointer to an array of event descriptors. The number of objects in the array is **PAPI_MAX_PRESET_EVENTS** and each object is a descriptor as returned by **PAPI_query_event_verbose().**

ARGUMENTS

EventCode -- a defined event such as PAPI_TOT_INS.

EventName -- the event name as described in PAPI_presets .

EventDescr -- a descriptive string for the event of length less than **PAPI_MAX_STR_LEN**.

EventLabel -- a short descriptive label for the event of length less than 18 characters.

avail -- zero if the event CANNOT be counted.

EventNote -- additional text information about an event if available.

flags -- provides additional information about an event, e.g., **PAPI_DERIVED** for an event derived from 2 or more other events.

RETURN VALUES

On success, **PAPI_query_event** and **PAPI_query_event_verbose** return **PAPI_OK ,** and on error, a non-zero error code is returned.

PAPI_query_all_events_verbose returns a pointer to an array of *PAPI_preset_info_t* structures on success, and a NULL pointer on error.

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 EventSet = PAPI_NULL;
unsigned int native = 0x0;
int retval,i;
PAPI_preset_info_t info;
PAPI_preset_info_t *infostructs;

/* 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);
  }

if (PAPI_query_event_verbose(PAPI_TOT_INS,&info) != PAPI_OK) {
```

```
    fprintf(stderr,"No instruction counter? How lame.\n");
    exit(1);
    }

  if(info.avail)
    printf("This event is available on this hardware.\n");

  if(info.flags & PAPI_DERIVED)
    printf("This event is a derived event on this hardware.\n");

  retval=0;
  infostructs=PAPI_query_all_events_verbose();
  if(infostructs)
    for(i=0;i<PAPI_MAX_PRESET_EVENTS;i++)
      if(infostructs[i].avail)
        retval += 1;
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_native, PAPI_set_opt, PAPI_start, PAPI_rem_event, PAPI_rem_events, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset, The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_read, PAPI_accum - read hardware events, accumulate and reset hardware events from 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

PAPI_read() copies the counters of the indicated event set into the array *values*. The counters are left counting after the read.

PAPI_accum() adds the counters of the indicated event set into the array *values*. The counters are zeroed and left counting after the operation.

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

```
int EventSet = PAPI_NULL;
unsigned int native = 0x0;
long_long values[1] = (long_long) 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_start(EventSet) != PAPI_OK)
  handle_error(1);

poorly_tuned_function();

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

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

poorly_tuned_function();

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

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_add_event, PAPI_reset, PAPI_set_opt, PAPI_rem_event, PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_read_counters, PAPI_accum_counters - PAPI High Level: read counting hardware events

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.

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

```
int Events[2] = { PAPI_TOT_CYC, PAPI_TOT_INS };
long_long values[2];
int num_hwcntrs = 0;

if ((num_hwcntrs = PAPI_num_counters()) != PAPI_OK)
  handle_error(1);

if (num_hwcntrs > 2)
  num_hwcntrs = 2;

/* Start counting events */

if (PAPI_start_counters(Events, num_hwcntrs) != PAPI_OK)
  handle_error(1);

your_slow_code();

/* Start counting events */

if (PAPI_read_counters(values, num_hwcntrs) != PAPI_OK)
  handle_error(1);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_num_counters, PAPI_start_counters, PAPI_stop_counters,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_rem_event, PAPI_rem_events, PAPI_rem_pevent - remove PAPI preset or native hardware event from an EventSet

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_rem_event(int *EventSet, int EventCode);
int PAPI_rem_events(int *EventSet, int *EventCode, int number);
int PAPI_rem_pevent(int *EventSet, int code, void * misc);
```

Fortran Interface

```
#include fpapi.h
PAPIF_rem_event(C_INT EventSet, C_INT EventCode, C_INT check)
PAPIF_rem_events(C_INT EventSet, C_INT(*) EventCode, C_INT number,
C_INT check)
```

DESCRIPTION

PAPI_rem_event() removes a hardware event to a PAPI event set. **PAPI_rem_events()** does the same, but for an array of hardware event codes. **PAPI_rem_pevent()** is reserved for future use.

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*

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 the encoding of native events, see the <arch>.README file also included the distribution.

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.

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_rem_event(&EventSet, PAPI_TOT_INS) != PAPI_OK)
  handle_error(1);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_set_opt, PAPI_start, PAPI_add_event, PAPI_add_events, PAPI_add_pevents, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset, The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_rem_event, PAPI_rem_events, PAPI_rem_pevent - remove PAPI preset or native hardware event from an EventSet

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_rem_event(int *EventSet, int EventCode);
int PAPI_rem_events(int *EventSet, int *EventCode, int number);
int PAPI_rem_pevent(int *EventSet, int code, void * misc);
```

Fortran Interface

```
#include fpapi.h
PAPIF_rem_event(C_INT EventSet, C_INT EventCode, C_INT check)
PAPIF_rem_events(C_INT EventSet, C_INT(*) EventCode, C_INT number,
C_INT check)
```

DESCRIPTION

PAPI_rem_event() removes a hardware event to a PAPI event set. **PAPI_rem_events()** does the same, but for an array of hardware event codes. **PAPI_rem_pevent()** is reserved for future use.

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*

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 the encoding of native events, see the <arch>.README file also included the distribution.

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.

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_rem_event(&EventSet, PAPI_TOT_INS) != PAPI_OK)
  handle_error(1);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_set_opt, PAPI_start, PAPI_add_event, PAPI_add_events, PAPI_add_pevents, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset, The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_rem_event, PAPI_rem_events, PAPI_rem_pevent - remove PAPI preset or native hardware event from an EventSet

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_rem_event(int *EventSet, int EventCode);
int PAPI_rem_events(int *EventSet, int *EventCode, int number);
int PAPI_rem_pevent(int *EventSet, int code, void * misc);
```

Fortran Interface

```
#include fpapi.h
PAPIF_rem_event(C_INT EventSet, C_INT EventCode, C_INT check)
PAPIF_rem_events(C_INT EventSet, C_INT(*) EventCode, C_INT number,
C_INT check)
```

DESCRIPTION

PAPI_rem_event() removes a hardware event to a PAPI event set. **PAPI_rem_events()** does the same, but for an array of hardware event codes. **PAPI_rem_pevent()** is reserved for future use.

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*

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 the encoding of native events, see the <arch>.README file also included the distribution.

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.

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_rem_event(&EventSet, PAPI_TOT_INS) != PAPI_OK)
  handle_error(1);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_set_opt, PAPI_start, PAPI_add_event, PAPI_add_events, PAPI_add_pevents, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset, The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_save - Save the current state of the PAPI library
PAPI_restore - Restore the saved state of the PAPI library

SYNOPSIS

C Interface

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

DESCRIPTION

PAPI_save() "freezes" the current state of the PAPI_library. Among tasks done is to register the current values of any running virtual counters to be able to restore these at the time of a subsequent **PAPI_restore**.

PAPI_restore() restores the current state of the PAPI_library to a state that has previously been saved using **PAPI_save**.

RETURN VALUES

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

ERRORS

PAPI_ESBSTR

The substrate does not provide the save/restore functionality.

PAPI_EINVAL

A **PAPI_restore** was attempted, but there is no saved state to restore.

EXAMPLES

```
int retval;

retval = PAPI_save();
THIRD_PARTY_TOOL_update_gui_counters();
retval = PAPI_restore();
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions are not yet implemented.
NOTE: As currently defined, these functions are not necessarily nestable or thread safe. To become so, PAPI_save should return a void pointer to a platform specific structure that can be used by PAPI_restore to restore data on a per thread or per EventSet basis.

SEE ALSO

The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_reset - reset the hardware event counts in an EventSet

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*.

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

```
int EventSet = PAPI_NULL;
unsigned int native = 0x0;
long_long values[1];

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]);

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_add_event, PAPI_reset, PAPI_read, PAPI_set_opt, PAPI_rem_event, PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_save - Save the current state of the PAPI library
PAPI_restore - Restore the saved state of the PAPI library

SYNOPSIS

C Interface

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

DESCRIPTION

PAPI_save() "freezes" the current state of the PAPI_library. Among tasks done is to register the current values of any running virtual counters to be able to restore these at the time of a subsequent **PAPI_restore**.

PAPI_restore() restores the current state of the PAPI_library to a state that has previously been saved using **PAPI_save**.

RETURN VALUES

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

ERRORS

PAPI_ESBSTR

The substrate does not provide the save/restore functionality.

PAPI_EINVAL

A **PAPI_restore** was attempted, but there is no saved state to restore.

EXAMPLES

```
int retval;

retval = PAPI_save();
THIRD_PARTY_TOOL_update_gui_counters();
retval = PAPI_restore();
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions are not yet implemented.
NOTE: As currently defined, these functions are not necessarily nestable or thread safe. To become so, PAPI_save should return a void pointer to a platform specific structure that can be used by PAPI_restore to restore data on a per thread or per EventSet basis.

SEE ALSO

The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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 default debug level for 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 internally stored in the PAPI library and is used by the default internal PAPI error handler subroutine. The error handler is called by library routines on the occurence of recoverable errors. The default PAPI error handler handles the possible debug levels shown in the table below.

PAPI_QUIET	Quietly handle errors
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

One or more of the arguments is invalid.

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

The default error handler is currently not possible to substitute by the user. It is also only activated when PAPI is compiled with debugging enabled.

SEE ALSO

PAPI_set_opt,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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_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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_set_opt,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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)
```

PAPI_set_granularity

DESCRIPTION

This function is currently unimplemented.

RETURN VALUES

ERRORS

EXAMPLES

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function is currently unimplemented.

SEE ALSO

PAPI_set_domain, PAPI_set_opt, PAPI_get_opt,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_set_multiplex - convert a standard event set to a multiplexed event set

SYNOPSIS

C Interface

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

Fortran Interface

```
#include fpapi.h
PAPIF_set_multiplex(C_INT EventSet)
```

DESCRIPTION

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 eventset 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* -- a pointer to 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_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.

EXAMPLES

```
int retval, i, EventSet = PAPI_NULL, max_to_add = 6, j = 0;
long_long *values;
const PAPI_preset_info_t *pset;

/* Initialize the library */

retval = PAPI_library_init(PAPI_VER_CURRENT);
if (retval != PAPI_VER_CURRENT)
  handle_error(1);

pset = PAPI_query_all_events_verbose();
if (pset == NULL)
  handle_error(1);

if (PAPI_multiplex_init() != PAPI_OK)
  handle_error(1);

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

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

for (i=0;i<PAPI_MAX_PRESET_EVENTS;i++)
{
  if ((pset->avail) && (pset->event_code != PAPI_TOT_CYC))
      {
        retval = ;
        if (PAPI_add_event(&EventSet, pset->event_code) != PAPI_OK)
      handle_error(1);
        if (++j >= max_to_add)
          break;
      }
      pset++;
}

values = (long_long *)malloc(max_to_add*sizeof(long_long));
if (values == NULL)
  handle_error(1);

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_multiplex_init, PAPI_set_opt,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_get_opt, PAPI_set_opt - get/set PAPI library or event set options
PAPIF_get_clockrate, PAPIF_get_domain, PAPIF_get_granularity, PAPIF_get_preload - Fortran implementation of some PAPI_get_opt options

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. 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_SET_DOMAIN, 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 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

Predefined name	Explanation
General information requests	
PAPI_GET_CLOCKRATE	Return clockrate in MHz.
PAPI_GET_MAX_CPUS	Return number of CPUs.
PAPI_GET_MAX_HWCTRS	Return number of counters.
PAPI_GET_EXEINFO	Addresses for text/data/bss.
PAPI_GET_HWINFO	Info. about hardware.
PAPI_GET_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
Defaults for the global library	
PAPI_GET_DEFDOM	Return default counting domain for newly created event sets.
PAPI_SET_DEFDOM	Set default counting domain.
PAPI_GET_DEFGRN	Return default counting granularity.
PAPI_SET_DEFGRN	Set default counting granularity.
PAPI_GET_DEBUG	Get the PAPI debug state. The available debug states are defined in The debug state is available in ptr->debug
PAPI_SET_DEBUG	Set the PAPI debug state
Multiplexing control	
PAPI_GET_MULTIPLEX	Get options for multiplexing. Currently not implemented.
PAPI_SET_MULTIPLEX	Set options for multiplexing
Manipulating individual event sets	
PAPI_GET_DOMAIN	Get domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_SET_DOMAIN	Set the domain for a single event set.
PAPI_GET_GRANUL	Get granularity for a single event set. The event set is specified in ptr->granularity.eventset
PAPI_SET_GRANUL	Set the granularity for a single event set.

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_inherit_option_t inherit;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_exe_info_t *exe_info; } PAPI_option_t;
```

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 num, EventSet = PAPI_NULL;
PAPI_option_t options;

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

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

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

/* 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_SET_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

There are no known bugs in these functions.

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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_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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_set_opt,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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 the capability to free memory and resources used by the PAPI Library.

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_profil, PAPI_sprofil - generate PC histogram data where hardware counter overflow occurs

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_profil(unsigned short * buf, unsigned bufsiz, unsigned long
offset,
        unsigned  int EventSet,
             int  EventCode,  int  threshold,  int  flags );
int PAPI_sprofil(PAPI_sprofil_t * prof, int profcnt, int EventSet,
        int int threshold, int flags );
```

Fortran Interface

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

DESCRIPTION

PAPI_profil() uses its first four parameters to create the data structures needed by PAPI_sprofil and then calls PAPI_sprofil to do the work.

PAPI_sprofil() assumes a preinitialized sprofil structure, and initiates profiling based on its values.

ARGUMENTS

**buf* -- pointer to profile buffer array.

bufsiz -- number of entries in *buf.

offset -- starting value of lowest memory address to profile.

scale -- scaling factor for bin values.

EventSet -- The PAPI EventSet to profile when it is started.

EventCode -- Code of the Event in the EventSet to profile.

threshold -- threshold value for the Event triggers the handler.

flags -- bit pattern to control profiling behavior. Defined values are shown in the table below.

**prof* -- pointer to PAPI_sprofil_t structure.

profcnt -- number of buffers for hardware profiling (*reserved*)

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 if hash buckets get big.

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;
int EventSet = PAPI_NULL;
unsigned long start, end, length;
PAPI_exe_info_t *prginfo;
unsigned short *profbuf;

retval = PAPI_library_init(PAPI_VER_CURRENT);
```

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

if (retval < 0)
  handle_error(retval);

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

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

profbuf = (unsigned short *)malloc(length*sizeof(unsigned short));
if (profbuf == NULL)
  handle_error(1);
memset(profbuf,0x00,length*sizeof(unsigned short));

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

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

if ((retval = PAPI_add_event(&EventSet, PAPI_FP_INS)) != PAPI_OK)
  handle_error(retval);

if ((retval = PAPI_profil(profbuf, length, start, 65536, EventSet,
                PAPI_FP_INS, 1000000, PAPI_PROFIL_POSIX)) != PAPI_OK)
  handle_error(retval);

/* Start counting */

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_get_executable_info, PAPI_set_opt, PAPI_start, PAPI_rem_event, PAPI_rem_events, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_start, PAPI_stop - start/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 the counting of a previously defined event set

PAPI_stop halts the counting of a previously defined 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_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLES

```
int EventSet = PAPI_NULL;
unsigned int native = 0x0;
long_long values[1];

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]);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI_destroy_eventset, PAPI_add_event, PAPI_rem_event, PAPI_reset, PAPI_set_opt,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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 implicitly stops and initializes any counters running as a result of a previous call to PAPI_start_counters(). 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_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

```
int Events[2] = { PAPI_TOT_CYC, PAPI_TOT_INS };
int num_hwcntrs = 0;

if ((num_hwcntrs = PAPI_num_counters()) != PAPI_OK)
  handle_error(1);

if (num_hwcntrs > 2)
  num_hwcntrs = 2;

/* Start counting events */

if (PAPI_start_counters(Events, num_hwcntrs) != PAPI_OK)
  handle_error(1);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_num_counters, PAPI_read_counters, PAPI_stop_counters,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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	EventSet has accumulating enabled

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);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_start, PAPI_stop,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_start, PAPI_stop - start/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 the counting of a previously defined event set

PAPI_stop halts the counting of a previously defined 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_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLES

```
int EventSet = PAPI_NULL;
unsigned int native = 0x0;
long_long values[1];

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]);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI_destroy_eventset, PAPI_add_event, PAPI_rem_event, PAPI_reset, PAPI_set_opt,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_stop_counters - PAPI High Level: stop counting hardware events

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 returns their values. The counters must have been started by a previous call to PAPI_start_counters().

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.

EXAMPLES

```
int Events[2] = { PAPI_TOT_CYC, PAPI_TOT_INS };
long_long values[2];
int num_hwcntrs = 0;
```

```
if ((num_hwcntrs = PAPI_num_counters()) < PAPI_OK)
  handle_error(1);

if (num_hwcntrs > 2)
  num_hwcntrs = 2;

/* Start counting events */

if (PAPI_start_counters(Events, num_hwcntrs) != PAPI_OK)
  handle_error(1);

your_slow_code();

/* Stop counting events */

if (PAPI_stop_counters(values, num_hwcntrs) != PAPI_OK)
  handle_error(1);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_num_counters, PAPI_start_counters, PAPI_read_counters,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_perror, PAPI_strerror - convert PAPI error codes to strings

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.

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);
  }

/* Add native event (0xc1 on hardware counter 1) */

native = (0xc1 << 8) | 1;
if ((retval = PAPI_add_event(&EventSet, native)) != PAPI_OK)
  {
     /* Dump error string directly to stderr. */

      PAPI_perror(retval,NULL,NULL);
     exit(1);
  }

/* Start counting */

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

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_set_debug, PAPI_set_opt, PAPI_get_opt, PAPI_shutdown,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_thread_id - get the thread identifier of the current thread

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

None.

RETURN VALUES

On success, this function returns a valid thread identifier.
On error, (unsigned long int)-1 is returned.

ERRORS

None.

EXAMPLE

```
unsigned long int tid;

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

if (PAPI_thread_init(pthread_self, 0) != PAPI_OK)
  exit(1);

if ((tid = PAPI_thread_id()) == (unsigned long int)-1)
  exit(1);

printf("Initial thread id is: %lu\n",tid);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function has no known bugs.

SEE ALSO

PAPI_thread_init,
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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)(), int flag);
```

Fortran Interface

```
#include fpapi.h
PAPIF_thread_init(C_INT FUNCTION handle, C_INT flag, C_INT check)
```

DESCRIPTION

PAPI_thread_init initializes thread support in the PAPI library. Itshould be called **only once**, just after PAPI_library_init, and before any other PAPI calls. Applications that make no use of threads do not need to call this routine.

ARGUMENTS

handle -- Pointer to a function that returns current thread ID.

flag -- This is reserved for future use and should be set to zero.

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.

EXAMPLES

For Pthreads:

```
if (PAPI_thread_init(pthread_self, 0) != PAPI_OK)
  handle_error(1);
```

For OpenMP:

```
if (PAPI_thread_init(omp_get_thread_num,0) != PAPI_OK)
  handle_error(1);
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

This function causes the application to exit if it is called more than once.

SEE ALSO

The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

NAME

PAPI_lock - Lock the PAPI internal mutex variable
PAPI_unlock - Unlock the PAPI internal mutex variable

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

PAPI_lock() Grabs access to the PAPI mutex variable. This function is provided to the user to have a platform independent call to an (hopefully) efficiently implemented mutex.

PAPI_unlock() unlocks the mutex acquired by a call to **PAPI_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 PAPI mutex.

NOTES

Care must be taken since the mutex variable used is the internal PAPI mutex variable. The thread that has acquired the mutex should be designed so as not to call any PAPI library functions until a call to **PAPI_unlock** has been made. Also, other threads that call PAPI library functions may be blocked until the thread that has grabbed the mutex has released it.

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

Due to its possible interference with internal PAPI and that there is only one mutex available through this interface some care must be observed by the programmer using these calls.

SEE ALSO

The PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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* in to 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 write the values to the running counters as well. 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

The EventSet is currently counting events and the substrate could not change the values of the running counters.

EXAMPLES

```
/* Yet to be written */
```

AUTHOR

Philip J. Mucci <mucci@cs.utk.edu>

BUGS

Theis function has no known bugs.

SEE ALSO

PAPI_accum
The PAPI Web Site: http://icl.cs.utk.edu/projects/papi