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

PAPI Native Events

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

High Level Functions

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

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

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

Low Level Functions

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

PAPI_accum - accumulate and reset hardware events from an event set
PAPI_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_cleanup_eventset - remove all PAPI events from an event set
PAPI_create_eventset - create a new empty PAPI event set
PAPI_destroy_eventset - deallocates memory associated with an empty PAPI event set
PAPI_enum_event - return the event code for the next available preset or natvie event
PAPI_event_code_to_name - translate an integer PAPI event code into an ASCII PAPI preset or native name
PAPI_event_name_to_code - translate an ASCII PAPI preset or native name into an integer PAPI event code
PAPI_get_dmem_info - get information about the dynamic memory usage of the current program
PAPI_get_event_info - get the name and descriptions for a given preset or native 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_multiplex - get the multiplexing status of specified event set
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_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_shared_lib_info - get information about the shared libraries used by the process
PAPI_get_thr_specific - return a pointer to a thread specific stored data structure
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_is_initialized - return the initialized state of the PAPI library
PAPI_library_init - initialize the PAPI library
PAPI_list_events - list the events defined in an event set
PAPI_lock - lock one of two PAPI internal user mutex variables
PAPI_multiplex_init - initialize multiplex support in the PAPI library ror code
PAPI_num_hwctrs - return the number of hardware counters
PAPI_num_events - return the number of events in an event set

PAPI_overflow - set up an event set to begin registering overflows
PAPI_perror - convert PAPI error codes to strings
PAPI_profil - generate PC histogram data where hardware counter overflow occurs
PAPI_query_event - query if a PAPI event exists
PAPI_read - read hardware events from an event set with no reset
PAPI_register_thread - read hardware events from an event set with no reset
PAPI_remove_event - remove a hardware event from a PAPI event set
PAPI_remove_events - remove an array of hardware events from a PAPI event set
PAPI_reset - reset the hardware event counts in an event set
PAPI_set_debug - set the current debug level for PAPI
PAPI_set_domain - set the default execution domain for new event sets
PAPIF_set_event_domain - set the execution domain for a specific event set. Fortran only.
PAPI_set_granularity - set the default granularity for new event sets
PAPI_set_multiplex - convert a standard event set to a multiplexed event set
PAPI_set_opt - change the option settings of the PAPI library or a specific event set
PAPI_set_thr_specific - save a pointer as a thread specific stored data structure
PAPI_shutdown - finish using PAPI and free all related resources
PAPI_sprofil - generate 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 one of two PAPI internal user mutex variables
PAPI_write - write counter values into counters

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

SEE ALSO

PAPIF, PAPI_presets, PAPI_native

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.

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

DIAGNOSTICS

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

SEE ALSO

The PAPI Interface: PAPI

NAME

PAPI_presets - PAPI predefined named events

SYNOPSIS

#include <papi.h>

DESCRIPTION

The PAPI library names a number of predefined events. This set is a collection of events typically found in many CPUs that provide performance counters. A PAPI preset event name is mapped onto one or more of the countable native events on each hardware platform. On any particular platform, the preset can either be directly available as a single counter, derived using a combination of counters or unavailable.

The PAPI preset events can be broken loosely into several categories, as shown in the table below:

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_FMA_INS	FMA instructions completed
PAPI_FML_INS	Floating point multiply instructions
PAPI_FNV_INS	Floating point inverse instructions
PAPI_FP_INS	Floating point instructions
PAPI_FP_OPS	Floating point operations
PAPI_FP_STAL	Cycles the FP unit
PAPI_FPU_IDL	Cycles floating point units are idle
PAPI_FSQ_INS	Floating point square root instructions
Instruction Counting:	
PAPI_FUL_CCY	Cycles with maximum instructions completed
PAPI_FUL_ICY	Cycles with maximum instruction issue
PAPI_FXU_IDL	Cycles integer units are idle
PAPI_HW_INT	Hardware interrupts
PAPI_INT_INS	Integer instructions
PAPI_TOT_CYC	Total cycles
PAPI_TOT_IIS	Instructions issued
PAPI_TOT_INS	Instructions completed
PAPI_VEC_INS	Vector/SIMD instructions
Cache Access:	
PAPI_L1_DCA	L1 data cache accesses
PAPI_L1_DCH	L1 data cache hits
PAPI_L1_DCM	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

AUTHORS

Nils Smeds <smeds@cs.utk.edu>

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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_native, PAPI_enum_event, PAPI_get_event_info,
PAPI_event_code_to_name, PAPI_event_name_to_code

NAME

PAPI_native - Accessing PAPI native events

SYNOPSIS

#include <papi.h>

DESCRIPTION

In addition to the predefined PAPI preset events, the PAPI library also exposes a majority of the events native to each platform. Native events form the basic building blocks for PAPI presets. They can also be used directly to access functions specific to a given platform.

Since native events are *by definition* specific to each platform, the names for these events are unique to each platform. Native events for a given platform can be discovered by combining the PAPI_enum_event and PAPI_event_code_to_name or PAPI_get_event_info functions.

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI, PAPI_presets, PAPI_enum_event, PAPI_get_event_info, PAPI_event_code_to_name, PAPI_event_name_to_code

NAME

PAPI_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 continue counting after the read.

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

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]);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_add_event, PAPI_reset, PAPI_set_opt, PAPI_remove_event,
PAPI_cleanup_eventset, PAPI_destroy_eventset,

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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_num_counters, PAPI_start_counters, PAPI_stop_counters

NAME

PAPI_add_event, PAPI_add_events - 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 *EventCodes, int number);
```

Fortran Interface

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

DESCRIPTION

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

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 a list of native events available on current platform, run native_avail test case in the PAPI distribution. For the encoding of native events, see PAPI_event_name_to_code to learn how to generate native code for the supported native event on the underlying architecture.

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) */

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

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

/* Start counting */

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_set_opt, PAPI_start, PAPI_remove_event, PAPI_remove_events, PAPI_query_event, PAPI_cleanup_eventset, PAPI_destroy_eventset, PAPI_event_code_to_name

NAME

PAPI_add_event, PAPI_add_events - 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 *EventCodes, int number);
```

Fortran Interface

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

DESCRIPTION

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

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 a list of native events available on current platform, run native_avail test case in the PAPI distribution. For the encoding of native events, see PAPI_event_name_to_code to learn how to generate native code for the supported native event on the underlying architecture.

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) */

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

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

/* Start counting */

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_set_opt, PAPI_start, PAPI_remove_event, PAPI_remove_events, PAPI_query_event, PAPI_cleanup_eventset, PAPI_destroy_eventset, PAPI_event_code_to_name

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 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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI_query_event, PAPI_add_event, PAPI_start,
PAPI_remove_event, PAPI_remove_events, PAPI_shutdown

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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_destroy_eventset, PAPI_cleanup_eventset, PAPI_remove_event, PAPI_remove_events, PAPI_add_event, PAPI_add_events

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 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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI_query_event, PAPI_add_event, PAPI_start, PAPI_remove_event, PAPI_remove_events, PAPI_shutdown

NAME

`PAPI_enum_event - enumerate PAPI preset or native events`

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

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

ARGUMENTS

EventCode -- a defined preset or native event such as PAPI_TOT_INS.

modifier -- modifies the search logic. For preset events, TRUE specifies available events only. For native events, each platform behaves differently. See platform-specific documentation for details

PENTIUM 4

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

POWER 4

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

RETURN VALUES

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

ERRORS

PAPI_EINVAL

One or more of the arguments (typically *modifier)* is invalid.

PAPI_ENOEVNT

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

EXAMPLES

```
   int i = 0 | NATIVE_MASK;
   int retval;
   PAPI_event_info_t info;

/* Initialize the library */

retval = PAPI_library_init(PAPI_VER_CURRENT);

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

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

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_preset, PAPI_native, PAPI_get_event_info, PAPI_library_init

NAME

PAPI_event_code_to_name, PAPI_event_name_to_code - convert a numeric hardware event code to a name, or vice versa.

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

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

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

ARGUMENTS

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

EventCode -- the numeric code for the event

* * Check the return values and example code. *

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);

/* 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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_native, PAPI_enum_events, PAPI_add_event, PAPI_set_opt, PAPI_get_event_info

NAME

PAPI_event_code_to_name, PAPI_event_name_to_code - convert a numeric hardware event code to a name, or vice versa.

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

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

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

ARGUMENTS

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

EventCode -- the numeric code for the event

* * Check the return values and example code. *

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);

/* 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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_presets, PAPI_native, PAPI_enum_events, PAPI_add_event, PAPI_set_opt, PAPI_get_event_info

NAME

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

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

The first call to **PAPI_flips() or PAPI_flops()** will initialize the PAPI High Level interface, set up the counters to monitor PAPI_FP_INS or PAPI_FP_OPS and PAPI_TOT_CYC events and start the counters. Subsequent calls will read the counters and return total real time, total process time, total floating point instructions or operations since the start of the measurement and the Mflip/s or Mflop/s rate since latest call to **PAPI_flops() or PAPI_flops().**

ARGUMENTS

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

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

**flpins, flpops* -- total floating point instructions or operations since the first call

**mflips, *mflops* -- Mflip/s or Mflop/s achieved since the previous call

RETURN VALUES

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

NOTES

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

`PAPI_flops()` `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()
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_accum , PAPI_ipc

NAME

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

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

The first call to **PAPI_flips() or PAPI_flops()** will initialize the PAPI High Level interface, set up the counters to monitor PAPI_FP_INS or PAPI_FP_OPS and PAPI_TOT_CYC events and start the counters. Subsequent calls will read the counters and return total real time, total process time, total floating point instructions or operations since the start of the measurement and the Mflip/s or Mflop/s rate since latest call to **PAPI_flops() or PAPI_flops().**

ARGUMENTS

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

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

**flpins, flpops* -- total floating point instructions or operations since the first call

**mflips, *mflops* -- Mflip/s or Mflop/s achieved since the previous call

RETURN VALUES

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

NOTES

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

`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()
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_accum , PAPI_ipc

NAME

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

SYNOPSIS

C Interface

```
#include <papi.h>
 long PAPI_get_dmem_info(int option);
```

DESCRIPTION

In C, this function returns a value specified by the option. There is no Fortran equivalent call.

NOTE

The exact calling syntax and returned information for this function is likely to change.

ARGUMENTS

option -- Can currently be one of the following: **PAPI_GET_SIZE** (Size of process image in pages), **PAPI_GET_RESSIZE** (Resident set size in pages), **PAPI_GET_PAGESIZE** (Pagesize in bytes).

RETURN VALUES

On success, this function returns the positive value of the specified option. On error a negative error value is returned.

ERRORS

PAPI_EINVAL

The value of *option* is invalid.

PAPI_SYS

A system error occured.

EXAMPLE

```
int retval;

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

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

   printf("Resident Size in Pages: %ld  Size in Pages: %ld  Pagesize in
bytes: %ld0,
          PAPI_get_dmem_info(PAPI_GET_RESSIZE),
          PAPI_get_dmem_info(PAPI_GET_SIZE),
          PAPI_get_dmem_info(PAPI_GET_PAGESIZE));
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_library_init, PAPI_get_opt, PAPI_get_hardware_info, PAPI_get_executable_info

NAME

PAPI_get_event_info - get the event's name and description info

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

In C, this function fills the event information into a structure. In Fortran, some fields of the structure are returned explicitly.

ARGUMENTS

The following arguments are implicit in the structure returned by the C function, or explicitly returned by Fortran.

EventCode -- event code(preset or native)

info -- structure with the event information

symbol -- whether the preset is part of the API

long_descr -- detail description about the event

short_descr -- short description about the event

event_note -- notes about the event

RETURN VALUES

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

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ENOTPRESET

The hardware event specified is not a valid PAPI preset.

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLE

```
int EventCode;
PAPI_event_info_t info;

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

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

if (PAPI_get_event_info(EventCode, &info) == PAPI_OK)
  handle_error(1);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_hardware_info, PAPI_get_opt

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_INT
```

DESCRIPTION

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

ARGUMENTS

The following arguments are implicit in the structure returned by the C function, or explicitly returned by Fortran.

fullname -- fully qualified path + filename of the executable

name -- filename of the executable with no path information

text_start, text_end -- Start and End addresses of program text segment

data_start, data_end -- Start and End addresses of program data segment

bss_start, bss_end -- Start and End addresses of program bss segment

RETURN VALUES

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

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

EXAMPLE

```
const PAPI_exe_info_t *prginfo = NULL;

if (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);
```

DATA STRUCTURES

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

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_library_init, PAPI_get_hardware_info, PAPI_get_opt

NAME

PAPI_get_hardware_info - get information about the system hardware

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

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

NOTE

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

ARGUMENTS

The following arguments are implicit in the structure returned by the C function, or explicitly returned by Fortran.

ncpu -- number of CPUs in an SMP Node

nnodes -- number of Nodes in the entire system

totalcpus -- total number of CPUs in the entire system

vendor -- vendor id number of CPU

vendor_string -- vendor id string of CPU

model -- model number of CPU

model_string -- model string of CPU

revision -- Revision number of CPU

mhz -- Cycle time of this CPU; *may* be an estimate generated at init time with a quick timing routine

RETURN VALUES

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

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

EXAMPLE

```
const PAPI_hw_info_t *hwinfo = NULL;

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

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

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

DATA STRUCTURE

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

```
   typedef struct _papi_hw_info {
      int ncpu;                 /* Number of CPU’s in an SMP Node */
      int nnodes;               /* Number of Nodes in the entire system
*/
      int totalcpus;            /* Total number of CPU’s in the entire
system */
      int vendor;               /* Vendor number of CPU */
      char vendor_string[PAPI_MAX_STR_LEN];     /* Vendor string of CPU
*/
      int model;                /* Model number of CPU */
      char model_string[PAPI_MAX_STR_LEN];      /* Model string of CPU
*/
      float revision;           /* Revision of CPU */
      float mhz;                  /* Cycle time of this CPU, *may* be
estimated at
                                     init time with a quick timing
routine */

      /* Memory Information */
      int L1_tlb_size;            /*Data + Instruction Size */
      int L1_itlb_size;           /*Instruction TLB size in KB */
      short int L1_itlb_assoc;    /*Instruction TLB associtivity */
      int L1_dtlb_size;           /*Data TLB size in KB */
      short L1_dtlb_assoc;        /*Data TLB associtivity */

      int L2_tlb_size;            /*Data + Instruction Size */
      int L2_itlb_size;           /*Instruction TLB size in KB */
      short int L2_itlb_assoc;    /*Instruction TLB associtivity */
      int L2_dtlb_size;           /*Data TLB size in KB */
      short L2_dtlb_assoc;        /*Data TLB associtivity */

      int L1_size;                /* I+D */
      int L1_icache_size;         /*Level 1 instruction cache size in KB
*/
      short int L1_icache_assoc; /*Level 1 instruction cache
associtivity */
      int L1_icache_lines;        /*Number of lines in Level 1
instruction cache */
      int L1_icache_linesize;     /*Line size in KB of Level 1
instruction cache */

      int L1_dcache_size;         /*Level 1 data cache size in KB */
      short int L1_dcache_assoc; /*Level 1 data cache associtivity */
      int L1_dcache_lines;        /*Number of lines in Level 1 data cache
*/
      int L1_dcache_linesize;     /*Line size in KB of Level 1 data cache
*/

      int L2_cache_size;          /*Level 2 cache size in KB */
      short int L2_cache_assoc;   /*Level 2 cache associtivity */
      int L2_cache_lines;         /*Number of lines in Level 2 cache */
      int L2_cache_linesize;      /*Line size in KB of Level 2 cache */

      int L3_cache_size;          /*Level 3 cache size in KB */
      short int L3_cache_assoc;   /*Level 3 cache associtivity */
      int L3_cache_lines;         /*Number of lines in Level 3 cache */
      int L3_cache_linesize;      /*Line size of Level 3 cache */
   } PAPI_hw_info_t;
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_library_init, PAPI_get_opt, PAPI_get_dmem_info, PAPI_get_executable_info

NAME

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

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

PAPI_get_multiplex Returns *TRUE if a PAPI event set is multiplexed, or FALSE if not.*

PAPI_set_multiplex Converts a standard PAPI event set created by a call to **PAPI_create_eventset()** into an event set capable of handling multiplexed events. This must be done after calling **PAPI_multiplex_init()** , but prior to calling **PAPI_start().** Events can be added to an 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 -- an integer handle for a PAPI event set as created by PAPI_create_eventset

RETURN VALUES

PAPI_get_multiplex returns either *TRUE (non-zero) or FALSE (zero).*

On success, **PAPI_get_multiplex 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;
PAPI_event_info_t pset;


/* Initialize the library */

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

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

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

if (PAPI_get_multiplex(EventSet) != 0)
  handle_error(1);

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

if (PAPI_get_multiplex(EventSet) == 0)
  handle_error(1);

   for (i = 0; i < PAPI_MAX_PRESET_EVENTS; i++) {
    retval = PAPI_get_event_info(i | PRESET_MASK, &pset);
    if (retval != PAPI_OK)
       test_fail(__FILE__, __LINE__, "PAPI_get_event_info", retval);

    if ((pset.count) && (pset.event_code != PAPI_TOT_CYC)) {
       if (!TESTS_QUIET)
          printf("Adding %s0, pset.symbol);

       retval = PAPI_add_event(EventSet, pset.event_code);
       if ((retval != PAPI_OK) && (retval != PAPI_ECNFLCT))
          test_fail(__FILE__, __LINE__, "PAPI_add_event", retval);

       if (!TESTS_QUIET) {
          if (retval == PAPI_OK)
             printf("Added %s0, pset.symbol);
          else
             printf("Could not add %s0, pset.symbol);
       }

       if (retval == PAPI_OK) {
          if (++j >= max_to_add)
             break;
```

```
            }
         }
      }

     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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_multiplex_init, PAPI_set_opt

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_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.

The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described 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_MAXMEM	not implemented yet.
PAPI_CLOCKRATE	Get clockrate in MHz.
PAPI_MAX_CPUS	Get number of CPUs.
PAPI_MAX_HWCTRS	Get number of counters.
PAPI_EXEINFO	Get Addresses for text/data/bss.
PAPI_HWINFO	Get Info. about hardware.
PAPI_SHLIBINFO	Get shared library Info. used by the program.
PAPI_PRELOAD	Get ‘‘LD_PRELOAD’’ environment equivalent.
Defaults for the global library	
PAPI_DEFDOM	Get/Set default counting domain for newly created event sets.
PAPI_DEFGRN	Get/Set default counting granularity.
PAPI_DEBUG	Get/Set the PAPI debug state. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level.
Multiplexing control	
PAPI_MULTIPLEX	Get/Set options for multiplexing.
Manipulating individual event sets	
PAPI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_GRANUL	Get/Set granularity for a single event set. The event set is specified in ptr->granularity.eventset. Not implemented yet.

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

```
typedef union {
  PAPI_preload_option_t preload;
  PAPI_debug_option_t debug;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_shlib_info_t *shlib_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_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_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start

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

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_virt_usec, PAPI_get_virt_cyc

NAME

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

SYNOPSIS

C Interface

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

DESCRIPTION

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

NOTE

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

RETURN VALUES

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

DATA STRUCTURE

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

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_library_init, PAPI_get_opt, PAPI_get_dmem_info, PAPI_get_executable_info, PAPI_get_hardware_info

NAME

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

SYNOPSIS

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

DESCRIPTION

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

ARGUMENTS

tag -- An identifier, the value of which is between 0 and PAPI_MAX_THREAD_STORAGE (currently 4). This identifier indicates which of several data structures associated with this thread is to be accessed.

ptr -- A pointer to the memory containing the data structure.

RETURN VALUES

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

ERRORS

PAPI_EINVAL

The *tag* argument is out of range.

EXAMPLE

```
   unsigned long int tid;
   HighLevelInfo *state = NULL;

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

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

```
/*
 * Do we have the thread specific data setup yet?
 */
if ((retval = PAPI_get_thr_specific(1, (void *) &state))
     != PAPI_OK || state == NULL) {
   state = (HighLevelInfo *) malloc(sizeof(HighLevelInfo));
   if (state == NULL)
      return (PAPI_ESYS);

   memset(state, 0, sizeof(HighLevelInfo));
   state->EventSet = -1;

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

   if ((retval = PAPI_set_thr_specific(1, state)) != PAPI_OK)
      return (retval);
}
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

There are no known bugs in these functions.

SEE ALSO

PAPI_thread_init, PAPI_thread_id, PAPI_register_thread

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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_real_usec, PAPI_get_real_cyc

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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_library_init, PAPI_get_real_usec, PAPI_get_real_cyc

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_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.

The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described 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_MAXMEM	not implemented yet.
PAPI_CLOCKRATE	Get clockrate in MHz.
PAPI_MAX_CPUS	Get number of CPUs.
PAPI_MAX_HWCTRS	Get number of counters.
PAPI_EXEINFO	Get Addresses for text/data/bss.
PAPI_HWINFO	Get Info. about hardware.
PAPI_SHLIBINFO	Get shared library Info. used by the program.
PAPI_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
	Defaults for the global library
PAPI_DEFDOM	Get/Set default counting domain for newly created event sets.
PAPI_DEFGRN	Get/Set default counting granularity.
PAPI_DEBUG	Get/Set the PAPI debug state. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level.
	Multiplexing control
PAPI_MULTIPLEX	Get/Set options for multiplexing.
	Manipulating individual event sets
PAPI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_GRANUL	Get/Set granularity for a single event set. The event set is specified in ptr->granularity.eventset. Not implemented yet.

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

```
typedef union {
  PAPI_preload_option_t preload;
  PAPI_debug_option_t debug;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_shlib_info_t *shlib_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_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_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start

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_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.

The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described 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_MAXMEM	not implemented yet.
PAPI_CLOCKRATE	Get clockrate in MHz.
PAPI_MAX_CPUS	Get number of CPUs.
PAPI_MAX_HWCTRS	Get number of counters.
PAPI_EXEINFO	Get Addresses for text/data/bss.
PAPI_HWINFO	Get Info. about hardware.
PAPI_SHLIBINFO	Get shared library Info. used by the program.
PAPI_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
	Defaults for the global library
PAPI_DEFDOM	Get/Set default counting domain for newly created event sets.
PAPI_DEFGRN	Get/Set default counting granularity.
PAPI_DEBUG	Get/Set the PAPI debug state. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level.
	Multiplexing control
PAPI_MULTIPLEX	Get/Set options for multiplexing.
	Manipulating individual event sets
PAPI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_GRANUL	Get/Set granularity for a single event set. The event set is specified in ptr->granularity.eventset. Not implemented yet.

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

```
typedef union {
  PAPI_preload_option_t preload;
  PAPI_debug_option_t debug;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_shlib_info_t *shlib_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_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_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start

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_INT
```

DESCRIPTION

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

ARGUMENTS

The following arguments are implicit in the structure returned by the C function, or explicitly returned by Fortran.

fullname -- fully qualified path + filename of the executable

name -- filename of the executable with no path information

text_start, text_end -- Start and End addresses of program text segment

data_start, data_end -- Start and End addresses of program data segment

bss_start, bss_end -- Start and End addresses of program bss segment

RETURN VALUES

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

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

EXAMPLE

```
const PAPI_exe_info_t *prginfo = NULL;

if (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);
```

DATA STRUCTURES

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

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_library_init, PAPI_get_hardware_info, PAPI_get_opt

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_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.

The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described 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_MAXMEM	not implemented yet.
PAPI_CLOCKRATE	Get clockrate in MHz.
PAPI_MAX_CPUS	Get number of CPUs.
PAPI_MAX_HWCTRS	Get number of counters.
PAPI_EXEINFO	Get Addresses for text/data/bss.
PAPI_HWINFO	Get Info. about hardware.
PAPI_SHLIBINFO	Get shared library Info. used by the program.
PAPI_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
	Defaults for the global library
PAPI_DEFDOM	Get/Set default counting domain for newly created event sets.
PAPI_DEFGRN	Get/Set default counting granularity.
PAPI_DEBUG	Get/Set the PAPI debug state. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level.
	Multiplexing control
PAPI_MULTIPLEX	Get/Set options for multiplexing.
	Manipulating individual event sets
PAPI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_GRANUL	Get/Set granularity for a single event set. The event set is specified in ptr->granularity.eventset. Not implemented yet.

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

```
typedef union {
  PAPI_preload_option_t preload;
  PAPI_debug_option_t debug;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_shlib_info_t *shlib_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_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_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start

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_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.

The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described 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_MAXMEM	not implemented yet.
PAPI_CLOCKRATE	Get clockrate in MHz.
PAPI_MAX_CPUS	Get number of CPUs.
PAPI_MAX_HWCTRS	Get number of counters.
PAPI_EXEINFO	Get Addresses for text/data/bss.
PAPI_HWINFO	Get Info. about hardware.
PAPI_SHLIBINFO	Get shared library Info. used by the program.
PAPI_PRELOAD	Get ‘‘LD_PRELOAD’’ environment equivalent.
Defaults for the global library	
PAPI_DEFDOM	Get/Set default counting domain for newly created event sets.
PAPI_DEFGRN	Get/Set default counting granularity.
PAPI_DEBUG	Get/Set the PAPI debug state. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level.
Multiplexing control	
PAPI_MULTIPLEX	Get/Set options for multiplexing.
Manipulating individual event sets	
PAPI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_GRANUL	Get/Set granularity for a single event set. The event set is specified in ptr->granularity.eventset. Not implemented yet.

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

```
typedef union {
    PAPI_preload_option_t preload;
    PAPI_debug_option_t debug;
    PAPI_granularity_option_t granularity;
    PAPI_granularity_option_t defgranularity;
    PAPI_domain_option_t domain;
    PAPI_domain_option_t defdomain;
    PAPI_multiplex_option_t multiplex;
    PAPI_hw_info_t *hw_info;
    PAPI_shlib_info_t *shlib_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_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_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start

NAME

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

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

The first call to **PAPI_ipc()** will initialize the PAPI High Level interface, set up the counters to monitor PAPI_TOT_INS and PAPI_TOT_CYC events and start the counters. Subsequent calls will read the counters and return total real time, total process time, total instructions since the start of the measurement and the instructions per cycle rate since latest call to **PAPI_ipc().**

ARGUMENTS

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

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

**ins* -- total instructions since the first call

**ipc* -- instructions per cycle achieved since the previous call

RETURN VALUES

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

ERRORS

EXAMPLES

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_accum , PAPI_flips , PAPI_flops

NAME

PAPI_library_init, PAPI_is_initialized - initialize the PAPI library; check for initialization

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

PAPI_library_init() initializes the PAPI library. It must be called before any low level PAPI functions can be used. If your application is making use of threads PAPI_thread_init must also be called prior to making any calls to the library other than **PAPI_library_init()**.

PAPI_is_initialized() returns the status of the PAPI library. The PAPI library can be in one of three states, as described under RETURN VALUES.

ARGUMENTS

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

RETURN VALUES

PAPI_library_init : On success, this function returns **PAPI_VER_CURRENT .** A positive return code other than **PAPI_VER_CURRENT** indicates a library version mis-match. A negative error code indicates an initialization error.

PAPI_is_initialized :

PAPI_NOT_INITED
-- PAPI has not been initialized

PAPI_LOW_LEVEL_INITED
-- PAPI_library_init has been called

PAPI_HIGH_LEVEL_INITED
-- a high level PAPI function has been called

ERRORS

PAPI_is_initialized never returns an error.

PAPI_library_init can return the following:

PAPI_EINVAL

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

PAPI_ENOMEM

Insufficient memory to complete the operation.

PAPI_ESBSTR

This substrate does not support the underlying hardware.

PAPI_ESYS

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

EXAMPLES

```
int retval;

/* Initialize the library */

retval = PAPI_library_init(PAPI_VER_CURRENT);

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

if (retval < 0)
  handle_error(retval);

retval = PAPI_is_initialized();

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_thread_init, PAPI_preset, PAPI

NAME

PAPI_list_events - list the events 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 greater 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;
int Events[4], number = 4;

/* Create an EventSet */

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

/* Convert an event name to an event code */

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

/* Query if Total Instructions Executed exists */

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

/* Add Total Instructions Executed to our EventSet */

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

/* Convert a second event name to an event code */

if (PAPI_event_name_to_code("PAPI_L1_LDM",&EventCode) != PAPI_OK)
  handle_error(1);

/* Query if L1 Load Misses exists */

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

/* Add L1 Load Misses to our EventSet */

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

/* List the events in our EventSet */

number = 4;
if(PAPI_list_events(EventSet, Events, &number);
  handle_error(1);

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_event_name_to_code, PAPI_query_event, PAPI_add_event

NAME

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

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

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

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

ARGUMENT

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

RETURN VALUES

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

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 total physical counters by time sharing the existing counters at some loss in precision. Applications that make no use of multiplexing do not need to call this routine.

RETURN VALUES

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 thread support in PAPI */
```

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_set_multiplex

NAME

PAPI_num_events - return the number of events in an event set

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

PAPI_num_events() returns the number of preset events contained in an event set.

ARGUMENTS

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

**count* -- On output the variable contains the number of events in the event set.

RETURN VALUES

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

ERRORS

PAPI_EINVAL

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

PAPI_ENOEVST

The EventSet specified does not exist.

EXAMPLES

```
int EventCode, EventSet = PAPI_NULL;

/* Create an EventSet */
```

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

/* Convert an event name to an event code */

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

/* Query if Total Instructions Executed exists */

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

/* Add Total Instructions Executed to our EventSet */

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

/* Convert a second event name to an event code */

if (PAPI_event_name_to_code("PAPI_L1_LDM",&EventCode) != PAPI_OK)
  handle_error(1);

/* Query if L1 Load Misses exists */

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

/* Add L1 Load Misses to our EventSet */

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

/* Count the events in our EventSet */

if (PAPI_num_events(EventSet) != 2)
  handle_error(1);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_event_name_to_code, PAPI_query_event, PAPI_add_event

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 to **PAPI_HIGH_LEVEL_INITED** if necessary.

RETURN VALUES

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

ERRORS

PAPI_EINVAL

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

PAPI_ENOMEM

Insufficient memory to complete the operation.

PAPI_ESYS

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

EXAMPLES

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

/*  The installation does not support PAPI */
```

```
  if ((num_hwcntrs = PAPI_num_counters()) < 0 )
    handle_error(1);

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

  if (num_hwcntrs > 2)
    num_hwcntrs = 2;

  /* Start counting events */

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_start_counters, PAPI_read_counters, PAPI_stop_counters, PAPI_init_library, PAPI_num_hwctrs, 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

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

None.

SEE ALSO

PAPI_get_opt, PAPI_init_library, PAPI_num_counters, PAPI

NAME

PAPI_overflow - set up an event set to begin registering overflows;

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_overflow (int EventSet, int EventCode, int threshold, int
flags, PAPI_overflow_handler_t handler);
```

Fortran Interface

```
Not implemented
```

DESCRIPTION

PAPI_overflow() marks a specific *EventCode* in an *EventSet* to generate an overflow signal after every *threshold* events are counted. More than one event in an event set can be used as overflow triggers, in such case, the user must call this function several times. **To turn off overflow, set the threshold to 0. If the user get negative number** or small number for the result of the overflow events, don't be surprised. To reduce the overhead of PAPI_read, we don't adjust the number. If the platform use hardware overflow, then based on the threshold and how many times it overflowed, you can get an approximate number.

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

RETURN VALUES

On success, PAPI_overflow 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

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

  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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_set_opt, PAPI_start, PAPI_remove_event, PAPI_remove_events, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset

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);
  }


/* Start counting */

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_set_debug, PAPI_set_opt, PAPI_get_opt, PAPI_shutdown,

NAME

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

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_profil(void * 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. The user can use more than one event to generate more than one histogram in one eventset. For example, if you want to use two events to generate two histograms while running your program once, then you just call PAPI_profil twice.

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.
PAPI_PROFIL_BUCKET_16	use unsigned short(16bit) as bucket, This is the default bucket.
PAPI_PROFIL_BUCKET_32	use unsigned int(32bit) as bucket.
PAPI_PROFIL_BUCKET_64	use unsigned long(64bit) as bucket.

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)/sizeof(unsigned short) *sizeof(unsigned short);

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

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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_get_executable_info, PAPI_set_opt, PAPI_start, PAPI_remove_event, PAPI_remove_events, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset

NAME

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

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

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

ARGUMENTS

EventCode -- a defined event such as PAPI_TOT_INS.

RETURN VALUES

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

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ENOTPRESET

The hardware event specified is not a valid PAPI preset.

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLES

```
int 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);
  }
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_native, PAPI_set_opt, PAPI_start, PAPI_remove_event, PAPI_remove_events, PAPI_cleanup_eventset, PAPI_destroy_eventset,

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 continue counting after the read.

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

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]);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_add_event, PAPI_reset, PAPI_set_opt, PAPI_remove_event,
PAPI_cleanup_eventset, PAPI_destroy_eventset,

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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_num_counters, PAPI_start_counters, PAPI_stop_counters

NAME

PAPI_remove_event, PAPI_remove_events - remove PAPI preset or native hardware event from an EventSet

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

PAPI_remove_event() removes a hardware event to a PAPI event set.
PAPI_remove_events() does the same, but for an array of hardware event codes.

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 a list of native events available on current platform, run native_avail test case in the PAPI distribution. For the encoding of native events, see PAPI_event_name_to_code to learn how to generate native code for the supported native event on the underlying architecture.

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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_event, PAPI_cleanup_eventset, PAPI_destroy_eventset, PAPI_event_name_to_code

NAME

PAPI_remove_event, PAPI_remove_events - remove PAPI preset or native hardware event from an EventSet

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

PAPI_remove_event() removes a hardware event to a PAPI event set.
PAPI_remove_events() does the same, but for an array of hardware event codes.

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 a list of native events available on current platform, run native_avail test case in the PAPI distribution. For the encoding of native events, see PAPI_event_name_to_code to learn how to generate native code for the supported native event on the underlying architecture.

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

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_event, PAPI_cleanup_eventset, PAPI_destroy_eventset, PAPI_event_name_to_code

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) != PAPI_OK)
  handle_error(1);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_add_event, PAPI_reset, PAPI_read, PAPI_set_opt, PAPI_remove_event, PAPI_cleanup_eventset, PAPI_destroy_eventset

NAME

PAPI_set_debug - set the current debug level for PAPI

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

PAPI_set_debug sets the debug level for 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.

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_set_opt

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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_set_opt

NAME

PAPI_set_granularity - set the execution granularity for which events are counted

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

This function is currently unimplemented.

RETURN VALUES

ERRORS

EXAMPLES

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function is currently unimplemented.

SEE ALSO

PAPI_set_domain, PAPI_set_opt, PAPI_get_opt

NAME

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

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

PAPI_get_multiplex Returns *TRUE if a PAPI event set is multiplexed, or FALSE if not.*

PAPI_set_multiplex Converts a standard PAPI event set created by a call to **PAPI_create_eventset()** into an event set capable of handling multiplexed events. This must be done after calling **PAPI_multiplex_init()** , but prior to calling **PAPI_start().** Events can be added to an 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 -- an integer handle for a PAPI event set as created by PAPI_create_eventset

RETURN VALUES

PAPI_get_multiplex returns either *TRUE (non-zero) or FALSE (zero).*

On success, **PAPI_get_multiplex 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;
PAPI_event_info_t pset;


/* Initialize the library */

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

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

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

if (PAPI_get_multiplex(EventSet) != 0)
  handle_error(1);

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

if (PAPI_get_multiplex(EventSet) == 0)
  handle_error(1);

   for (i = 0; i < PAPI_MAX_PRESET_EVENTS; i++) {
    retval = PAPI_get_event_info(i | PRESET_MASK, &pset);
    if (retval != PAPI_OK)
       test_fail(__FILE__, __LINE__, "PAPI_get_event_info", retval);

    if ((pset.count) && (pset.event_code != PAPI_TOT_CYC)) {
       if (!TESTS_QUIET)
          printf("Adding %s0, pset.symbol);

       retval = PAPI_add_event(EventSet, pset.event_code);
       if ((retval != PAPI_OK) && (retval != PAPI_ECNFLCT))
          test_fail(__FILE__, __LINE__, "PAPI_add_event", retval);

       if (!TESTS_QUIET) {
          if (retval == PAPI_OK)
             printf("Added %s0, pset.symbol);
          else
             printf("Could not add %s0, pset.symbol);
       }

       if (retval == PAPI_OK) {
          if (++j >= max_to_add)
             break;
             }
          }
       }

      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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_multiplex_init, PAPI_set_opt

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_DOMAIN and PAPI_MULTIPLEX, are also available as separate entry points in both C and Fortran.

The reader is urged to see the example code in the PAPI distribution for usage of PAPI_get_opt. The file **papi.h** contains definitions for the structures unioned in the *PAPI_option_t* structure.

ARGUMENTS

option -- is an input parameter describing the course of action. Possible values are defined in **papi.h** and briefly described 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_MAXMEM	not implemented yet.
PAPI_CLOCKRATE	Get clockrate in MHz.
PAPI_MAX_CPUS	Get number of CPUs.
PAPI_MAX_HWCTRS	Get number of counters.
PAPI_EXEINFO	Get Addresses for text/data/bss.
PAPI_HWINFO	Get Info. about hardware.
PAPI_SHLIBINFO	Get shared library Info. used by the program.
PAPI_PRELOAD	Get ''LD_PRELOAD'' environment equivalent.
	Defaults for the global library
PAPI_DEFDOM	Get/Set default counting domain for newly created event sets.
PAPI_DEFGRN	Get/Set default counting granularity.
PAPI_DEBUG	Get/Set the PAPI debug state. The available debug states are defined in papi.h. The debug state is available in ptr->debug.level.
	Multiplexing control
PAPI_MULTIPLEX	Get/Set options for multiplexing.
	Manipulating individual event sets
PAPI_DOMAIN	Get/Set domain for a single event set. The event set is specified in ptr->domain.eventset
PAPI_GRANUL	Get/Set granularity for a single event set. The event set is specified in ptr->granularity.eventset. Not implemented yet.

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

```
typedef union {
  PAPI_preload_option_t preload;
  PAPI_debug_option_t debug;
  PAPI_granularity_option_t granularity;
  PAPI_granularity_option_t defgranularity;
  PAPI_domain_option_t domain;
  PAPI_domain_option_t defdomain;
  PAPI_multiplex_option_t multiplex;
  PAPI_hw_info_t *hw_info;
  PAPI_shlib_info_t *shlib_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_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_DOMAIN, &options) != PAPI_OK)
  handle_error();
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_create_eventset, PAPI_add_event, PAPI_start

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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_set_opt

NAME

PAPI_shutdown - finish using PAPI and free all related resources

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_cleanup_eventset, PAPI_destroy_eventset

NAME

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

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_profil(void * 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. The user can use more than one event to generate more than one histogram in one eventset. For example, if you want to use two events to generate two histograms while running your program once, then you just call PAPI_profil twice.

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.
PAPI_PROFIL_BUCKET_16	use unsigned short(16bit) as bucket, This is the default bucket.
PAPI_PROFIL_BUCKET_32	use unsigned int(32bit) as bucket.
PAPI_PROFIL_BUCKET_64	use unsigned long(64bit) as bucket.

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)/sizeof(unsigned short) *sizeof(unsigned short);

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

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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_preset, PAPI_get_executable_info, PAPI_set_opt, PAPI_start, PAPI_remove_event, PAPI_remove_events, PAPI_query, PAPI_cleanup_eventset, PAPI_destroy_eventset

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 counting all of the hardware events contained in the previously defined EventSet. All counters are implicitly set to zero before counting.

PAPI_stop halts the counting of a previously defined event set and the counter values contained in that EventSet are copied into the values array

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]);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI_destroy_eventset, PAPI_add_event, PAPI_remove_event, PAPI_reset, PAPI_set_opt,

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

ARGUMENTS

**events* -- an array of codes for events such as PAPI_INT_INS or a native event code

array_len -- the number of items in the *events array

RETURN VALUES

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

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_EISRUN

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

PAPI_ESYS

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

PAPI_ENOMEM

Insufficient memory to complete the operation.

PAPI_ECNFLCT

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

PAPI_ENOEVNT

The PAPI preset is not available on the underlying hardware.

EXAMPLES

```
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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_num_counters, PAPI_read_counters, PAPI_stop_counters

NAME

PAPI_state - return the counting state of an EventSet

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

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

ARGUEMENTS

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

status -- an integer containing a boolean combination of one or more of the following nonzero constants as defined in the PAPI header file papi.h:

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

RETURN VALUES

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

ERRORS

PAPI_EINVAL

One or more of the arguments is invalid.

PAPI_ENOEVST

The EventSet specified does not exist.

EXAMPLES

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

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

/* Add Total Instructions Executed to our EventSet */

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

/* Start counting */

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

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

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

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

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_start, PAPI_stop

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 counting all of the hardware events contained in the previously defined EventSet. All counters are implicitly set to zero before counting.

PAPI_stop halts the counting of a previously defined event set and the counter values contained in that EventSet are copied into the values array

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]);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_create_eventset, PAPI_destroy_eventset, PAPI_add_event, PAPI_remove_event, PAPI_reset, PAPI_set_opt,

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.

PAPI_ENOTRUN

The eventset is not started yet.

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);
```

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_num_counters, PAPI_start_counters, PAPI_read_counters

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);
  }


/* Start counting */

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

PAPI_set_debug, PAPI_set_opt, PAPI_get_opt, PAPI_shutdown,

NAME

PAPI_thread_id - get the thread identifier of the current thread

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

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 tid;

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

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

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

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_thread_init

NAME

PAPI_thread_init - initialize thread support in the PAPI library

SYNOPSIS

C Interface

```
#include <papi.h>
int PAPI_thread_init (unsigned long int (*handle)());
```

Fortran Interface

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

DESCRIPTION

PAPI_thread_init initializes thread support in the PAPI library. 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.

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) != PAPI_OK)
  handle_error(1);
```

For OpenMP:

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

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

SEE ALSO

PAPI_library_init (3), **PAPI_thread_id** (3), **PAPI_get_thr_specific** (3), **PAPI_set_thr_specific** (3)

NAME

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

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

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

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

ARGUMENT

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

RETURN VALUES

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

AUTHORS

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

These functions have no known bugs.

SEE ALSO

NAME

`PAPI_write - Write counter values into counters`

SYNOPSIS

C Interface

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

Fortran Interface

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

DESCRIPTION

PAPI_write() writes the counter values provided in the array *values* into the event set *EventSet.* The virtual counters managed by the PAPI library will be set to the values provided. If the event set is running, an attempt will be made to write the values to the running counters. This operation is not permitted by all substrates and may result in a run-time error.

ARGUMENTS

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

**values* -- an array to hold the counter values of the counting events

RETURN VALUES

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

ERRORS

PAPI_ENOEVST

The EventSet specified does not exist.

PAPI_ESBSTR

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

EXAMPLES

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

AUTHOR

The PAPI Team. See them at the PAPI Web Site: http://icl.cs.utk.edu/projects/papi

BUGS

This function has no known bugs.

SEE ALSO

PAPI_accum

☐