# Contents

1 APEX: Autonomic Performance Environment for eXascale ........................ 1
  1.1 Copyright .................................................................................. 1
  1.2 Overview .................................................................................... 2
  1.3 Introduction ................................................................................ 2
    1.3.1 Interfaces .............................................................................. 2
  1.4 User Manual ................................................................................ 3
  1.5 Acknowledgements ...................................................................... 3

2 Namespace Index ............................................................................. 5
  2.1 Namespace List ............................................................................ 5

3 File Index ....................................................................................... 7
  3.1 File List ..................................................................................... 7

4 Namespace Documentation ................................................................ 9
  4.1 apex Namespace Reference .......................................................... 9
    4.1.1 Detailed Description .................................................................. 11
    4.1.2 Function Documentation .......................................................... 11
      4.1.2.1 cleanup(void) ................................................................. 11
      4.1.2.2 custom_event(apex_event_type event_type, void *custom_data) .. 11
      4.1.2.3 deregister_policy(apex_policy_handle *handle) ......................... 12
      4.1.2.4 exit_thread(void) .................................................................. 12
      4.1.2.5 finalize(void) ...................................................................... 12
      4.1.2.6 get_profile(apex_function_address function_address) .............. 12
      4.1.2.7 get_profile(const std::string &timer_name) ............................. 13
      4.1.2.8 get_thread_cap(void) ......................................................... 13
      4.1.2.9 get_tunable_params(apex_tuning_session_handle h) ................... 13
      4.1.2.10 has_session_converged(apex_tuning_session_handle handle) ....... 13
      4.1.2.11 init(const char *thread_name) .............................................. 14
      4.1.2.12 init(int argc, char **argv, const char *thread_name) ............. 14
      4.1.2.13 new_task(const std::string &name, void *task_id) .................. 14
      4.1.2.14 new_task(apex_function_address function_address, void *task_id) . 15
4.1.2.15 register_custom_event(const std::string &name) ........................................ 15
4.1.2.16 register_periodic_policy(unsigned long period, std::function<int(apex_context const &)> f) ................................................................. 15
4.1.2.17 register_policy(const apex_event_type when, std::function<int(apex_context const &)> f) ................................................................. 16
4.1.2.18 register_policy(std::set<apex_event_type> when, std::function<int(apex_context const &)> f) ................................................................. 16
4.1.2.19 register_thread(const std::string &name) .................................................... 16
4.1.2.20 reset(const std::string &timer_name) ......................................................... 17
4.1.2.21 reset(apex_function_address function_address) ........................................ 17
4.1.2.22 resume(const std::string &timer_name) ..................................................... 17
4.1.2.23 resume(apex_function_address function_address) ...................................... 18
4.1.2.24 sample_value(const std::string &name, double value) ................................ 18
4.1.2.25 set_node_id(int id) .................................................................................... 18
4.1.2.26 set_state(apex_thread_state state) .............................................................. 18
4.1.2.27 set_thread_cap(int new_cap) ....................................................................... 19
4.1.2.28 setup_custom_tuning(std::function<double(void)>, apex_event_type event_type, int num_inputs, long **inputs, long **mins, long **maxs, long **steps) ........................................................... 19
4.1.2.29 setup_custom_tuning(apex_tuning_request &request) ........................................ 19
4.1.2.30 setup_power_cap_throttling(void) ................................................................. 20
4.1.2.31 setup_throughput_tuning(apex_function_address the_address, apex_optimization_criteria_t criteria, apex_event_type event_type, int num_inputs, long **inputs, long **mins, long **maxs, long **steps) ........................................................... 20
4.1.2.32 setup_timer_throttling(apex_function_address the_address, apex_optimization_criteria_t criteria, apex_optimization_method_t method, unsigned long update_interval) ........................................................... 21
4.1.2.33 setup_timer_throttling(const std::string &the_name, apex_optimization_criteria_t criteria, apex_optimization_method_t method, unsigned long update_interval) ........................................................... 22
4.1.2.34 shutdown_throttling(void) .......................................................................... 22
4.1.2.35 start(const std::string &timer_name) .......................................................... 22
4.1.2.36 start(apex_function_address function_address) ........................................... 23
4.1.2.37 stop(profiler *the_profiler) ...................................................................... 23
4.1.2.38 version(void) ............................................................................................. 23
4.1.2.39 yield(profiler *the_profiler) ...................................................................... 23

5 File Documentation ................................. 25

5.1 /Users/khuck/src/xpress-apex/doc/apex.doc File Reference ................................................. 25
5.2 /Users/khuck/src/xpress-apex/src/apex/apex.h File Reference ............................................. 25
5.2.1 Function Documentation ................................................................. 26
5.2.1.1 apex_cleanup() ....................................................................................... 26
5.2.1.2 apex_current_power_high(void) ............................................................... 27
5.2.1.3 apex_custom_event(apex_event_type event_type, void *custom_data) ........ 27
5.2.1.4 apex_deregister_policy(apex_policy_handle *handle) .............................. 27
Chapter 1

APEX: Autonomic Performance Environment for eXascale

1.1 Copyright

******************************************************************************
APEX - Autonomic Performance Environment for eXascale

sub-project of:

eXascale PResources for Ecient and Reliable Systemx (XPRESS)
http://xstack.sandia.gov/xpress/

******************************************************************************

Copyright 2013-2015
Department of Computer and Information Science, University of Oregon
and partner institutions:
Sandia National Laboratories
Indiana University
Lawrence Berkeley National Laboratory
Louisiana State University
Oak Ridge National Laboratory
University of Houston
University of North Carolina

All Rights Reserved.
******************************************************************************

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided that
the above copyright notice appear in all copies and that both that copyright
notice and this permission notice appear in supporting documentation, and that
the name of University of Oregon (UO) and all partner institutions not be used
in advertising or publicity pertaining to distribution of the software without
specific, written prior permission. The University of Oregon and partner
institutions make no representations about the suitability of this software for
any purpose. It is provided "as is" without express or implied warranty.

THE UNIVERSITY OF OREGON AND PARTNER INSTITUTIONS DISCLAIM ALL WARRANTIES WITH
REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL THE UNIVERSITY OF OREGON OR PARTNER INSTITUTIONS
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF
CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION
WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
******************************************************************************
1.2 Overview

One of the key components of the XPRESS project is a new approach to performance observation, measurement, analysis and runtime decision making in order to optimize performance. The particular challenges of accurately measuring the performance characteristics of ParalleX applications requires a new approach to parallel performance observation. The standard model of multiple operating system processes and threads observing themselves in a first-person manner while writing out performance profiles or traces for offline analysis will not adequately capture the full execution context, nor provide opportunities for runtime adaptation within OpenX. The approach taken in the XPRESS project is a new performance measurement system, called (Autonomic Performance Environment for eXascale). APEX will include methods for information sharing between the layers of the software stack, from the hardware through operating and runtime systems, all the way to domain specific or legacy applications. The performance measurement components will incorporate relevant information across stack layers, with merging of third-person performance observation of node-level and global resources, remote processes, and both operating and runtime system threads.

1.3 Introduction

1.3.1 Interfaces

Essentially, APEX is both a measurement system for introspection, as well as a Policy Engine for modifying runtime behavior based on the observations. While APEX has capabilities for generating profile data for post-mortem analysis, the key purpose of the measurement is to provide support for policy enforcement. To that end, APEX is designed to have very low overhead and minimize perturbation of runtime worker thread productivity. APEX supports both start/stop timers and either event-based or periodic counter samples. Measurements are taken synchronously, but profiling statistics and internal state management is performed by (preferably lower-priority) threads distinct from the running application. The heart of APEX is an event handler that dispatches events to registered listeners within APEX. Policy enforcement can trigger synchronously when events are triggered by the OS/RS or application, or can occur asynchronously on a periodic basis.

APEX is a library written in C++, and has both C and C++ external interfaces. While the C interface can be used for either language, some C++ applications prefer to work with namespaces (i.e. apex::*) rather than prefixes (i.e. apex_). All functionality is supported through both interfaces, and the C interface contains inlined implementations of the C++ code.

While the designed purpose for APEX is supporting the current and future needs of ParalleX runtimes within the XPRESS project (such as HPX3, HPX5), experimental support is also available for introspection of current runtimes such as OpenMP. APEX could potentially be integrated into other runtime systems, such as any of a number of lightweight task based systems. The introspection provided by APEX is intended to be in the third-person model, rather than traditional first-person, per-thread/per-process application profile or tracing measurement. APEX is designed to combine information from the OS, Runtime, hardware and application in order to guide policy decisions.

For distributed communication, APEX provides an API to be implemented for the required communication for a given application. An MPI implementation is provided as a reference, and both HPX3 and HPX5 implementations have been implemented. In this way, APEX is integrated into the observed runtime, and asynchronous communication is provided at a lower priority, in order to minimize perturbation of the application.

The direct links to each API are here:

- C API : apex.h
- C++ API : apex
1.4 User Manual

For a complete user manual, please see the APEX documentation.

1.5 Acknowledgements

Support for this work was provided through Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences/Biological and Environmental Research/High Energy Physics/Fusion Energy Sciences/Nuclear Physics) under award numbers DE-SC0008638, DE-SC0008704, DE-FG02-11ER26050 and DE-SC0006925.
Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

apex

The main APEX namespace .................................................. 9
Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

/Users/khuck/src/xpress-apex/src/apex/apex.h .................................................. 25
/Users/khuck/src/xpress-apex/src/apex/apex_api.hpp ............................................. 36
/Users/khuck/src/xpress-apex/src/apex/apex_types.h ........................................... 38
/Users/khuck/src/xpress-apex/src/comm/apex_global.h ......................................... 42
Chapter 4

Namespace Documentation

4.1 apex Namespace Reference

The main APEX namespace.

Functions

- void init (const char *thread_name)
  Initialize APEX.
- void init (int argc, char **argv, const char *thread_name)
  Initialize APEX.
- void finalize (void)
  Finalize APEX.
- void cleanup (void)
  Cleanup APEX.
- profiler * start (const std::string &timer_name)
  Start a timer.
- profiler * start (apex_function_address function_address)
  Start a timer.
- void stop (profiler *the_profiler)
  Stop a timer.
- void yield (profiler *the_profiler)
  Stop a timer, but don't increment the number of calls.
- profiler * resume (const std::string &timer_name)
  Resume a timer.
- profiler * resume (apex_function_address function_address)
  Resume a timer.
- void reset (const std::string &timer_name)
  Reset a timer or counter.
- void reset (apex_function_address function_address)
  Reset a timer.
- void set_state (apex_thread_state state)
  Set the thread state.
- void sample_value (const std::string &name, double value)
  Sample a state value.
- void new_task (const std::string &name, void *task_id)
  Create a new task (dependency).
• void new_task (apex_function_address function_address, void *task_id)
  Create a new task (dependency).
• apex_event_type register_custom_event (const std::string &name)
  Register an event type with APEX.
• void custom_event (apex_event_type event_type, void *custom_data)
  Trigger a custom event.
• std::string & version (void)
  Return the APEX version.
• void set_node_id (int id)
  Set this process' node ID.
• void register_thread (const std::string &name)
  Register a new thread.
• void exit_thread (void)
  Exit a thread.
• apex_policy_handle * register_policy (const apex_event_type when, std::function<int(apex_context const &)> f)
  Register a policy with APEX.
• std::set< apex_policy_handle *> register_policy (std::set< apex_event_type > when, std::function<int(apex_context const &)> f)
  Register a policy with APEX.
• apex_policy_handle * register_periodic_policy (unsigned long period, std::function<int(apex_context const &)> f)
  Register a policy with APEX.
• void deregister_policy (apex_policy_handle *handle)
  Deregister a policy with APEX.
• apex_profile * get_profile (apex_function_address function_address)
  Get the current profile for the specified function address.
• apex_profile * get_profile (const std::string &timer_name)
  Get the current profile for the specified function address.
• int setup_power_cap_throttling (void)
  Initialize the power cap throttling policy.
• int setup_timer_throttling (apex_function_address the_address, apex_optimization_criteria_t criteria, apex←_optimization_method_t method, unsigned long update_interval)
  Setup throttling to optimize for the specified function.
• int setup_throughput_tuning (apex_function_address the_address, apex_optimization_criteria_t criteria, apex_event_type event_type, int num_inputs, long **inputs, long *mins, long *maxs, long *steps)
  Setup throttling to optimize for the specified function, using multiple input criteria.
• apex_tuning_session_handle setup_custom_tuning (std::function<double(void)> metric, apex_event_type event_type, int num_inputs, long **inputs, long *mins, long *maxs, long *steps)
  Setup tuning of specified parameters to optimize for a custom metric, using multiple input criteria.
• apex_tuning_session_handle setup_custom_tuning (apex_tuning_request &request)
  Setup tuning of specified parameters to optimize for a custom metric, using multiple input criteria of potentially multiple types.
• int setup_timer_throttling (const std::string &the_name, apex_optimization_criteria_t criteria, apex←optimization_method_t method, unsigned long update_interval)
  Setup throttling to optimize for the specified function or counter.
• int shutdown_throttling (void)
  Terminate the throttling policy.
• int get_thread_cap (void)
  Get the current thread cap set by the throttling.
• void set_thread_cap (int new_cap)
  Set the current thread cap for throttling.
• std::vector<std::pair<std::string, long>> get_tunable_params (apex_tuning_session_handle h)
  Return a vector of the current tunable parameters.
• bool has_session_converged (apex_tuning_session_handle handle)
  Check whether a tuning session has converged.
• void print_options (void)
  Print out all configuration settings for APEX.

4.1.1 Detailed Description

The main APEX namespace.
The C++ interface for APEX uses the apex namespace. In comparison, The C interface has functions that start with "apex_".

4.1.2 Function Documentation

4.1.2.1 void apex::cleanup ( void )

Cleanup APEX.

Warning

For best results, this function should be explicitly called to free all memory allocated by APEX. If not explicitly called from the application or runtime, it will be automatically called when the APEX main singleton object is destructed. apex::finalize will be automatically called from apex::cleanup if it has not yet been called.

The cleanup method will free all allocated memory for APEX.

Returns

No return value.

See also

apex::init apex::finalize

4.1.2.2 void apex::custom_event ( apex_event_type event_type, void * custom_data )

Trigger a custom event.

This function will pass a custom event to the APEX event listeners. Each listeners’ custom event handler will handle the custom event. Policy functions will be passed the custom event name in the event context.

Parameters

<table>
<thead>
<tr>
<th>event_type</th>
<th>The type of the custom event</th>
</tr>
</thead>
<tbody>
<tr>
<td>custom_data</td>
<td>Data specific to the custom event</td>
</tr>
</tbody>
</table>

Returns

No return value.

See also

apex::register_custom_event
4.1.2.3  void apex::deregister_policy ( apex_policy_handle * handle )

Deregister a policy with APEX.
This function will deregister the specified policy. In order to enable the policy again, it should be registered using
apex::register_policy or apex::register_periodic_policy.

Parameters

| handle | The handle of the policy to be deregistered. |

See also

apex::register_policy, apex::register_periodic_policy

4.1.2.4  void apex::exit_thread ( void )

Exit a thread.
For multithreaded applications, exit this thread and clean up.

Warning

Failure to exit a thread with APEX may invalidate statistics.

Returns

No return value.

4.1.2.5  void apex::finalize ( void )

Finalize APEX.
The stop measurement method will terminate all measurement and optionally:

- print a report to the screen
- write a TAU profile to disk

Returns

No return value.

See also

apex::init

4.1.2.6  apex_profile* apex::get_profile ( apex_function_address function_address )

Get the current profile for the specified function address.
This function will return the current profile for the specified address. Because profiles are updated out-of-band, it is
possible that this profile value is out of date.
Parameters

| function_address | The address of the function. |

Returns

The current profile for that timed function.

4.1.2.7 \texttt{apex\_profile}\texttt{*} apex::get\_profile \ ( \ const \ std::string & \ \textit{timer\_name} )

Get the current profile for the specified function address.

This function will return the current profile for the specified address. Because profiles are updated out-of-band, it is possible that this profile value is out of date. This profile can be either a timer or a sampled value.

Parameters

\begin{tabular}{|c|c|}
\hline
\textit{timer\_name} & The name of the function \\
\hline
\end{tabular}

Returns

The current profile for that timed function or sampled value.

4.1.2.8 \texttt{int} apex::get\_thread\_cap \ ( \ void \ )

Get the current thread cap set by the throttling.

This function will return the current thread cap based on the throttling policy.

Returns

The current thread cap value.

4.1.2.9 \texttt{std::vector<\texttt{std::pair<\texttt{std::string,long>}>}} & apex::get\_tunable\_params \ ( \ \texttt{apex\_tuning\_session\_handle} \ \textit{h} )

Return a vector of the current tunable parameters.

Returns

A vector of pairs; the first element is the name of the tunable parameter, while the second is a pointer to its value.

4.1.2.10 \texttt{bool} apex::has\_session\_converged \ ( \ \texttt{apex\_tuning\_session\_handle} \ \textit{handle} )

Check whether a tuning session has converged.

Parameters

\begin{tabular}{|c|c|}
\hline
\textit{handle} & The handle for the tuning session of interest. \\
\hline
\end{tabular}

Returns

true if the tuning session has converged, otherwise false
4.1.2.11 void apex::init ( const char * thread_name )

Initialize APEX.

Warning
For best results, this function should be called before any other APEX functions.
Use this version of apex::init when you do not have access to the input arguments.

Parameters

| thread_name | The name of the thread, or NULL. The lifetime of the thread will be timed with a timer using this same name. |

Returns
No return value.

See also
apex::init apex::finalize

4.1.2.12 void apex::init ( int argc, char ** argv, const char * thread_name )

Initialize APEX.

Warning
For best results, this function should be called before any other APEX functions.
Use this version of apex::init when you have access to the input arguments.

Parameters

| argc | The number of arguments passed in to the program. |
| argv | An array of arguments passed in to the program. |
| thread_name | The name of the thread, or NULL. The lifetime of the thread will be timed with a timer using this same name. |

Returns
No return value.

See also
apex::init apex::finalize

4.1.2.13 void apex::new_task ( const std::string & name, void * task_id )

Create a new task (dependency).
This function will note a task dependency between the current timer (task) and the new task.
Parameters

<table>
<thead>
<tr>
<th>name</th>
<th>The name of the timer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_id</td>
<td>The ID of the task</td>
</tr>
</tbody>
</table>

Returns

No return value.

4.1.2.14 `void apex::new_task ( apex_function_address function_address, void * task_id )`

Create a new task (dependency).

This function will note a task dependency between the current timer (task) and the new task.

Parameters

<table>
<thead>
<tr>
<th>function_address</th>
<th>The function address of the timer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_id</td>
<td>The ID of the task</td>
</tr>
</tbody>
</table>

Returns

No return value.

4.1.2.15 `apex_event_type apex::register_custom_event ( const std::string & name )`

Register an event type with APEX.

Create a user-defined event type for APEX.

Parameters

| name     | The name of the custom event |

Returns

The index of the custom event.

See also

`apex::custom_event`

4.1.2.16 `apex_policy_handle* apex::register_periodic_policy ( unsigned long period, std::function< int(apex_context const &)> f )`

Register a policy with APEX.

Apex provides the ability to call an application-specified function periodically. This assigns the passed in function to be called on a periodic basis. The context for the event will be passed to the registered function.

Parameters

| period   | How frequently the function should be called |
4.1.2.17  

**apex_policy_handle** = apex::register_policy ( const apex_event_type when, std::function<int(apex_context const &)> f )

Register a policy with APEX.

Apex provides the ability to call an application-specified function when certain events occur in the APEX library, or periodically. This assigns the passed in function to the event, so that when that event occurs in APEX, the function is called. The context for the event will be passed to the registered function.

**Parameters**

<table>
<thead>
<tr>
<th>when</th>
<th>The APEX event when this function should be called</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>The function to be called when that event is handled by APEX.</td>
</tr>
</tbody>
</table>

**Returns**

A handle to the policy, to be stored if the policy is to be un-registered later.

See also

apex::deregister_policy, apex::register_periodic_policy

4.1.2.18  

**std::set<apex_policy_handle>** = apex::register_policy ( std::set<apex_event_type> when, std::function<int(apex_context const &)> f )

Register a policy with APEX.

Apex provides the ability to call an application-specified function when certain events occur in the APEX library, or periodically. This assigns the passed in function to the event, so that when that event occurs in APEX, the function is called. The context for the event will be passed to the registered function.

**Parameters**

<table>
<thead>
<tr>
<th>when</th>
<th>The set of APEX events when this function should be called</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>The function to be called when that event is handled by APEX.</td>
</tr>
</tbody>
</table>

**Returns**

A handle to the policy, to be stored if the policy is to be un-registered later.

See also

apex::deregister_policy, apex::register_periodic_policy

4.1.2.19  

**void** apex::register_thread ( const std::string & name )

Register a new thread.

For multithreaded applications, register a new thread with APEX.

**Warning**

Failure to register a thread with APEX may invalidate statistics, and may prevent the ability to use timers or sampled values for this thread.
Parameters

| name         | The name that will be assigned to the new thread. |

Returns

No return value.

4.1.2.20 void apex::reset ( const std::string & timer_name )

Reset a timer or counter.
This function will reset the profile associated with the specified timer or counter name to zero.

Parameters

| timer_name | The name of the timer. |

Returns

No return value.

See also

apex::get_profile

4.1.2.21 void apex::reset ( apex_function_address function_address )

Reset a timer.
This function will reset the profile associated with the specified timer to zero.

Parameters

| function_address | The function address of the timer. |

Returns

No return value.

4.1.2.22 profiler= apex::resume ( const std::string & timer_name )

Resume a timer.
This function will create a profiler object in APEX, and return a handle to the object. The object will be associated with the name passed in to this function. The difference between this function and the apex::start function is that the number of calls to that timer will not be incremented.

Parameters

| timer_name | The name of the timer. |

Returns

The handle for the timer object in APEX. Not intended to be queried by the application. Should be retained locally, if possible, and passed in to the matching apex::stop() call when the timer should be stopped.

See also

apex::stop, apex::yield, apex::start
### 4.1.2.23 profiler

```cpp
apex::resume ( apex_function_address function_address )
```

Resume a timer.

This function will create a profiler object in APEX, and return a handle to the object. The object will be associated with the address passed in to this function. The difference between this function and the `apex::start` function is that the number of calls to that timer will not be incremented.

**Parameters**

| `function_address` | The address of the function to be timed |

**Returns**

The handle for the timer object in APEX. Not intended to be queried by the application. Should be retained locally, if possible, and passed in to the matching `apex::stop` call when the timer should be stopped.

See also

`apex::stop, apex::yield, apex::start`

### 4.1.2.24 void apex::sample_value ( const std::string & name, double value )

Sample a state value.

This function will retain a sample of some value. The profile for this sampled value will store the min, mean, max, total and standard deviation for this value for all times it is sampled.

**Parameters**

| `name` | The name of the sampled value |
| `value` | The sampled value |

**Returns**

No return value.

### 4.1.2.25 void apex::set_node_id ( int id )

Set this process' node ID.

For distributed applications, this function will store the node ID. Common values are the MPI rank, the HPX locality, etc. This ID will be used to identify the process in the global performance space.

**Parameters**

| `id` | The node ID for this process. |

**Returns**

No return value.

### 4.1.2.26 void apex::set_state ( apex_thread_state state )

Set the thread state.

This function will set the thread state in APEX for 3rd party observation.
Parameters

| state | The state of the thread. |

Returns

No return value.

4.1.2.27 void apex::set_thread_cap ( int new_cap )

Set the current thread cap for throttling.
This function will set the current thread cap based on an external throttling policy.
Parameters

| new_cap | The current thread cap value. |

4.1.2.28 apex_tuning_session_handle apex::setup_custom_tuning ( std::function< double(void) > metric,
                        apex_event_type event_type, int num_inputs, long * inputs, long * mins, long * maxs, long * steps )

Setup tuning of specified parameters to optimize for a custom metric, using multiple input criteria.
This function will initialize a policy to optimize a custom metric, using the list of tunable parameters. The system tries to minimize the custom metric. After evaluating the state of the system, the policy will assign new values to the inputs.
Parameters

| metric | A function returning the value to be minimized. |
| event_type | The apex_event_type that should trigger this policy |
| num_inputs | The number of tunable inputs for optimization |
| inputs | An array of addresses to inputs for optimization |
| mins | An array of minimum values for each input |
| maxs | An array of maximum values for each input |
| steps | An array of step values for each input |

Returns

A handle to the tuning session

4.1.2.29 apex_tuning_session_handle apex::setup_custom_tuning ( apex_tuning_request & request )

Setup tuning of specified parameters to optimize for a custom metric, using multiple input criteria of potentially multiple types.
This function will initialize a policy to optimize a custom metric, using metric and parameters specified in the tuning request. The system tries to minimize the custom metric.
Parameters

| request | An apex_tuning_request object that specifies the tuning parameters. |

Returns

A handle to the tuning session.
Initialize the power cap throttling policy.

This function will initialize APEX for power cap throttling. There are several environment variables that control power cap throttling:

**HPX_THROTTLING** If set, throttling will be enabled and initialized at startup.

**APEX_THROTTLING_MAX_THREADS** The maximum number of threads the throttling system will allow. The default value is 48.

**APEX_THROTTLING_MIN_THREADS** The minimum number of threads the throttling system will allow. The default value is 12.

**APEX_THROTTLING_MAX_WATTS** The maximum number of Watts the system can consume as an average rate. The default value is 220.

**APEX_THROTTLING_MIN_WATTS** The minimum number of Watts the system can consume as an average rate. The default value is 180.

**HPX_ENERGY_THROTTLING** If set, power/energy throttling will be performed.

**HPX_ENERGY** TBD

After evaluating the state of the system, the policy will set the thread cap, which can be queried using `apex::get_thread_cap()`.

Returns

APEX_NOERROR on success, otherwise an error code.

Setup throttling to optimize for the specified function, using multiple input criteria.

This function will initialize a policy to optimize the specified function, using the list of tunable inputs for the specified function. The optimization criteria include maximizing throughput, minimizing or maximizing time spent in the specified function. After evaluating the state of the system, the policy will assign new values to the inputs.

**Parameters**

<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>the_address</code></td>
<td>The address of the function to be optimized.</td>
</tr>
<tr>
<td><code>criteria</code></td>
<td>The optimization criteria.</td>
</tr>
<tr>
<td><code>event_type</code></td>
<td>The <code>apex_event_type</code> that should trigger this policy</td>
</tr>
<tr>
<td><code>num_inputs</code></td>
<td>The number of tunable inputs for optimization</td>
</tr>
<tr>
<td><code>inputs</code></td>
<td>An array of addresses to inputs for optimization</td>
</tr>
<tr>
<td><code>mins</code></td>
<td>An array of minimum values for each input</td>
</tr>
<tr>
<td><code>maxs</code></td>
<td>An array of maximum values for each input</td>
</tr>
<tr>
<td><code>steps</code></td>
<td>An array of step values for each input</td>
</tr>
</tbody>
</table>

Returns

APEX_NOERROR on success, otherwise an error code.
Setup throttling to optimize for the specified function.

This function will initialize the throttling policy to optimize for the specified function. The optimization criteria include maximizing throughput, minimizing or maximizing time spent in the specified function. After evaluating the state of the system, the policy will set the thread cap, which can be queried using `apex::get_thread_cap()`.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>the_address</td>
<td>The address of the function to be optimized.</td>
</tr>
<tr>
<td>criteria</td>
<td>The optimization criteria.</td>
</tr>
<tr>
<td>method</td>
<td>The optimization method.</td>
</tr>
<tr>
<td>update_interval</td>
<td>The time between observations, in microseconds.</td>
</tr>
</tbody>
</table>

Returns
APEX_NOERROR on success, otherwise an error code.

4.1.2.33 int apex::setup_timer_throttling ( const std::string & the_name, apex_optimization_criteria_t criteria, apex_optimization_method_t method, unsigned long update_interval )

Setup throttling to optimize for the specified function or counter.
This function will initialize the throttling policy to optimize for the specified function or counter. The optimization criteria include maximizing throughput, minimizing or maximizing time spent in the specified function or value sampled in the counter. After evaluating the state of the system, the policy will set the thread cap, which can be queried using apex::get_thread_cap().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>the_name</td>
<td>The name of the function or counter to be optimized.</td>
</tr>
<tr>
<td>criteria</td>
<td>The optimization criteria.</td>
</tr>
<tr>
<td>method</td>
<td>The optimization method.</td>
</tr>
<tr>
<td>update_interval</td>
<td>The time between observations, in microseconds.</td>
</tr>
</tbody>
</table>

Returns
APEX_NOERROR on success, otherwise an error code.

4.1.2.34 int apex::shutdown_throttling ( void )

Terminate the throttling policy.
This function will terminate the throttling policy.

Returns
APEX_NOERROR on success, otherwise an error code.

4.1.2.35 profiler* apex::start ( const std::string & timer_name )

Start a timer.
This function will create a profiler object in APEX, and return a handle to the object. The object will be associated with the name passed in to this function.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer_name</td>
<td>The name of the timer.</td>
</tr>
</tbody>
</table>

Returns
The handle for the timer object in APEX. Not intended to be queried by the application. Should be retained locally, if possible, and passed in to the matching apex::stop() call when the timer should be stopped.
4.1.2.36 **profiler**\* apex::start ( apex_function_address function_address )

Start a timer.

This function will create a profiler object in APEX, and return a handle to the object. The object will be associated with the address passed in to this function.

**Parameters**

| function_address | The address of the function to be timed |

**Returns**

The handle for the timer object in APEX. Not intended to be queried by the application. Should be retained locally, if possible, and passed in to the matching `apex::stop` call when the timer should be stopped.

See also

`apex::stop, apex::yield, apex::resume`

4.1.2.37 void apex::stop ( profiler \* the_profiler )

Stop a timer.

This function will stop the specified profiler object, and queue the profiler to be processed out-of-band. The timer value will eventually added to the profile for the process.

**Parameters**

| the_profiler | The handle of the profiler object. |

**Returns**

No return value.

See also

`apex::start, apex::yield, apex::resume`

4.1.2.38 std::string& apex::version ( void )

Return the APEX version.

**Returns**

A string with the APEX version.

4.1.2.39 void apex::yield ( profiler \* the_profiler )

Stop a timer, but don't increment the number of calls.

This function will stop the specified profiler object, and queue the profiler to be processed out-of-band. The timer value will eventually added to the profile for the process. The number of calls will NOT be incremented - this “task” was yielded, not completed. It will be resumed by another thread at a later time.
Parameters

| the_profiler | The handle of the profiler object. |

Returns
No return value.

See also
apex::start, apex::stop, apex::resume
Chapter 5

File Documentation

5.1 /Users/khuck/src/xpress-apex/doc/apex.dox File Reference

5.2 /Users/khuck/src/xpress-apex/src/apex/apex.h File Reference

#include "apex_types.h"
#include "apex_export.h"

Functions

- void apex_init (const char *thread_name)
  Initialize APEX.
- void apex_init_args (int argc, char **argv, const char *thread_name)
  Initialize APEX.
- void apex_finalize ()
  Finalize APEX.
- void apex_cleanup ()
  Cleanup APEX.
- apex_profiler_handle apex_start (apex_profiler_type type, void *identifier)
  Start a timer.
- void apex_stop (apex_profiler_handle profiler)
  Stop a timer.
- void apex_yield (apex_profiler_handle profiler)
  Stop a timer, but don't increment the number of calls.
- apex_profiler_handle apex_resume (apex_profiler_type type, void *identifier)
  Resume a timer.
- void apex_reset (apex_profiler_type type, void *identifier)
  Reset a timer or counter.
- void apex_set_state (apex_thread_state state)
  Set the thread state.
- void apex_sample_value (const char *name, double value)
  Sample a state value.
- void apex_new_task (apex_profiler_type type, void *identifier, void *task_id)
  Create a new task (dependency).
- apex_event_type apex_register_custom_event (const char *name)
  Register an event type with APEX.
• void `apex_custom_event` (apex_event_type event_type, void *custom_data)
  
  Trigger a custom event.

• const char * `apex_version` (void)

  Return the APEX version.

• void `apex_set_node_id` (int id)

  Set this process’ node ID.

• void `apex_register_thread` (const char *name)

  Register a new thread.

• void `apex_exit_thread` (void)

  Exit a thread.

• `apex_policy_handle` * `apex_register_policy` (const apex_event_type when, apex_policy_function f)

  Register a policy with APEX.

• `apex_policy_handle` * `apex_register_periodic_policy` (unsigned long period, apex_policy_function f)

  Register a policy with APEX.

• void `apex_deregister_policy` (apex_policy_handle *handle)

  Deregister a policy with APEX.

• `apex_profile` * `apex_get_profile` (apex_profiler_type type, void *identifier)

  Get the current profile for the specified id.

• double `apex_current_power_high` (void)

  Get the current power reading.

• int `apex_setup_power_cap_throttling` (void)

  Initialize the power cap throttling policy.

• int `apex_setup_timer_throttling` (apex_profiler_type type, void *identifier, apex_optimization_criteria_t criteria, apex_optimization_method_t method, unsigned long update_interval)

  Setup throttling to optimize for the specified function.

• int `apex_setup_throughput_tuning` (apex_profiler_type type, void *identifier, apex_optimization_criteria_t criteria, apex_event_type event_type, int num_inputs, long **inputs, long *mins, long *maxs, long *steps)

  Setup throttling to optimize for the specified function, using multiple input criteria.

• int `apex_shutdown_throttling` (void)

  Terminate the throttling policy.

• int `apex_get_thread_cap` (void)

  Get the current thread cap set by the throttling.

• void `apex_set_thread_cap` (int new_cap)

  Set the current thread cap for throttling.

• void `apex_print_options` (void)

  Print the current APEX settings.

### 5.2.1 Function Documentation

#### 5.2.1.1 `void apex_cleanup` ( )

Cleanup APEX.

**Warning**

For best results, this function should be explicitly called to free all memory allocated by APEX. If not explicitly called from the application or runtime, it will be automatically called when the APEX main singleton object is destructed. `apex_finalize` will be automatically called from `apex_cleanup` if it has not yet been called.

The cleanup method will free all allocated memory for APEX.
Returns

No return value.

See also

apex_finalize

5.2.1.2 double apex_current_power_high ( void )

Get the current power reading.
This function will return the current power level for the node, measured in Watts.

Returns

The current power level in Watts.

5.2.1.3 void apex_custom_event ( apex_event_type event_type, void * custom_data )

Trigger a custom event.
This function will pass a custom event to the APEX event listeners. Each listeners' custom event handler will handle
the custom event. Policy functions will be passed the custom event name in the event context.

Parameters

| event_type | The type of the custom event |
| custom_data | Data specific to the custom event |

Returns

No return value.

See also

apex_register_custom_event

5.2.1.4 void apex_deregister_policy ( apex_policy_handle * handle )

Deregister a policy with APEX.
This function will deregister the specified policy. In order to enable the policy again, it should be registered using
apex_register_policy or apex_register_periodic_policy.

Parameters

| handle | The handle of the policy to be deregistered. |

See also

apex_register_policy, apex_register_periodic_policy
5.2.1.5  void apex_exit_thread ( void )

Exit a thread.
For multithreaded applications, exit this thread and clean up.

Warning

Failure to exit a thread with APEX may invalidate statistics.

Returns

No return value.

5.2.1.6  void apex_finalize ( )

Finalize APEX.
The stop measurement method will terminate all measurement and optionally:

• print a report to the screen
• write a TAU profile to disk

Returns

No return value.

See also

apex_init, apex_init_args

5.2.1.7  apex_profile apex_get_profile ( apex_profiler_type type, void * identifier )

Get the current profile for the specified id.
This function will return the current profile for the specified profiler id. Because profiles are updated out-of-band, it
is possible that this profile value is out of date. This profile can be either a timer or a sampled value.

Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>The type of the address to be returned. This can be one of the apex_profiler_type values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>The function address of the function to be returned, or a &quot;const char *&quot; pointer to the name</td>
</tr>
<tr>
<td></td>
<td>of the timer / counter.</td>
</tr>
</tbody>
</table>

Returns

The current profile for that timed function or sampled value.

5.2.1.8  int apex_get_thread_cap ( void )

Get the current thread cap set by the throttling.
This function will return the current thread cap based on the throttling policy.

Returns

The current thread cap value.
5.2.1.9  void apex_init ( const char * thread_name )

Initialize APEX.

Warning
For best results, this function should be called before any other APEX functions.
Use this version of apex_init when you do not have access to the input arguments.

Parameters

| thread_name | The name of the thread, or NULL. The lifetime of the thread will be timed with a timer using this same name. |

Returns
No return value.

See also
apex_init_args, apex_finalize

5.2.1.10 void apex_init_args ( int argc, char ** argv, const char * thread_name )

Initialize APEX.

Warning
For best results, this function should be called before any other APEX functions.
Use this version of apex_init when you have access to the input arguments.

Parameters

| argc | The number of arguments passed in to the program. |
| argv | An array of arguments passed in to the program. |
| thread_name | The name of the thread, or NULL. The lifetime of the thread will be timed with a timer using this same name. |

Returns
No return value.

See also
apex_init, apex_finalize

5.2.1.11 void apex_new_task ( apex_profiler_type type, void * identifier, void * task_id )

Create a new task (dependency).
This function will note a task dependency between the current timer (task) and the new task.
Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>The type of the address to be reset. This can be one of the apex_profiler_type values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>The function address of the function of the task, or a &quot;const char ∗&quot; pointer to the name of the task.</td>
</tr>
<tr>
<td>task_id</td>
<td>The ID of the task</td>
</tr>
</tbody>
</table>

Returns

No return value.

5.2.1.12 void apex_print_options ( void )

Print the current APEX settings.
This function will print all the current APEX settings.

5.2.1.13 apex_event_type apex_register_custom_event ( const char ∗ name )

Register an event type with APEX.
Create a user-defined event type for APEX.
Parameters

| name | The name of the custom event |

Returns

The index of the custom event.

See also

apex_custom_event

5.2.1.14 apex_policy_handle ∗ apex_register_periodic_policy ( unsigned long period, apex_policy_function f )

Register a policy with APEX.
Apex provides the ability to call an application-specified function periodically. This assigns the passed in function to be called on a periodic basis. The context for the event will be passed to the registered function.
Parameters

<table>
<thead>
<tr>
<th>period</th>
<th>How frequently the function should be called</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>The function to be called when that event is handled by APEX.</td>
</tr>
</tbody>
</table>

Returns

A handle to the policy, to be stored if the policy is to be un-registered later.

See also

apex_deregister_policy, apex_register_policy
5.2.1.15  apex_policy_handle* apex_register_policy ( const apex_event_type when, apex_policy_function f )

Register a policy with APEX.

Apex provides the ability to call an application-specified function when certain events occur in the APEX library, or periodically. This assigns the passed in function to the event, so that when that event occurs in APEX, the function is called. The context for the event will be passed to the registered function.

Parameters

<table>
<thead>
<tr>
<th>when</th>
<th>The APEX event when this function should be called</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>The function to be called when that event is handled by APEX.</td>
</tr>
</tbody>
</table>

Returns

A handle to the policy, to be stored if the policy is to be un-registered later.

See also

apex_deregister_policy, apex_register_periodic_policy

5.2.1.16  void apex_register_thread ( const char * name )

Register a new thread.

For multithreaded applications, register a new thread with APEX.

Warning

Failure to register a thread with APEX may invalidate statistics, and may prevent the ability to use timers or sampled values for this thread.

Parameters

| name | The name that will be assigned to the new thread. |

Returns

No return value.

5.2.1.17  void apex_reset ( apex_profiler_type type, void * identifier )

Reset a timer or counter.

This function will reset the profile associated with the specified timer or counter id to zero.

Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>The type of the address to be reset. This can be one of the apex_profiler_type values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>The function address of the function to be reset, or a &quot;const char *&quot; pointer to the name of the timer / counter.</td>
</tr>
</tbody>
</table>

Returns

No return value.

See also

apex_get_profile
5.2.1.18  `apex_profiler_handle apex_resume ( apex_profiler_type type, void * identifier )`
Resume a timer.
This function will create a profiler object in APEX, and return a handle to the object. The object will be associated
with the name and/or function address passed in to this function. The difference between this function and the
`apex_start` function is that the number of calls to that timer will not be incremented.

**Parameters**

<table>
<thead>
<tr>
<th>type</th>
<th>The type of the address to be stored. This can be one of the <code>apex_profiler_type</code> values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>The function address of the function to be timed, or a &quot;const char *&quot; pointer to the name of the timer.</td>
</tr>
</tbody>
</table>

**Returns**
The handle for the timer object in APEX. Not intended to be queried by the application. Should be retained
locally, if possible, and passed in to the matching `apex_stop` call when the timer should be stopped.

**See also**
`apex_start`, `apex_stop`, `apex_yield`

5.2.1.19  `void apex_sample_value ( const char * name, double value )`
Sample a state value.
This function will retain a sample of some value. The profile for this sampled value will store the min, mean, max,
total and standard deviation for this value for all times it is sampled.

**Parameters**

<table>
<thead>
<tr>
<th>name</th>
<th>The name of the sampled value</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The sampled value</td>
</tr>
</tbody>
</table>

**Returns**
No return value.

5.2.1.20  `void apex_set_node_id ( int id )`
Set this process' node ID.
For distributed applications, this function will store the node ID. Common values are the MPI rank, the HPX locality,
etc. This ID will be used to identify the process in the global performance space.

**Parameters**

<table>
<thead>
<tr>
<th>id</th>
<th>The node ID for this process.</th>
</tr>
</thead>
</table>

**Returns**
No return value.

5.2.1.21  `void apex_set_state ( apex_thread_state state )`
Set the thread state.
This function will set the thread state in APEX for 3rd party observation
Parameters

| state | The state of the thread |

Returns

No return value.

5.2.1.22 void apex_set_thread_cap ( int new_cap )

Set the current thread cap for throttling.
This function will set the current thread cap based on an external throttling policy.

Parameters

| new_cap | The current thread cap value |

5.2.1.23 int apex_setup_power_cap_throttling ( void )

Initialize the power cap throttling policy.
This function will initialize APEX for power cap throttling. There are several environment variables that control power cap throttling:

**HPX_THROTTLING** If set, throttling will be enabled and initialized at startup.

**APEX_THROTTLING_MAX_THREADS** The maximum number of threads the throttling system will allow. The default value is 48.

**APEX_THROTTLING_MIN_THREADS** The minimum number of threads the throttling system will allow. The default value is 12.

**APEX_THROTTLING_MAX_WATTS** The maximum number of Watts the system can consume as an average rate. The default value is 220.

**APEX_THROTTLING_MIN_WATTS** The minimum number of Watts the system can consume as an average rate. The default value is 180.

**HPX_ENERGY_THROTTLING** If set, power/energy throttling will be performed.

**HPX_ENERGY** TBD

After evaluating the state of the system, the policy will set the thread cap, which can be queried using apex_get_thread_cap().

Returns

APEX_NOERROR on success, otherwise an error code.

5.2.1.24 int apex_setup_throughput_tuning ( apex_profiler_type type, void *identifier, apex_optimization_criteria_t criteria, apex_event_type event_type, int num_inputs, long **inputs, long *mins, long *maxs, long *steps )

Setup throttling to optimize for the specified function, using multiple input criteria.
This function will initialize a policy to optimize the specified function, using the list of tunable inputs for the specified function. The optimization criteria include maximizing throughput, minimizing or maximizing time spent in the specified function. After evaluating the state of the system, the policy will assign new values to the inputs.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong></td>
<td>The type of the address to be optimized. This can be one of the apex_profiler_type values.</td>
</tr>
<tr>
<td><strong>identifier</strong></td>
<td>The function address of the function to be optimized, or a “const char *” pointer to the name of the counter/timer.</td>
</tr>
<tr>
<td><strong>criteria</strong></td>
<td>The optimization criteria.</td>
</tr>
<tr>
<td><strong>event_type</strong></td>
<td>The apex_event_type that should trigger this policy</td>
</tr>
<tr>
<td><strong>num_inputs</strong></td>
<td>The number of tunable inputs for optimization</td>
</tr>
<tr>
<td><strong>inputs</strong></td>
<td>An array of addresses to inputs for optimization</td>
</tr>
<tr>
<td><strong>mins</strong></td>
<td>An array of minimum values for each input</td>
</tr>
<tr>
<td><strong>maxs</strong></td>
<td>An array of maximum values for each input</td>
</tr>
<tr>
<td><strong>steps</strong></td>
<td>An array of step values for each input</td>
</tr>
</tbody>
</table>

### Returns

APEX_NOERROR on success, otherwise an error code.

#### 5.2.1.25 int apex_setup_timer_throttling ( apex_profiler_type type, void *identifier, apex_optimization_criteria_t criteria, apex_optimization_method_t method, unsigned long update_interval )

Setup throttling to optimize for the specified function.

This function will initialize the throttling policy to optimize for the specified function. The optimization criteria include maximizing throughput, minimizing or maximizing time spent in the specified function. After evaluating the state of the system, the policy will set the thread cap, which can be queried using apex_get_thread_cap().

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong></td>
<td>The type of the address to be optimized. This can be one of the apex_profiler_type values.</td>
</tr>
<tr>
<td><strong>identifier</strong></td>
<td>The function address of the function to be optimized, or a “const char *” pointer to the name of the counter/timer.</td>
</tr>
<tr>
<td><strong>criteria</strong></td>
<td>The optimization criteria.</td>
</tr>
<tr>
<td><strong>method</strong></td>
<td>The optimization method.</td>
</tr>
<tr>
<td><strong>update_interval</strong></td>
<td>The time between observations, in microseconds.</td>
</tr>
</tbody>
</table>

### Returns

APEX_NOERROR on success, otherwise an error code.

#### 5.2.1.26 int apex_shutdown_throttling ( void )

Terminate the throttling policy.

This function will terminate the throttling policy.

### Returns

APEX_NOERROR on success, otherwise an error code.

#### 5.2.1.27 apex_profiler_handle apex_start ( apex_profiler_type type, void *identifier )

Start a timer.

This function will create a profiler object in APEX, and return a handle to the object. The object will be associated with the address or name passed in to this function. If both are zero (null) then the call will fail and the return value will be null.
5.2 /Users/khuck/src/xpress-apex/src/apex/apex.h File Reference 35

Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>The type of the address to be stored. This can be one of the apex_profiler_type values.</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>The function address of the function to be timed, or a &quot;const char *&quot; pointer to the name of the timer.</td>
</tr>
</tbody>
</table>

Returns

The handle for the timer object in APEX. Not intended to be queried by the application. Should be retained locally, if possible, and passed in to the matching apex_stop call when the timer should be stopped.

See also

apex_stop, apex_resume, apex_yield

5.2.1.28  void apex_stop ( apex_profiler_handle profiler )

Stop a timer.
This function will stop the specified profiler object, and queue the profiler to be processed out-of-band. The timer value will eventually added to the profile for the process.

Parameters

| profiler | The handle of the profiler object. |

Returns

No return value.

See also

apex_start, apex_yield, apex_resume

5.2.1.29  const char* apex_version ( void )

Return the APEX version.

Returns

A character string with the APEX version. This string should not be freed after the calling function is done with it.

5.2.1.30  void apex_yield ( apex_profiler_handle profiler )

Stop a timer, but don't increment the number of calls.
This function will stop the specified profiler object, and queue the profiler to be processed out-of-band. The timer value will eventually added to the profile for the process. The number of calls will NOT be incremented - this "task" was yielded, not completed. It will be resumed by another thread at a later time.

Parameters
### profiler

The handle of the profiler object.

**Returns**

No return value.

**See also**

`apex_start, apex_stop, apex_resume`

### 5.3 /Users/khuck/src/xpress-apex/src/apex/apex_api.hpp File Reference

#### Namespaces

- **apex**
  
  The main APEX namespace.

#### Functions

- **void apex::init (const char *thread_name)**
  
  Initialize APEX.

- **void apex::init (int argc, char **argv, const char *thread_name)**
  
  Initialize APEX.

- **void apex::finalize (void)**
  
  Finalize APEX.

- **void apex::cleanup (void)**
  
  Cleanup APEX.

- **profiler * apex::start (const std::string &timer_name)**
  
  Start a timer.

- **profiler * apex::start (apex_function_address function_address)**
  
  Start a timer.

- **void apex::stop (profiler *the_profiler)**
  
  Stop a timer.

- **void apex::yield (profiler *the_profiler)**
  
  Stop a timer, but don't increment the number of calls.

- **profiler * apex::resume (const std::string &timer_name)**
  
  Resume a timer.

- **profiler * apex::resume (apex_function_address function_address)**
  
  Resume a timer.

- **void apex::reset (const std::string &timer_name)**
  
  Reset a timer or counter.

- **void apex::reset (apex_function_address function_address)**
  
  Reset a timer.

- **void apex::set_state (apex_thread_state state)**
  
  Set the thread state.

- **void apex::sample_value (const std::string &name, double value)**
  
  Sample a state value.

- **void apex::new_task (const std::string &name, void *task_id)**
  
  Create a new task (dependency).

- **void apex::new_task (apex_function_address function_address, void *task_id)**
Create a new task (dependency).

- `apex_event_type apex::register_custom_event (const std::string &name)`
  Register an event type with APEX.
- `void apex::custom_event (apex_event_type event_type, void *custom_data)`
  Trigger a custom event.
- `std::string & apex::version (void)`
  Return the APEX version.
- `void apex::set_node_id (int id)`
  Set this process’ node ID.
- `void apex::register_thread (const std::string &name)`
  Register a new thread.
- `void apex::exit_thread (void)`
  Exit a thread.
- `apex_policy_handle * apex::register_policy (const apex_event_type when, std::function<int(apex_context const &)> f)`
  Register a policy with APEX.
- `std::set<apex_policy_handle *> apex::register_policy (std::set<apex_event_type> when, std::function<int(apex_context const &)> f)`
  Register a policy with APEX.
- `apex_policy_handle * apex::register_periodic_policy (unsigned long period, std::function<int(apex_context const &)> f)`
  Register a policy with APEX.
- `void apex::deregister_policy (apex_policy_handle *handle)`
  Deregister a policy with APEX.
- `apex_profile * apex::get_profile (apex_function_address function_address)`
  Get the current profile for the specified function address.
- `apex_profile * apex::get_profile (const std::string &timer_name)`
  Get the current profile for the specified function address.
- `int apex::setup_power_cap_throttling (void)`
  Initialize the power cap throttling policy.
- `int apex::setup_timer_throttling (apex_function_address the_address, apex_optimization_criteria_t criteria, apex_optimization_method_t method, unsigned long update_interval)`
  Setup throttling to optimize for the specified function.
- `int apex::setup_throughput_tuning (apex_function_address the_address, apex_optimization_criteria_t criteria, apex_event_type event_type, int num_inputs, long **inputs, long *mins, long *maxs, long *steps)`
  Setup throttling to optimize for the specified function, using multiple input criteria.
- `apex_tuning_session_handle apex::setup_custom_tuning (std::function< double(void)> metric, apex_event_type event_type, int num_inputs, long **inputs, long *mins, long *maxs, long *steps)`
  Setup tuning of specified parameters to optimize for a custom metric, using multiple input criteria.
- `apex_tuning_session_handle apex::setup_custom_tuning (apex_tuning_request &request)`
  Setup tuning of specified parameters to optimize for a custom metric, using multiple input criteria of potentially multiple types.
- `int apex::setup_timer_throttling (const std::string &the_name, apex_optimization_criteria_t criteria, apex_optimization_method_t method, unsigned long update_interval)`
  Setup throttling to optimize for the specified function or counter.
- `int apex::shutdown_throttling (void)`
  Terminate the throttling policy.
- `int apex::get_thread_cap (void)`
  Get the current thread cap set by the throttling.
- `void apex::set_thread_cap (int new_cap)`
  Set the current thread cap for throttling.
- `std::vector< std::pair< std::string, long * > > & apex::get_tunable_params (apex_tuning_session_handle h)`
  Get the tunable parameters.
Return a vector of the current tunable parameters.

- bool apex::has_session_converged (apex_tuning_session_handle handle)
  Check whether a tuning session has converged.
- void apex::print_options (void)
  Print out all configuration settings for APEX.

5.4 /Users/khuck/src/xpress-apex/src/apex/apex_types.h File Reference

#include <stdint.h>
#include <stdbool.h>
#include <unistd.h>

Classes
- struct apex_policy_handle
- struct apex_context
- struct apex_profile

Macros
- #define APEX_NULL_PROFILER_HANDLE (apex_profiler_handle)(NULL)
- #define APEX_MAX_EVENTS 128
- #define APEX_NULL_FUNCTION_ADDRESS 0L
- #define APEX_IDLE_TIME "APEX Idle"
- #define APEX_NON_IDLE_TIME "APEX Non-Idle"
- #define APEX_IDLE_RATE "APEX Idle Rate"

Typedefs
- typedef void * apex_profiler_handle
- typedef uintptr_t apex_function_address
- typedef int(*)( apex_policy_function ) (apex_context const context)
- typedef uint32_t apex_tuning_session_handle

Enumerations
- enum apex_profiler_type { APEX_FUNCTION_ADDRESS = 0, APEX_NAME_STRING }
- enum apex_error_code { APEX_NOERROR = 0, APEX_ERROR }
- enum apex_event_type { APEX_INVALID_EVENT = -1, APEX_STARTUP = 0, APEX_SHUTDOWN, APEX_NEW_NODE, APEX_NEW_THREAD, APEX_EXIT_THREAD, APEX_START_EVENT, APEX_RESUME_EVENT, APEX_STOP_EVENT, APEX_YIELD_EVENT, APEX_SAMPLE_VALUE, APEX_PERIODIC, APEX_CUSTOM_EVENT_1, APEX_CUSTOM_EVENT_2, APEX_CUSTOM_EVENT_3, APEX_CUSTOM_EVENT_4, APEX_CUSTOM_EVENT_5, APEX_CUSTOM_EVENT_6, APEX_CUSTOM_EVENT_7, APEX_CUSTOM_EVENT_8, APEX_UNUSED_EVENT = APEX_MAX_EVENTS }
- enum apex_thread_state { APEX_IDLE, APEX_BUSY, APEX_THROTTLED, APEX_WAITING, APEX_BLOCKED }

Generated on Wed May 4 2016 10:18:47 for Autonomic Performance Environment for eXascale (APEX) by Doxygen
• `enum apex_optimization_criteria_t { APEX_MAXIMIZE_THROUGHPUT, APEX_MAXIMIZE_ACCUMULATED, APEX_MINIMIZE_ACCUMULATED }`

• `enum apex_optimization_method_t { APEX_SIMPLE_HYSTHERESIS, APEX_DISCRETE_HILL_CLIMBING, APEX_ACTIVE_HARMONY }`

• `enum apex_profile_type { APEX_TIMER, APEX_COUNTER }`

5.4.1 Class Documentation

5.4.1.1 struct apex_policy_handle

A reference to the policy object, so that policies can be "unregistered", or paused later

Class Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>apex::event_type</code></td>
<td>event_type</td>
<td>The type of policy</td>
</tr>
<tr>
<td><code>int</code></td>
<td>id</td>
<td>The ID of the policy, used internally to APEX</td>
</tr>
<tr>
<td><code>unsigned long</code></td>
<td>period</td>
<td>If periodic, the length of the period</td>
</tr>
</tbody>
</table>

5.4.1.2 struct apex_context

The APEX context when an event occurs.

Class Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void*</code></td>
<td>data</td>
<td>Data associated with the event, such as the custom_data for a custom_event</td>
</tr>
<tr>
<td><code>apex::event_type</code></td>
<td>event_type</td>
<td>The type of the event currently processing</td>
</tr>
<tr>
<td><code>apex::policy_handle*</code></td>
<td>policy_handle</td>
<td>The policy handle for the current policy function</td>
</tr>
</tbody>
</table>

5.4.1.3 struct apex_profile

The profile object for a timer in APEX.

Class Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>double</code></td>
<td>accumulated</td>
<td>Accumulated values for all calls/samples</td>
</tr>
<tr>
<td><code>double</code></td>
<td>calls</td>
<td>Number of times a timer was called, or the number of samples collected for a counter</td>
</tr>
<tr>
<td><code>double</code></td>
<td>maximum</td>
<td>Maximum value seen by the timer or counter</td>
</tr>
<tr>
<td><code>double</code></td>
<td>minimum</td>
<td>Minimum value seen by the timer or counter</td>
</tr>
<tr>
<td><code>double</code></td>
<td>papi_metrics[8]</td>
<td>Array of accumulated PAPI hardware metrics</td>
</tr>
<tr>
<td><code>double</code></td>
<td>sum_squares</td>
<td>Running sum of squares calculation for all calls/samples</td>
</tr>
<tr>
<td><code>apex::profile_type</code></td>
<td>type</td>
<td>Whether this is a timer or a counter</td>
</tr>
</tbody>
</table>

5.4.2 Macro Definition Documentation

5.4.2.1 `#define APEX_IDLE_RATE "APEX Idle Rate"`

Special profile counter for derived idle rate
5.4.2.2 \texttt{#define APEX_IDLE\_TIME} "APEX Idle"

Special profile counter for derived idle time

5.4.2.3 \texttt{#define APEX\_MAX\_EVENTS} 128

The maximum number of event types. Allows for \(~20\) custom events.

5.4.2.4 \texttt{#define APEX\_NON\_IDLE\_TIME} "APEX Non-Idle"

Special profile counter for derived non-idle time

5.4.2.5 \texttt{#define APEX\_NULL\_FUNCTION\_ADDRESS} 0L

A null pointer representing an APEX function address. Used when a null APEX function address is to be passed in to any apex functions to represent "all functions".

5.4.2.6 \texttt{#define APEX\_NULL\_PROFILER\_HANDLE} (apex\_profiler\_handle)(NULL)

A null pointer representing an APEX profiler handle. Used when a null APEX profile handle is to be passed in to \texttt{apex::stop} when the profiler object wasn't retained locally.

5.4.3 Typedef Documentation

5.4.3.1 \texttt{typedef uintptr\_t} apex\_function\_address

Rather than use void pointers everywhere, be explicit about what the functions are expecting.

5.4.3.2 \texttt{typedef int(\star} apex\_policy\_function) (apex\_context const context)

Rather than use void pointers everywhere, be explicit about what the functions are expecting.

5.4.3.3 \texttt{typedef void\star} apex\_profiler\_handle

The address of a C++ object in APEX. Not useful for the caller that gets it back, but required for stopping the timer later.

5.4.3.4 \texttt{typedef uint32\_t} apex\_tuning\_session\_handle

A handle to a tuning session.

5.4.4 Enumeration Type Documentation

5.4.4.1 \texttt{enum apex\_error\_code}

Typedef for enumerating the different event types

Enumerators

\begin{itemize}
\item \texttt{APEX\_NOERROR} \hspace{5mm} No error occurred
\item \texttt{APEX\_ERROR} \hspace{5mm} Some error occurred - check stderr output for details
\end{itemize}
5.4.4.2 enum apex_event_type

Typedef for enumerating the different event types

Enumerator

APEX_STARTUP APEX is initialized
APEX_SHUTDOWN APEX is terminated
APEX_NEW_NODE APEX has registered a new process ID
APEX_NEW_THREAD APEX has registered a new OS thread
APEX_EXIT_THREAD APEX has exited an OS thread
APEX_START_EVENT APEX has processed a timer start event
APEX_RESUME_EVENT APEX has processed a timer resume event (the number of calls is not incremented)
APEX_STOP_EVENT APEX has processed a timer stop event
APEX_YIELD_EVENT APEX has processed a timer yield event
APEX_SAMPLE_VALUE APEX has processed a sampled value
APEX_PERIODIC APEX has processed a periodic timer
APEX_CUSTOM_EVENT_1 APEX has processed a custom event - useful for large granularity application control events

5.4.4.3 enum apex_optimization_criteria_t

Typedef for enumerating the different optimization strategies for throttling.

Enumerator

APEX_MAXIMIZE_THROUGHPUT maximize the number of calls to a timer/counter
APEX_MAXIMIZE_ACCUMULATED maximize the accumulated value of a timer/counter
APEX_MINIMIZE_ACCUMULATED minimize the accumulated value of a timer/counter

5.4.4.4 enum apex_optimization_method_t

Typedef for enumerating the different optimization methods for throttling.

Enumerator

APEX_SIMPLE_HYSTERESIS optimize using sliding window of historical observations. A running average of the most recent N observations are used as the measurement.
APEX_DISCRETE_HILL_CLIMBING Use a discrete hill climbing algorithm for optimization
APEX_ACTIVE_HARMONY Use Active Harmony for optimization.

5.4.4.5 enum apex_profile_type

The type of a profiler object

Enumerator

APEX_TIMER This profile is a instrumented timer
APEX_COUNTER This profile is a sampled counter
5.4.4.6 enum apex_profiler_type

Typedef for enumerating the different timer types

Enumerator

APEX_FUNCTION_ADDRESS The ID is a function (or instruction) address
APEX_NAME_STRING The ID is a character string

5.4.4.7 enum apex_thread_state

Typedef for enumerating the thread states.

Enumerator

APEX_IDLE Thread is idle
APEX_BUSY Thread is working
APEX_THROTTLED Thread is throttled (sleeping)
APEX_WAITING Thread is waiting for a resource
APEX_BLOCKED Thread is blocked

5.5 /Users/khuck/src/xpress-apex/src/comm/apex_global.h File Reference

#include "apex.h"

Functions

- int action_apex_reduce (void *unused)
  the function declaration, this is the function that does the reduction

- int action_apex_get_value (void *args)
  Each node has to populate their local value.

- int apex_periodic_policy_func (apex_context const context)
  A policy function to do periodic output.

- void apex_global_setup (apex_profiler_type type, void *in_action)
  The function to set up global reductions.

- void apex_global_teardown (void)
  The function to tear down global reductions, if necessary.

5.5.1 Function Documentation

5.5.1.1 int action_apex_get_value ( void * args )

Each node has to populate their local value.

Parameters

| args   | Local data to be exchanged globally. |

Returns

0 on no error.
5.5.1.2  int action_apex_reduce ( void * unused )

the function declaration, this is the function that does the reduction
Parameters

| unused | Unused value. |

Returns

0 on no error.

5.5.1.3 void apex_global_setup ( apex_profiler_type type, void * in_action )

The function to set up global reductions.

Parameters

<table>
<thead>
<tr>
<th>type</th>
<th>The type of the profiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_action</td>
<td>The name of a timer or address of a function. This is the function timer that should be reduced globally. This value is used for the example.</td>
</tr>
</tbody>
</table>

Returns

0 on no error.

5.5.1.4 int apex_periodic_policy_func ( apex_context const context )

A policy function to do periodic output.

Parameters

| context | The context for the periodic policy. |

Returns

0 on no error.
Index

/AUsers/khuck/src/xpress-apex/doc/apex.dox, 25
/AUsers/khuck/src/xpress-apex/src/apex/apex.h, 25
/AUsers/khuck/src/xpress-apex/src/apex/apex_api.hpp, 36
/AUsers/khuck/src/xpress-apex/src/apex/apex_types.h, 38
/AUsers/khuck/src/xpress-apex/src/comm/apex_global.h, 42

APEX_ACTIVE_HARMONY
    apex_types.h, 41
APEX_BLOCKED
    apex_types.h, 42
APEX_BUSY
    apex_types.h, 42
APEX_COUNTER
    apex_types.h, 41
APEX_CUSTOM_EVENT_1
    apex_types.h, 41
APEX_DISCRETE_HILL_CLIMBING
    apex_types.h, 41
APEX_ERROR
    apex_types.h, 40
APEX_EXIT_THREAD
    apex_types.h, 41
APEX_FUNCTION_ADDRESS
    apex_types.h, 42
APEX_IDLE
    apex_types.h, 42
APEX_IDLE_RATE
    apex_types.h, 39
APEX_IDLE_TIME
    apex_types.h, 39
APEX_MAX_EVENTS
    apex_types.h, 40
APEX_MAXIMIZE_ACCUMULATED
    apex_types.h, 41
APEX_MAXIMIZE_THROUGHPUT
    apex_types.h, 41
APEX_MINIMIZE_ACCUMULATED
    apex_types.h, 41
APEX_NAME_STRING
    apex_types.h, 42
APEX_NEW_NODE
    apex_types.h, 41
APEX_NEW_THREAD
    apex_types.h, 41
APEX_NOERROR
    apex_types.h, 40
APEX_NON_IDLE_TIME
    apex_types.h, 40
APEX_NULL_FUNCTION_ADDRESS
    apex_types.h, 40
APEX_NULL_PROFILER_HANDLE
    apex_types.h, 40
APEX_PERIODIC
    apex_types.h, 41
APEX_RESUME_EVENT
    apex_types.h, 41
APEX_SAMPLE_VALUE
    apex_types.h, 41
APEX_SHUTDOWN
    apex_types.h, 41
APEX_SIMPLE_HYSTERESIS
    apex_types.h, 41
APEX_START_EVENT
    apex_types.h, 41
APEX_STARTUP
    apex_types.h, 41
APEX_STOP_EVENT
    apex_types.h, 41
APEX_THROTTLED
    apex_types.h, 42
APEX_TIMER
    apex_types.h, 41
APEX_WAITING
    apex_types.h, 41
APEX_YIELD_EVENT
    apex_types.h, 42
APEX_YIELD_VALUE
    apex_types.h, 41
action_apex_get_value
    apex_global.h, 42
action_apex_reduce
    apex_global.h, 42
action
    9
cleanup
    11
custom_event
    11
deregister_policy
    11
exit_thread
    12
finalize
    12
get_profile
    12, 13
get_thread_cap
    13
get_tunable_params
    13
has_session_converged
    13
init
    13, 14
new_task
    14, 15
register_custom_event
    15
register_periodic_policy
    15
register_policy
    16
register_thread
    16
reset, 17
resume, 17
sample_value, 18
set_node_id, 18
set_state, 18
set_thread_cap, 19
setup_custom_tuning, 19
setup_power_cap_throttling, 19
setup_throughput_tuning, 20
setup_timer_throttling, 20, 22
shutdown_throttling, 22
start, 22, 23
stop, 23
version, 23
yield, 23

apex.h
apex_cleanup, 26
apex_current_power_high, 27
apex_custom_event, 27
apex_deregister_policy, 27
apex_exit_thread, 27
apex_finalize, 28
apex_get_profile, 28
apex_get_thread_cap, 28
apex_init, 28
apex_init_args, 29
apex_new_task, 29
apex_print_options, 30
apex_register_custom_event, 30
apex_register_periodic_policy, 30
apex_register_policy, 30
apex_register_thread, 31
apex_reset, 31
apex_resume, 31
apex_sample_value, 32
apex_set_node_id, 32
apex_set_state, 32
apex_set_thread_cap, 33
apex_setup_power_cap_throttling, 33
apex_setup_throughput_tuning, 33
apex_setup_timer_throttling, 34
apex_shutdown_throttling, 34
apex_start, 34
apex_stop, 35
apex_version, 35
apex_yield, 35
apex_cleanup
apex.h, 26
apex_context, 39
apex_current_power_high
apex.h, 27
apex_custom_event
apex.h, 27
apex_deregister_policy
apex.h, 27
apex_error_code
apex.h, 40
apex_event_type
apex_types.h, 40
apex_exit_thread
apex.h, 27
apex_finalize
apex.h, 28
apex_function_address
apex_types.h, 40
apex_get_profile
apex.h, 28
apex_get_thread_cap
apex.h, 28
apex_global.h
action_apex_get_value, 42
action_apex_reduce, 42
apex_global_setup
apex_global.h, 44
apex_init
apex.h, 28
apex_init_args
apex.h, 29
apex_new_task
apex.h, 29
apex_optimization_criteria_t
apex_types.h, 41
apex_optimization_method_t
apex_types.h, 41
apex_periodic_policy_func
apex_global.h, 44
apex_policy_function
apex_types.h, 40
apex_policy_handle, 39
apex_print_options
apex.h, 30
apex_profile
apex.h, 39
apex_profile_type
apex_types.h, 41
apex_profiler_handle
apex_types.h, 40
apex_profiler_type
apex_types.h, 41
apex_register_custom_event
apex.h, 30
apex_register_periodic_policy
apex.h, 30
apex_register_policy
apex.h, 30
apex_register_thread
apex.h, 31
apex_reset
apex.h, 31
apex_resume
apex.h, 31
apex_sample_value
apex.h, 32
apex_set_node_id
apex.h, 32
apex, 18
set_state
apex, 18
set_thread_cap
apex, 19
setup_custom_tuning
apex, 19
setup_power_cap_throttling
apex, 19
setup_throughput_tuning
apex, 20
setup_timer_throttling
apex, 20, 22
shutdown_throttling
apex, 22
start
apex, 22, 23
stop
apex, 23
version
apex, 23
yield
apex, 23