MIC COI API Reference Manual 0.65

Generated by Doxygen 1.7.3

Thu Sep 18 2014 14:29:38

Contents

1 MIC COI API Reference Manual 0.65

Disclaimer and Legal Information

Document Number:

World Wide Web: http://developer.intel.com

Intel Confidential

2 Disclaimer and Legal Information

Intel Confidential - This information contains highly sensitive technological or business information which could have a severely detrimental effect of disclosed to an unauthorized party.

All Intel Confidential media must be labeled and protected accordingly.

INTEL CORPORATION MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. INTEL CORPORATION ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT MAY APPEAR IN THIS DOCUMENT. INTEL CORPORATION MAKES NO COMMITMENT TO UPDATE NOR TO KEEP CURRENT THE INFORMATION CONTAINED IN THIS DOCUMENT. THIS SPECIFICATION IS COPYRIGHTED BY AND SHALL REMAIN THE PROPERTY OF INTEL CORPORATION. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED HEREIN. INTEL DISCLAIMS ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF ANY PROPRIETARY RIGHTS, RELATING TO IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. INTEL DOES NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATIONS WILL NOT INFRINGE SUCH RIGHTS. NO PART OF THIS DOCUMENT MAY BE COPIED OR

REPRODUCED IN ANY FORM OR BY ANY MEANS WITHOUT PRIOR WRITTEN CONSENT OF INTEL CORPORATION. INTEL CORPORATION RETAINS THE RIGHT TO MAKE CHANGES TO THESE SPECIFICATIONS AT ANY TIME, WITHOUT NOTICE.

Intel software products are copyrighted by and shall remain the property of Intel Corporation. Use, duplication or disclosure is subject to restrictions stated in Intel's Software License Agreement, or in the case of software delivered to the government, in accordance with the software license agreement as defined in FAR 52.227-7013.

The Intel logo is a registered trademark of Intel Corporation. Other brands and names are the property of their respective owners. Other names and brands may be claimed as the property of others. Copyright (C) 2007-2011, Intel Corporation. All rights reserved. Portions Copyright (C) 1996 John Birrell <jb@freebsd.org>. All rights reserved.

Portions of this document are reprinted and reproduced in electronic form in the FreeBSD∗ manual pages, from IEEE∗ Std 1003.1, 2004 Edition, Standard for Information Technology -- Portable Operating System Interface (POSIX∗), The Open Group Base Specifications Issue 6, Copyright (C) 2001-2004 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between these versions and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html.

3 Coprocessor Offload Infrastructure Overview

3.1 Overview

The Intel® Coprocessor Offload Infrastructure (Intel® COI) for Knights Corner is a software library designed to ease the development of software tools and applications that run on a discrete Knights Corner device. The primary usage model is for applications that run on the host processor (e.g. Intel(R) Xeon(R) processor) to launch and communicate with workloads on one or more Knights Corner cards. But the Intel® Coprocessor Offload Infrastructure (Intel® COI) model allows many different application configurations, including allowing applications running on Knights Corner to launch workloads on the host processor.

The Intel® Coprocessor Offload Infrastructure (Intel® COI) model exposes a pipelined programming model to the user. This model allows workloads to be run and data to be moved asynchronously, allowing the host processor, device processor, and DMA engines to stay busy. In the Intel® Coprocessor Offload Infrastructure (Intel® COI) pipelining model, work flows from a "source" to a "sink," either of which could be running on the host or device processors. Developers can configure one or more command pipelines to interact between sources and sinks. Commands on these pipelines are then run in an asynchronous, in-order fashion. This pipelined usage model exists in a number of offload environments, including graphics and network devices, and has been repeatedly shown to provide a balance between high performance and programmability. This model can also be used as the underpinnings on other popular programming models, including an RPC-like environment where device work is initiated by a large number of threads on the host.

The Intel® Coprocessor Offload Infrastructure (Intel® COI) model is agnostic with respect to how applications and workloads are written. It is a C-language API that interacts with workloads through standard API entry points, but does not impose or provide a framework for exploiting vector or thread parallelism on the host or the device. This allows Intel® Coprocessor Offload Infrastructure (Intel® COI) to be combined with any number of other programming models, including POSIX threads, Intel(R) Parallel Building Blocks, and Intel(R) compilers for both the host and the device.

3.2 Abstractions

Intel® Coprocessor Offload Infrastructure (Intel® COI) exposes four key abstractions to users, allowing them to accomplish tasks that would be otherwise difficult to accomplish using just lower-level abstractions.

- COIEngine - This abstraction allows an application to enumerate the Intel® Coprocessor Offload Infrastructure (Intel® COI) -capable devices in the system, including the host processor and any number of MIC devices in the same PCI root complex. The capabilities and dynamic load of the devices can be determined as well.
- COIProcess - The COIProcess abstraction encapsulates a process created by Intel® Coprocessor Offload Infrastructure (Intel® COI) on a remote engine. Creating an instance of a COIProcess creates a user process on a remote engine, and having a process handle allows an application to create buffers and pipeline objects that can be used by the process.
- COIPipeline - A pipeline is a uni-directional, asynchronous command stream between Intel® Coprocessor Offload Infrastructure (Intel® COI) processes. It allows remote functions to be run on a process running on another device. The process sending commands on a pipeline is called the "source" of the pipeline, and the process executing the commands is called the "sink" of the pipeline.
- COIBuffer - A COIBuffer object encapsulates data in a Intel® Coprocessor Offload Infrastructure (Intel® COI) system. Buffers can be created with various properties that affect their behavior. For example, a buffers can be created such that its virtual address is the same no matter where it is used, allowing pointers to be used internally to the buffer. An application can use a COIBuffer without thinking about if the physical memory for the buffer is in device or host memory, or it can decide to exert control over placement and movement if the application's buffer usage model lends itself to a particular data movement scheme.

In addition to these key API abstractions, Intel® Coprocessor Offload Infrastructure (Intel® COI) includes a few other useful abstractions. The COIEvent abstraction allows for synchronization between asynchronous commands, including functions run on a COIPipeline. And the COIPerf and COISysInfo abstractions offer utility libraries for people programming MIC devices.

4 File and Function Naming Conventions

4.1 General Concepts

Files and APIs may contain multiple version numbers, and will always contain at least one. Occasionally, you will find a minor version on a file or API. This minor version number will increment with less disruptive changes to the contents of a file or an API: new functions, signature changes, special versions of a function, etc.

4.2 Header Files

There are three types of header files:

- Headers for APIs that are portable to both the source and sink. Such headers are named COI<description>_common.h. COIResult_common.h and COIEvent_-common.h are examples, and are found in <install_dir>/install/common.
- Headers for APIs that can only be used in sink-specific code. Such headers are named COI<description>_sink.h. Examples are COIProcess_sink.h and COIBuffer_sink.h, and are found in <install_dir>/install/sink.
- Headers for APIs that only make sense on the source. Such headers are named COI<description>_source.h. Examples are COIProcess_source.h and COIPipeline_source.h, and are found in <install_dir>/install/source.

4.3 APIs

APIs follow a similar naming scheme to header files. Each API is named COI<sub-component>. Versioning in Linux is implemented using linker versioning, as described in http://sourceware.org/binutils/docs/ld/VERSION.html#VERSION.

```
// *** Current Major Release 2:
#if COI_LIBRARY_VERSION >= 2

COIRESULT
COIProcessLoadLibraryFromFile(
            COIPROCESS          in_Process,
    const   char*               in_pFileName,
    const   char*               in_pLibraryName,
    const   char*               in_LibrarySearchPath,
            uint32_t            in_Flags,
            COILIBRARY*         out_pLibrary);
__asm__(".symver COIProcessLoadLibraryFromFile,"
        "COIProcessLoadLibraryFromFile@COI_2.0");

#else

COIRESULT
COIProcessLoadLibraryFromFile(
            COIPROCESS          in_Process,
    const   char*               in_pFileName,
    const   char*               in_pLibraryName,
```

```
    const   char*               in_LibrarySearchPath,
            COILIBRARY*         out_pLibrary);
__asm__(".symver COIProcessLoadLibraryFromFile,"
        "COIProcessLoadLibraryFromFile@COI_1.0");

#endif
```

Currently, by default, each function binds to the earliest implementation to maintain compatibility with already existing code. Customers that wish to use the newer versions of the API can set the appropriate #define to take advantage of any new functionalities.

/*!

5 Module Documentation

5.1 COIBuffer

Modules

- COIBufferSource
- COIBufferSink

5.2 COIEngine

Modules

- COIEnginecommon
- COIEngineSource

5.3 COIResult

Modules

- COIResultCommon

5.4 COIPipeline

Modules

- COIPipelineSource
- COIPipelineSink

5.5 COIProcess

Modules

- COIProcessSource

- COIProcessSink

5.6 COIResultCommon

Typedefs

- typedef enum COIRESULT COIRESULT

Enumerations

- enum COIRESULT {
  COI_SUCCESS = 0,
  COI_ERROR,
  COI_NOT_INITIALIZED,
  COI_ALREADY_INITIALIZED,
  COI_ALREADY_EXISTS,
  COI_DOES_NOT_EXIST,
  COI_INVALID_POINTER,
  COI_OUT_OF_RANGE,
  COI_NOT_SUPPORTED,
  COI_TIME_OUT_REACHED,
  COI_MEMORY_OVERLAP,
  COI_ARGUMENT_MISMATCH,
  COI_SIZE_MISMATCH,
  COI_OUT_OF_MEMORY,
  COI_INVALID_HANDLE,
  COI_RETRY,
  COI_RESOURCE_EXHAUSTED,
  COI_ALREADY_LOCKED,
  COI_NOT_LOCKED,
  COI_MISSING_DEPENDENCY,
  COI_UNDEFINED_SYMBOL,
  COI_PENDING,
  COI_BINARY_AND_HARDWARE_MISMATCH,
  COI_PROCESS_DIED,
  COI_INVALID_FILE,
  COI_EVENT_CANCELED,
  COI_VERSION_MISMATCH,
  COI_BAD_PORT,
  COI_AUTHENTICATION_FAILURE,
  COI_NUM_RESULTS }

Functions

- COIACCESSAPI const char * COIResultGetName (COIRESULT in_ResultCode)

  Returns the string version of the passed in COIRESULT.

5.6.1 Typedef Documentation

5.6.1.1 typedef enum COIRESULT COIRESULT

5.6.2 Enumeration Type Documentation

5.6.2.1 enum COIRESULT

Enumerator:

COI_SUCCESS The function succeeded without error.

COI_ERROR Unspecified error.

COI_NOT_INITIALIZED The function was called before the system was initialized.

COI_ALREADY_INITIALIZED The function was called after the system was initialized.

COI_ALREADY_EXISTS Cannot complete the request due to the existence of a similar object.

COI_DOES_NOT_EXIST The specified object was not found.

COI_INVALID_POINTER One of the provided addresses was not valid.

COI_OUT_OF_RANGE One of the arguments contains a value that is invalid.

COI_NOT_SUPPORTED This function is not currently supported as used.

COI_TIME_OUT_REACHED The specified time out caused the function to abort.

COI_MEMORY_OVERLAP The source and destination range specified overlaps for the same buffer.

COI_ARGUMENT_MISMATCH The specified arguments are not compatible.

COI_SIZE_MISMATCH The specified size does not match the expected size.

COI_OUT_OF_MEMORY The function was unable to allocate the required memory.

COI_INVALID_HANDLE One of the provided handles was not valid.

COI_RETRY This function currently can't complete, but might be able to later.

COI_RESOURCE_EXHAUSTED The resource was not large enough.

COI_ALREADY_LOCKED The object was expected to be unlocked, but was locked.

COI_NOT_LOCKED The object was expected to be locked, but was unlocked.

COI_MISSING_DEPENDENCY One or more dependent components could not be found.

COI_UNDEFINED_SYMBOL One or more symbols the component required was not defined in any library.

COI_PENDING Operation is not finished.

COI_BINARY_AND_HARDWARE_MISMATCH A specified binary will not run on the specified hardware.

COI_PROCESS_DIED

COI_INVALID_FILE The file is invalid for its intended usage in the function.

COI_EVENT_CANCELED Event wait on a user event that was unregistered or is being unregistered returns COI_EVENT_CANCELED.

COI_VERSION_MISMATCH The version of Intel(R) Coprocessor Offload Infrastructure on the host is not compatible with the version on the device.

COI_BAD_PORT The port that the host is set to connect to is invalid.

COI_AUTHENTICATION_FAILURE The daemon was unable to authenticate the user that requested an engine. Only reported if daemon is set up for authorization.

COI_NUM_RESULTS Reserved, do not use.

Definition at line 52 of file COIResult_common.h.

5.6.3 Function Documentation

5.6.3.1 COIACCESSAPI const char* COIResultGetName (in_ResultCode)

Returns the string version of the passed in COIRESULT.

Thus if COI_RETRY is passed in, this function returns the string "COI_RETRY". If the error code passed ins is not valid then "COI_ERROR" will be returned.

Parameters:

in_ResultCode [in] COIRESULT code to return the string version of.

Returns:

String version of the passed in COIRESULT code.

5.7 COITypesSource

Data Structures

- struct coievent

Files

- file COITypes_common.h

Typedefs

- typedef uint64_t COI_CPU_MASK [16]
- typedef wchar_t coi_wchar_t

  On Windows, coi_wchar_t is a uint32_t.

- typedef struct coibuffer * COIBUFFER
- typedef struct coiengine * COIENGINE
- typedef struct coievent COIEVENT
- typedef struct coifunction * COIFUNCTION
- typedef struct coilibrary * COILIBRARY
- typedef struct coimapinst * COIMAPINSTANCE
- typedef struct coipipeline * COIPIPELINE
- typedef struct coiprocess * COIPROCESS

5.7.1 Typedef Documentation

5.7.1.1 typedef uint64_t COI_CPU_MASK[16]

Definition at line 69 of file COITypes_common.h.

5.7.1.2 typedef wchar_t coi_wchar_t

On Windows, coi_wchar_t is a uint32_t.

On Windows, wchar_t is 16 bits wide, and on Linux it is 32 bits wide, so uint32_t is used for portability.

Definition at line 74 of file COITypes_common.h.

5.7.1.3 typedef struct coibuffer* COIBUFFER

Definition at line 65 of file COITypes_common.h.

5.7.1.4 typedef struct coiengine* COIENGINE

Definition at line 63 of file COITypes_common.h.

5.7.1.5 typedef struct coievent COIEVENT

Definition at line 64 of file COITypes_common.h.

5.7.1.6 typedef struct coifunction* COIFUNCTION

Definition at line 62 of file COITypes_common.h.

5.7.1.7 typedef struct coilibrary* COILIBRARY

Definition at line 66 of file COITypes_common.h.

5.7.1.8 typedef struct coimapinst* COIMAPINSTANCE

Definition at line 67 of file COITypes_common.h.

5.7.1.9 typedef struct coipipeline* COIPIPELINE

Definition at line 61 of file COITypes_common.h.

5.7.1.10 typedef struct coiprocess* COIPROCESS

Definition at line 60 of file COITypes_common.h.

5.8 COIPerfCommon

Files

- file COIPerf_common.h
  Performance Analysis API.

Functions

- COIACCESSAPI uint64_t COIPerfGetCycleCounter (void)
  Returns a performance counter value.
- COIACCESSAPI uint64_t COIPerfGetCycleFrequency (void)
  Returns the calculated system frequency in hertz.

5.8.1 Function Documentation

5.8.1.1 COIACCESSAPI uint64_t COIPerfGetCycleCounter (

)

Returns a performance counter value.

This function returns a performance counter value that increments at a constant rate for all time and is coherent across all cores.

Returns:

Current performance counter value or 0 if no performance counter is available

5.8.1.2 COIACCESSAPI uint64_t COIPerfGetCycleFrequency (

)

Returns the calculated system frequency in hertz.

Returns:

Current system frequency in hertz.

5.9 COISysInfoCommon

Files

- file COISysInfo_common.h
  This interface allows developers to query the platform for system level information.

Defines

- #define INITIAL_APIC_ID_BITS 0xFF000000

Functions

- COIACCESSAPI uint32_t COISysGetAPICID (void)
- COIACCESSAPI uint32_t COISysGetCoreCount (void)
- COIACCESSAPI uint32_t COISysGetCoreIndex (void)
- COIACCESSAPI uint32_t COISysGetHardwareThreadCount (void)
- COIACCESSAPI uint32_t COISysGetHardwareThreadIndex (void)
- COIACCESSAPI uint32_t COISysGetL2CacheCount (void)
- COIACCESSAPI uint32_t COISysGetL2CacheIndex (void)

5.9.1 Define Documentation

5.9.1.1 #define INITIAL_APIC_ID_BITS 0xFF000000

Definition at line 56 of file COISysInfo_common.h.

5.9.2 Function Documentation

5.9.2.1 uint32_t COISysGetAPICID (

)

Returns:

The Advanced Programmable Interrupt Controller (APIC) ID of the hardware thread on which the caller is running.

Warning:

APIC IDs are unique to each hardware thread within a processor, but may not be sequential.

5.9.2.2 COIACCESSAPI uint32_t COISysGetCoreCount (

)

Returns:

The number of cores exposed by the processor on which the caller is running. Returns 0 if there is an error loading the processor info.

5.9.2.3 COIACCESSAPI uint32_t COISysGetCoreIndex (

)

Returns:

The index of the core on which the caller is running.

The indexes of neighboring cores will differ by a value of one and are within the range zero through COISysGetCoreCount()-1. Returns ((uint32_t)-1) if there was an error loading processor info.

5.9.2.4 COIACCESSAPI uint32_t COISysGetHardwareThreadCount (

)

Returns:

The number of hardware threads exposed by the processor on which the caller is running. Returns 0 if there is an error loading processor info.

5.9.2.5 COIACCESSAPI uint32_t COISysGetHardwareThreadIndex (

)

Returns:

The index of the hardware thread on which the caller is running.

The indexes of neighboring hardware threads will differ by a value of one and are within the range zero through COISysGetHardwareThreadCount()-1. Returns ((uint32_t)-1) if there was an error loading processor info.

5.9.2.6 COIACCESSAPI uint32_t COISysGetL2CacheCount (

)

Returns:

The number of level 2 caches within the processor on which the caller is running. Returns ((uint32_t)-1) if there was an error loading processor info.

5.9.2.7 COIACCESSAPI uint32_t COISysGetL2CacheIndex (

)

Returns:

The index of the level 2 cache on which the caller is running. Returns ((uint32_t)-1) if there was an error loading processor info.

The indexes of neighboring cores will differ by a value of one and are within the range zero through COISysGetL2CacheCount()-1.

5.10 COIEnginecommon

Files

- file COIEngine_common.h

Defines

- #define COI_MAX_ISA_KNC_DEVICES COI_MAX_ISA_MIC_DEVICES
- #define COI_MAX_ISA_KNF_DEVICES COI_MAX_ISA_MIC_DEVICES
- #define COI_MAX_ISA_MIC_DEVICES 128
- #define COI_MAX_ISA_x86_64_DEVICES 1

Enumerations

- enum COI_ISA_TYPE {
  COI_ISA_INVALID = 0,
  COI_ISA_x86_64,
  COI_ISA_MIC,
  COI_ISA_KNF,
  COI_ISA_KNC }

  List of ISA types of supported engines.

Functions

- COIACCESSAPI COIRESULT COIEngineGetIndex (COI_ISA_TYPE *out_pType, uint32_t *out_pIndex)

  Get the information about the COIEngine executing this function call.

5.10.1 Define Documentation

5.10.1.1 #define COI_MAX_ISA_KNC_DEVICES COI_MAX_ISA_MIC_DEVICES

Definition at line 57 of file COIEngine_common.h.

COI_ISA_INVALID Represents an invalid ISA.

COI_ISA_x86_64 The ISA for an x86_64 host engine.

COI_ISA_MIC Special value used to represent any device in the Intel(R) Many Integrated Core architecture family.

COI_ISA_KNF ISA for L1OM devices.

COI_ISA_KNC ISA for K1OM devices.

Definition at line 64 of file COIEngine_common.h.

5.10.3 Function Documentation

5.10.3.1 COIACCESSAPI COIRESULT COIEngineGetIndex (

out_pType,

out_pIndex)

Get the information about the COIEngine executing this function call.

Parameters:

out_pType [out] The COI_ISA_TYPE of the engine.

out_pIndex [out] The zero-based index of this engine in the collection of engines of the ISA returned in out_pType.

Returns:

COI_INVALID_POINTER if the any of the parameters are NULL.
COI_SUCCESS

5.11 COIEventcommon

Files

- file COIEvent_common.h

Functions

- COIACCESSAPI COIRESULT COIEventSignalUserEvent (COIEVENT in_Event)

  Signal one shot user event.

5.11.1 Function Documentation

5.11.1.1 COIACCESSAPI COIRESULT COIEventSignalUserEvent (in_Event)

Signal one shot user event.

User events created on source can be signaled from both sink and source. This fires the event and wakes up threads waiting on COIEventWait.

Note: For events that are not registered or already signaled this call will behave as a NOP. Users need to make sure that they pass valid events on the sink side.

Parameters:

in_Event Event Handle to be signaled.

Returns:

COI_INVAILD_HANDLE if in_Event was not a User event.
COI_ERROR if the signal fails to be sent from the sink.
COI_SUCCESS if the event was successfully signaled or ignored.

5.12 COIEventSource

Files

- file COIEvent_source.h

Defines

- #define COI_EVENT_ASYNC ((COIEVENT*)1)

  Special case event values which can be passed in to APIs to specify how the API should behave.

- #define COI_EVENT_SYNC ((COIEVENT*)2)

Functions

- COIACCESSAPI COIRESULT COIEventRegisterUserEvent (COIEVENT *out_pEvent)

  Register a User COIEVENT so that it can be fired.

- COIACCESSAPI COIRESULT COIEventUnregisterUserEvent (COIEVENT in_Event)

  Unregister a User COIEVENT.

- COIACCESSAPI COIRESULT COIEventWait (uint16_t in_NumEvents, const COIEVENT *in_pEvents, int32_t in_TimeoutMilliseconds, uint8_t in_WaitForAll, uint32_t *out_pNumSignaled, uint32_t *out_pSignaledIndices)

  Wait for an arbitrary number of COIEVENTs to be signaled as completed, eg when the run function or asynchronous map call associated with an event has finished execution.

5.12.1 Define Documentation

5.12.1.1 #define COI_EVENT_ASYNC ((COIEVENT*)1)

Special case event values which can be passed in to APIs to specify how the API should behave.

In COIBuffer APIs passing in NULL for the completion event is the equivalent of passing COI_EVENT_SYNC. Note that passing COI_EVENT_ASYNC can be used when the caller wishes the operation to be performed asynchronously but does not care when the operation completes. This can be useful for operations that by definition must complete in order (DMAs, run functions on a single pipeline). If the caller does care when the operation completes then they should pass in a valid completion event which they can later wait on.

Definition at line 65 of file COIEvent_source.h.

5.12.1.2 #define COI_EVENT_SYNC ((COIEVENT*)2)

Definition at line 66 of file COIEvent_source.h.

5.12.2 Function Documentation

5.12.2.1 COIACCESSAPI COIRESULT COIEventRegisterUserEvent (out_pEvent)

Register a User COIEVENT so that it can be fired.

Registered event is a one shot User event; in other words once signaled it cannot be used again for signaling. You have to unregister and register again to enable signaling. An event will be reset if it is re-registered without unregistering, resulting in loss of all outstanding signals.

Parameters:

out_pEvent [out] Pointer to COIEVENT handle being Registered

Returns:

COI_SUCCESS an event is successfully registered
COI_INVALID_POINTER if out_pEvent is NULL

5.12.2.2 COIACCESSAPI COIRESULT COIEventUnregisterUserEvent (in_Event)

Unregister a User COIEVENT.

Unregistering a unsignaled event is similar to firing an event. Except Calling COIEventWait on an event that is being unregistered returns COI_EVENT_CANCELED

Parameters:

in_Event [in] Event Handle to be unregistered.

Returns:

COI_INVALID_HANDLE if in_Event is not a UserEvent
COI_SUCCESS an event is successfully registered

5.12.2.3 COIACCESSAPI COIRESULT COIEventWait (

in_NumEvents,

in_pEvents,

in_TimeoutMilliseconds,

in_WaitForAll,

out_pNumSignaled,

out_pSignaledIndices)

Wait for an arbitrary number of COIEVENTs to be signaled as completed, eg when the run function or asynchronous map call associated with an event has finished execution.

If the user sets in_WaitForAll = True and not all of the events are signaled when the timeout period is reached then COI_TIME_OUT_REACHED will be returned. If the user sets in_WaitForAll = False then if at least one event is signaled when the timeout is reached then COI_SUCCESS is returned.

Parameters:

in_NumEvents [in] The number of events to wait for.

in_pEvents [in] The array of COIEVENT handles to wait for.

in_Timeout [in] The time in milliseconds to wait for the event. 0 polls and returns immediately, -1 blocks indefinitely.

in_WaitForAll [in] Boolean value specifying behavior. If true, wait for all events to be signaled, or for timeout, whichever happens first. If false, return when any event is signaled, or at timeout.

out_pNumSignaled [out] The number of events that were signaled. If in_NumEvents is 1 or in_WaitForAll = True, this parameter is optional.

out_pSignaledIndices [out] Pointer to an array of indicies into the original event array. Those denoted have been signaled. The user must provide an array that is no smaller than the in_Events array. If in_NumEvents is 1 or in_WaitForAll = True, this parameter is optional.

Returns:

COI_SUCCESS once an event has been signaled completed.
COI_TIME_OUT_REACHED if the events are still in use when the timeout is reached or timeout is zero (a poll).
COI_OUT_OF_RANGE if a negative value other than -1 is passed in to the in_-Timeout parameter.
COI_OUT_OF_RANGE if in_NumEvents is 0.

COI_INVALID_POINTER if in_pEvents is NULL.
COI_ARGUMENT_MISMATCH if in_NumEvents > 1 and if in_WaitForAll is not true and out_pSignaled or out_pSignaledIndicies are NULL.
COI_ARGUMENT_MISMATCH if out_pNumSignaled is not NULL and out_-pSignaledIndices is NULL (or vice versa).
COI_EVENT_CANCELED if while waiting on a user event, it gets unregistered this returns COI_EVENT_CANCELED
COI_PROCESS_DIED if the remote process died. See COIProcessDestroy for more details.
COI_<REAL ERROR> if only a single event is passed in, and that event failed, COI will attempt to return the real error code that caused the original operation to fail, otherwise COI_PROCESS_DIED is reported.

5.13 COIBufferSource

Data Structures

- struct arr_desc
- struct dim_desc

Defines

- #define COI_SINK_OWNERS ((COIPROCESS)-2)

Typedefs

- typedef enum COI_BUFFER_TYPE COI_BUFFER_TYPE

  The valid buffer types that may be created using COIBufferCreate.

- typedef enum COI_COPY_TYPE COI_COPY_TYPE

  The valid copy operation types for the COIBufferWrite, COIBufferRead, and COIBufferCopy APIs.

- typedef enum COI_MAP_TYPE COI_MAP_TYPE

  These flags control how the buffer will be accessed on the source after it is mapped.

Enumerations

- enum COI_BUFFER_MOVE_FLAG {
  COI_BUFFER_MOVE = 0,
  COI_BUFFER_NO_MOVE }

  Note: A VALID_MAY_DROP declares a buffer's copy as secondary on a given process.

- enum COI_BUFFER_STATE {
  COI_BUFFER_VALID = 0,
  COI_BUFFER_INVALID,
  COI_BUFFER_VALID_MAY_DROP,
  COI_BUFFER_RESERVED }

  The buffer states are used to indicate whether a buffer is available for access in a COIPROCESS.

- enum COI_BUFFER_TYPE {
  COI_BUFFER_NORMAL = 1,
  COI_BUFFER_STREAMING_TO_SINK,
  COI_BUFFER_STREAMING_TO_SOURCE,
  COI_BUFFER_PINNED,
  COI_BUFFER_OPENCL }

  The valid buffer types that may be created using COIBufferCreate.

- enum COI_COPY_TYPE {
  COI_COPY_UNSPECIFIED = 0,
  COI_COPY_USE_DMA,
  COI_COPY_USE_CPU,
  COI_COPY_UNSPECIFIED_MOVE_ENTIRE,
  COI_COPY_USE_DMA_MOVE_ENTIRE,
  COI_COPY_USE_CPU_MOVE_ENTIRE }

  The valid copy operation types for the COIBufferWrite, COIBufferRead, and COIBufferCopy APIs.

- enum COI_MAP_TYPE {
  COI_MAP_READ_WRITE = 1,
  COI_MAP_READ_ONLY,
  COI_MAP_WRITE_ENTIRE_BUFFER }

  These flags control how the buffer will be accessed on the source after it is mapped.

Functions

- COIACCESSAPI COIRESULT COIBufferAddRefcnt (COIPROCESS in_Process, COIBUFFER in_Buffer, uint64_t in_AddRefcnt)

  Increments the reference count on the specified buffer and process by in_AddRefcnt.

- COIACCESSAPI COIRESULT COIBufferCopy (COIBUFFER in_DestBuffer, COIBUFFER in_SourceBuffer, uint64_t in_DestOffset, uint64_t in_SourceOffset, uint64_t in_Length, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

Copy data between two buffers.

- COIACCESSAPI COIRESULT COIBufferCopyEx (COIBUFFER in_DestBuffer, const COIPROCESS in_DestProcess, COIBUFFER in_SourceBuffer, uint64_t in_DestOffset, uint64_t in_SourceOffset, uint64_t in_Length, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data between two buffers.

- COIACCESSAPI COIRESULT COIBufferCreate (uint64_t in_Size, COI_BUFFER_TYPE in_Type, uint32_t in_Flags, const void *in_pInitData, uint32_t in_NumProcesses, const COIPROCESS *in_pProcesses, COIBUFFER *out_pBuffer)

  Creates a buffer that can be used in RunFunctions that are queued in pipelines.

- COIACCESSAPI COIRESULT COIBufferCreateFromMemory (uint64_t in_Size, COI_BUFFER_TYPE in_Type, uint32_t in_Flags, void *in_Memory, uint32_t in_NumProcesses, const COIPROCESS *in_pProcesses, COIBUFFER *out_pBuffer)

  Creates a buffer from some existing memory that can be used in RunFunctions that are queued in pipelines.

- COIACCESSAPI COIRESULT COIBufferCreateSubBuffer (COIBUFFER in_Buffer, uint64_t in_Length, uint64_t in_Offset, COIBUFFER *out_pSubBuffer)

  Creates a sub-buffer that is a reference to a portion of an existing buffer.

- COIACCESSAPI COIRESULT COIBufferDestroy (COIBUFFER in_Buffer)

  Destroys a buffer.

- COIACCESSAPI COIRESULT COIBufferGetSinkAddress (COIBUFFER in_Buffer, uint64_t *out_pAddress)

  Gets the Sink's virtual address of the buffer.

- COIACCESSAPI COIRESULT COIBufferMap (COIBUFFER in_Buffer, uint64_t in_Offset, uint64_t in_Length, COI_MAP_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion, COIMAPINSTANCE *out_pMapInstance, void **out_ppData)

  This call initiates a request to access a region of a buffer.

- COIACCESSAPI COIRESULT COIBufferRead (COIBUFFER in_SourceBuffer, uint64_t in_Offset, void *in_pDestData, uint64_t in_Length, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data from a buffer into local memory.

- COIACCESSAPI COIRESULT COIBufferReadMultiD (COIBUFFER in_SourceBuffer, uint64_t in_Offset, struct arr_desc *in_DestArray, struct arr_desc *in_SrcArray, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data specified by multi-dimensional array data structure from an existing COIBUFFER to another multi-dimensional array located in memory.

- COIACCESSAPI COIRESULT COIBufferReleaseRefcnt (COIPROCESS in_Process, COIBUFFER in_Buffer, uint64_t in_ReleaseRefcnt)

  Releases the reference count on the specified buffer and process by in_ReleaseRefcnt.

- COIACCESSAPI COIRESULT COIBufferSetState (COIBUFFER in_Buffer, COIPROCESS in_Process, COI_BUFFER_STATE in_State, COI_BUFFER_MOVE_FLAG in_DataMove, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  This API allows an experienced Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) developer to set where a COIBUFFER is located and when the COIBUFFER's data is moved.

- COIACCESSAPI COIRESULT COIBufferUnmap (COIMAPINSTANCE in_MapInstance, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Disables Source access to the region of the buffer that was provided through the corresponding call to COIBufferMap.

- COIACCESSAPI COIRESULT COIBufferWrite (COIBUFFER in_DestBuffer, uint64_t in_Offset, const void *in_pSourceData, uint64_t in_Length, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data from a normal virtual address into an existing COIBUFFER.

- COIACCESSAPI COIRESULT COIBufferWriteEx (COIBUFFER in_DestBuffer, const COIPROCESS in_DestProcess, uint64_t in_Offset, const void *in_pSourceData, uint64_t in_Length, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data from a normal virtual address into an existing COIBUFFER.

- COIACCESSAPI COIRESULT COIBufferWriteMultiD (COIBUFFER in_DestBuffer, const COIPROCESS in_DestProcess, uint64_t in_Offset, struct arr_desc *in_DestArray, struct arr_desc *in_SrcArray, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data specified by multi-dimensional array data structure into another multi-dimensional array in an existing COIBUFFER.

COIBUFFER creation flags.

Please see the COI_VALID_BUFFER_TYPES_AND_FLAGS matrix below which describes the valid combinations of buffer types and flags.

- #define COI_SAME_ADDRESS_SINKS 0x00000001

  Create the buffer such that it has the same virtual address on all of the sink processes with which it is associated.

- #define COI_SAME_ADDRESS_SINKS_AND_SOURCE 0x00000002

  Create the buffer such that it has the same virtual address on all of the sink processes with which it is associated and in the source process.

- #define COI_OPTIMIZE_SOURCE_READ 0x00000004

  Hint to the runtime that the source will frequently read the buffer.

- #define COI_OPTIMIZE_SOURCE_WRITE 0x00000008

  Hint to the runtime that the source will frequently write the buffer.

- #define COI_OPTIMIZE_SINK_READ 0x00000010

  Hint to the runtime that the sink will frequently read the buffer.

- #define COI_OPTIMIZE_SINK_WRITE 0x00000020

  Hint to the runtime that the sink will frequently write the buffer.

- #define COI_OPTIMIZE_NO_DMA 0x00000040

  Used to delay the pinning of memory into physical pages, until required for DMA.

- #define COI_OPTIMIZE_HUGE_PAGE_SIZE 0x00000080

  Hint to the runtime to try to use huge page sizes for backing store on the sink.

- #define COI_SINK_MEMORY 0x00000100

  Used to tell Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) to create a buffer using memory that has already been allocated on the sink.

5.13.1 Define Documentation

5.13.1.1 #define COI_OPTIMIZE_HUGE_PAGE_SIZE 0x00000080

Hint to the runtime to try to use huge page sizes for backing store on the sink.

Is currently not compatible with PINNED buffers or the SAME_ADDRESS flags or the SINK_MEMORY flag. It is important to note that this is a hint and internally the runtime may not actually promote to huge pages. Specifically if the buffer is too small

(less than 4KiB for example) then the runtime will not promote the buffer to use huge pages.

Definition at line 135 of file COIBuffer_source.h.

5.13.1.2 #define COI_OPTIMIZE_NO_DMA 0x00000040

Used to delay the pinning of memory into physical pages, until required for DMA.

This can be used to delay the cost of time spent pinning memory until absolutely necessary. Might speed up the execution of COIBufferCreate calls, but slow down the first access of the buffer in COIPipelineRunFunction(s) or other COIBuffer access API's. Also of important note, that with this flag enabled COI will not be able to check to see if this memory is read only. Ordinarily this is checked and an error is thrown upon buffer creation. With this flag, the error might occur later, and cause undetermined behavior. Be sure to always use writeable memory for COIBuffers.

Definition at line 127 of file COIBuffer_source.h.

5.13.1.3 #define COI_OPTIMIZE_SINK_READ 0x00000010

Hint to the runtime that the sink will frequently read the buffer.

Definition at line 112 of file COIBuffer_source.h.

5.13.1.4 #define COI_OPTIMIZE_SINK_WRITE 0x00000020

Hint to the runtime that the sink will frequently write the buffer.

Definition at line 115 of file COIBuffer_source.h.

5.13.1.5 #define COI_OPTIMIZE_SOURCE_READ 0x00000004

Hint to the runtime that the source will frequently read the buffer.

Definition at line 106 of file COIBuffer_source.h.

5.13.1.6 #define COI_OPTIMIZE_SOURCE_WRITE 0x00000008

Hint to the runtime that the source will frequently write the buffer.

Definition at line 109 of file COIBuffer_source.h.

5.13.1.7 #define COI_SAME_ADDRESS_SINKS 0x00000001

Create the buffer such that it has the same virtual address on all of the sink processes with which it is associated.

Definition at line 99 of file COIBuffer_source.h.

5.13.1.8 #define COI_SAME_ADDRESS_SINKS_AND_SOURCE 0x00000002

Create the buffer such that it has the same virtual address on all of the sink processes with which it is associated and in the source process.

Definition at line 103 of file COIBuffer_source.h.

5.13.1.9 #define COI_SINK_MEMORY 0x00000100

Used to tell Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) to create a buffer using memory that has already been allocated on the sink.

This flag is only valid when passed in to the COIBufferCreateFromMemory API.

Definition at line 141 of file COIBuffer_source.h.

5.13.1.10 #define COI_SINK_OWNERS ((COIPROCESS)-2)

Definition at line 370 of file COIBuffer_source.h.

5.13.2 Typedef Documentation

5.13.2.1 typedef enum COI_BUFFER_TYPE COI_BUFFER_TYPE

The valid buffer types that may be created using COIBufferCreate.

Please see the COI_VALID_BUFFER_TYPES_AND_FLAGS matrix below which describes the valid combinations of buffer types and flags.

5.13.2.2 typedef enum COI_COPY_TYPE COI_COPY_TYPE

The valid copy operation types for the COIBufferWrite, COIBufferRead, and COIBufferCopy APIs.

5.13.2.3 typedef enum COI_MAP_TYPE COI_MAP_TYPE

These flags control how the buffer will be accessed on the source after it is mapped.

Please see the COI_VALID_BUFFER_TYPES_AND_MAP matrix below for the valid buffer type and map operation combinations.

5.13.3 Enumeration Type Documentation

5.13.3.1 enum COI_BUFFER_MOVE_FLAG

Note: A VALID_MAY_DROP declares a buffer's copy as secondary on a given process.

This means that there needs to be at least one primary copy of the the buffer somewhere in order to mark the buffer as VALID_MAY_DROP on a process. In other words to make a buffer VALID_MAY_DROP on a given process it needs to be in COI_BUFFER_VALID state somewhere else. The operation gets ignored (or is a nop) if there is no primary copy of the buffer. The nature of this state to "drop the content" when evicted is a side effect of marking the buffer as secondary copy. So when a buffer marked VALID_MAY_DROP is evicted Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) doesn't back it up as it is assumed that there is a primary copy somewhere. The buffer move flags are used to indicate when a buffer should be moved when it's state is changed. This is used with COIBufferSetState.

Enumerator:

COI_BUFFER_MOVE

COI_BUFFER_NO_MOVE

Definition at line 363 of file COIBuffer_source.h.

5.13.3.2 enum COI_BUFFER_STATE

The buffer states are used to indicate whether a buffer is available for access in a COIPROCESS.

This is used with COIBufferSetState.

Buffer state holds only for NORMAL Buffers and OPENCL buffers. Pinned buffers are always valid everywhere they get created. Streaming buffers do not follow the state transition rules, as a new version of the buffer is created every time it is Mapped or you issue a RunFunction.

Rules on State Transition of the buffer: -. When a Buffer is created by default it is valid only on the source, except for buffers created with COI_SINK_MEMORY flag which

are valid only on the sink where the memory lies when created. -. Apart from SetState following APIs also alters the state of the buffer internally:

- COIBufferMap alters state of buffer depending on the COI_MAP_TYPE. COI_-MAP_READ_ONLY: Makes Valid on the Source. Doesn't affect the state of the buffer on the other devices. COI_MAP_READ_WRITE: Makes it Valid only the Source and Invalid everywhere else. OPENCL buffers are invalidated only if it is not in use. COI_MAP_WRITE_ENTIRE_BUFFER: Makes it valid only on the Source. OPENCL buffers are invalidated only if not in use.

- COIPipelineRunfunction alters the state of the buffer depending on the COI_-ACCESS_FLAGS COI_SINK_READ: Makes it valid on the sink where Run-Function is being issued. Doesn't affect the state of the buffer on other devices. COI_SINK_WRITE: Makes it valid only on the sink where Runfunction is being issued and invalid everywhere else. OPENCL buffers are invalidated only if the buffer is not in use. COI_SINK_WRITE_ENTIRE: Makes it valid only on the sink where Runfunction is being issued and invalid everywhere else OPENCL buffers are invalidated only if the buffer is not in use.

- COIBufferWrite makes the buffer exclusively valid where the write happens. Write gives preference to Source over Sink. In other words if a buffer is valid on the Source and multiple Sinks, Write will happen on the Source and will In-validate all other Sinks. If the buffer is valid on multiple Sinks (and not on the Source) then Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) se-lects process handle with the lowest numerical value to do the exclusive write Again, OPENCL buffers are invalidated only if the buffer is not in use on that SINK/SOURCE.

The preference rule mentioned above holds true even for SetState API, when data needs to be moved from a valid location. The selection of valid location happens as stated above.

- It is possible to alter only parts of the buffer and change it state In other words it is possible for different parts of the buffer to have different states on different devices. A byte is the minimum size at which state can be maintained internally. Granularity level is completely determined by how the buffer gets fragmented.

Note: Buffer is considered 'in use' if is

- Being used in RunFunction : In use on a Sink
- Mapped: In use on a Source
- AddRef'd: In use on Sink The buffer states used with COIBufferSetState call to indicate the new state of the buffer on a given process

Enumerator:

COI_BUFFER_VALID

COI_BUFFER_INVALID
COI_BUFFER_VALID_MAY_DROP
COI_BUFFER_RESERVED

Definition at line 339 of file COIBuffer_source.h.

5.13.3.3 enum COI_BUFFER_TYPE

The valid buffer types that may be created using COIBufferCreate.

This matrix shows the valid combinations of buffer types and buffer flags that may be passed in to COIBufferCreate and COIBufferCreateFromMemory.

Please see the COI_VALID_BUFFER_TYPES_AND_FLAGS matrix below which describes the valid combinations of buffer types and flags.

```
static const uint64_t
COI_VALID_BUFFER_TYPES_AND_FLAGS[COI_BUFFER_OPENCL+1] = {
/*            |        | SAME |      |        |      |        |     |       |       |
              | SAME   | ADDR | OPT  | OPT    | OPT  | OPT    | OPT | HUGE  | COI   |
              | ADDR   | SINK | SRC  | SRC    | SINK | SINK   | NO  | PAGE  | SINK  |
              | SINKS  | SRC  | READ | WRITE  | READ | WRITE  | DMA | SIZE  | MEM   |
              +--------+------+------+--------+------+--------+-----+-------+------*/
MTM(INVALID  ,   F    ,   F  ,   F  ,   F    ,   F  ,   F    ,  F  ,   F   ,  F   ),
MTM(NORMAL   ,   T    ,   T  ,   T  ,   T    ,   T  ,   T    ,  T  ,   T   ,  T   ),
MTM(TO_SINK  ,   F    ,   F  ,   F  ,   T    ,   T  ,   T    ,  F  ,   F   ,  F   ),
MTM(TO_SOURCE,   F    ,   F  ,   T  ,   F    ,   F  ,   T    ,  F  ,   F   ,  F   ),
MTM(PINNED   ,   T    ,   T  ,   T  ,   T    ,   T  ,   T    ,  F  ,   F   ,  F   ),
MTM(OPENCL   ,   T    ,   T  ,   T  ,   T    ,   T  ,   T    ,  T  ,   T   ,  F   ),
};
```

Enumerator:

COI_BUFFER_NORMAL Normal buffers exist as a single physical buffer in either Source or Sink physical memory. Mapping the buffer may stall the pipelines.

COI_BUFFER_STREAMING_TO_SINK A streaming buffer creates new versions each time it is passed to Runfunction. These new versions are consumed by run functions. To_SINK buffers are used to send data from SOURCE to SINK These buffers are SOURCE write only buffers. If read, won't get Data written by SINK

COI_BUFFER_STREAMING_TO_SOURCE To_SOURCE buffers are used to get data from SINK to SOURCE These buffers are SOURCE Read only buffers. If written, data won't get reflected on SINK side.

COI_BUFFER_PINNED A pinned buffer exists in a shared memory region and is always available for read or write operations. Note: Pinned Buffers larger than 4KB are not supported in Windows 7 kernels.

COI_BUFFER_OPENCL OpenCL buffers are similar to Normal buffers except they don't stall pipelines and don't follow any read write dependencies.

Definition at line 60 of file COIBuffer_source.h.

5.13.3.4 enum COI_COPY_TYPE

The valid copy operation types for the COIBufferWrite, COIBufferRead, and COIBufferCopy APIs.

Enumerator:

COI_COPY_UNSPECIFIED The runtime can pick the best suitable way to copy the data.

COI_COPY_USE_DMA The runtime should use DMA to copy the data.

COI_COPY_USE_CPU The runtime should use a CPU copy to copy the data. CPU copy is a synchronous copy. So the resulting operations are always blocking (even though a out_pCompletion event is specified).

COI_COPY_UNSPECIFIED_MOVE_ENTIRE Same as above, but forces moving entire buffer to target process in Ex extended APIs, even if the full buffer is not written.

COI_COPY_USE_DMA_MOVE_ENTIRE Same as above, but forces moving entire buffer to target process in Ex extended APIs, even if the full buffer is not written.

COI_COPY_USE_CPU_MOVE_ENTIRE Same as above, but forces moving entire buffer to target process in Ex extended APIs, even if the full buffer is not written.

Definition at line 242 of file COIBuffer_source.h.

5.13.3.5 enum COI_MAP_TYPE

These flags control how the buffer will be accessed on the source after it is mapped.

This matrix shows the valid combinations of buffer types and map operations that may be passed in to COIBufferMap.

Please see the COI_VALID_BUFFER_TYPES_AND_MAP matrix below for the valid buffer type and map operation combinations.

```
static const uint64_t
COI_VALID_BUFFER_TYPES_AND_MAP
[COI_BUFFER_OPENCL+1][COI_MAP_WRITE_ENTIRE_BUFFER+1] = {
/*                      | MAP   | MAP   | MAP   |
                        | READ  | READ  | WRITE |
                        | WRITE | ONLY  | ENTIRE|
                        +-------+-------+-------+*/
MMM(INVALID             ,   F   ,   F   ,   F   ),
MMM(NORMAL              ,   T   ,   T   ,   T   ),
MMM(STREAMING_TO_SINK   ,   F   ,   F   ,   T   ),
MMM(STREAMING_TO_SOURCE ,   F   ,   T   ,   F   ),
MMM(PINNED              ,   T   ,   T   ,   T   ),
MMM(OPENCL              ,   T   ,   T   ,   T   ),
};
```

Enumerator:

COI_MAP_READ_WRITE Allows the application to read and write the contents of the buffer after it is mapped.

COI_MAP_READ_ONLY If this flag is set then the application must only read from the buffer after it is mapped. If the application writes to the buffer the contents will not be reflected back to the sink or stored for the next time the buffer is mapped on the source. This allows the runtime to make significant performance optimizations in buffer handling.

COI_MAP_WRITE_ENTIRE_BUFFER Setting this flag means that the source will overwrite the entire buffer once it is mapped. The app must not read from the buffer and must not expect the contents of the buffer to be synchronized from the sink side during the map operation. This allows the runtime to make significant performance optimizations in buffer handling.

Definition at line 185 of file COIBuffer_source.h.

5.13.4 Function Documentation

5.13.4.1 COIACCESSAPI COIRESULT COIBufferAddRefcnt (

in_Process,

in_Buffer,

in_AddRefcnt)

Increments the reference count on the specified buffer and process by in_AddRefcnt.

The returned result being COI_SUCCESS indicates that the specified process contains a reference to the specified buffer or a new reference has been created and that refernce has a new refcnt. Otherwise, if the buffer or process specified do not exist, then COI_-INVALID_HANDLE will be returned. If the input buffer is not valid on the target process then COI_NOT_INITIALIZED will be returned since the buffer is not current or allocated on the process.

Parameters:

in_Process [in] The COI Process whose reference count for the specified buffer the user wants to increment.

in_Buffer [in] The buffer used in the specified coi process in which the user wants to increment the reference count.

in_AddRefcnt [in] The value the reference count will be incremented by.

Returns:

COI_SUCCESS if the reference count was successfully incremented.

source and destination regions overlap then this API returns error. Note that it is not possible to use this API with any type of Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) Streaming Buffers. Please note that COIBufferCopy does not follow implicit buffer dependencies. If a buffer is in use in a run function or has been added to a process using COIBufferAddRef the call to COIBufferCopy will not wait, it will still copy data immediately. This is to facilitate a usage model where a buffer is being used outside of a run function, for example in a spawned thread, but data still needs to be transferred to or from the buffer.

Parameters:

in_DestBuffer [in] Buffer to copy into.

in_SourceBuffer [in] Buffer to copy from.

in_DestOffset [in] Location in the destination buffer to start writing to.

in_SourceOffset [in] Location in the source buffer to start reading from.

in_Length [in] The number of bytes to copy from in_SourceBuffer into in_DestinationBuffer. If the length is specified as zero then length to be copied Must not be larger than the size of in_SourceBuffer or in_DestBuffer and must not over run in_SourceBuffer or in_DestBuffer if offsets are

specified.

in_Type [in] The type of copy operation to use, one of either COI_COPY_UNSPECIFIED, COI_COPY_USE_DMA, COI_COPY_USE_CPU.

in_NumDependencies [in] The number of dependencies specified in the in_pDependencies array. This may be 0 if the caller does not want the copy call to wait for any additional events to be signaled before starting the copy operation.

in_pDependencies [in] An optional array of handles to previously created COIEVENT objects that this copy operation will wait for before starting. This allows the user to create dependencies between buffer copy calls and other operations such as run functions and map calls. The user may pass in NULL if they do not wish to wait for any additional dependencies to complete before doing the copy.

out_pCompletion [out] An optional event to be signaled when the copy has completed. This event can be used as a dependency to order the copy with regard to future operations. If no completion event is passed in then the copy is synchronous and will block until the transfer is complete.

Returns:

COI_SUCCESS if the buffer was copied successfully.
COI_INVALID_HANDLE if either buffer handle was invalid.
COI_MEMORY_OVERLAP if in_SourceBuffer and in_DestBuffer are the same buffer(or have the same parent buffer) and the source and destination regions overlap
COI_OUT_OF_RANGE if in_DestOffset is is beyond the end of in_DestBuffer
COI_OUT_OF_RANGE if in_SourceOffset is beyond the end of in_SourceBuffer.
COI_OUT_OF_RANGE if in_DestOffset + in_Length exceeds the size of the in_DestBuffer
COI_OUT_OF_RANGE if in_SourceOffset + in_Length exceeds the size of in_SourceBuffer.
COI_ARGUMENT_MISMATCH if the in_pDependencies is non NULL but in_NumDependencies is 0.
COI_ARGUMENT_MISMATCH if in_pDependencies is NULL but in_NumDependencies is not 0.
COI_NOT_SUPPORTED if the source or destination buffers are of type COI_BUFFER_STREAMING_TO_SINK or COI_BUFFER_STREAMING_TO_SOURCE.
COI_NOT_SUPPORTED if either buffer is of type COI_BUFFER_STREAMING_TO_SINK or COI_BUFFER_STREAMING_TO_SOURCE.
COI_RETRY if in_DestBuffer or in_SourceBuffer are mapped and not COI_BUFFER_PINNED buffers or COI_BUFFER_OPENCL buffers.

5.13.4.3 COIACCESSAPI COIRESULT COIBufferCopyEx (

in_DestBuffer,

Parameters:

in_DestBuffer [in] Buffer to copy into.

in_DestProcess [in] A pointer to the process to which the data will be written. Buffer is updated only in this process and invalidated in other processes. Only a single process can be specified. Can be left NULL and default behavior will be chosen, which chooses the first valid process in which regions are found. Other buffer regions are invalidated if not updated.

in_SourceBuffer [in] Buffer to copy from.

in_DestOffset [in] Location in the destination buffer to start writing to.

in_SourceOffset [in] Location in the source buffer to start reading from.

in_Length [in] The number of bytes to copy from in_SourceBuffer into in_DestinationBuffer. If the length is specified as zero then length to be copied Must not be larger than the size of in_SourceBuffer or in_DestBuffer and must not over run in_SourceBuffer or in_DestBuffer if offsets are specified.

in_Type [in] The type of copy operation to use, one of either COI_COPY_UNSPECIFIED, COI_COPY_USE_DMA, COI_COPY_USE_CPU.

in_NumDependencies [in] The number of dependencies specified in the in_pDependencies array. This may be 0 if the caller does not want the copy call to wait for any additional events to be signaled before starting the copy operation.

in_pDependencies [in] An optional array of handles to previously created COIEVENT objects that this copy operation will wait for before starting. This allows the user to create dependencies between buffer copy calls and other operations such as run functions and map calls. The user may pass in NULL if they do not wish to wait for any additional dependencies to complete before doing the copy.

out_pCompletion [out] An optional event to be signaled when the copy has completed. This event can be used as a dependency to order the copy with regard to future operations. If no completion event is passed in then the copy is synchronous and will block until the transfer is complete.

Returns:

COI_SUCCESS if the buffer was copied successfully.
COI_INVALID_HANDLE if either buffer handle was invalid.
COI_MEMORY_OVERLAP if in_SourceBuffer and in_DestBuffer are the same buffer(or have the same parent buffer) and the source and destination regions overlap
COI_OUT_OF_RANGE if in_DestOffset is is beyond the end of in_DestBuffer
COI_OUT_OF_RANGE if in_SourceOffset is beyond the end of in_SourceBuffer.
COI_OUT_OF_RANGE if in_DestOffset + in_Length exceeds the size of the in_DestBuffer
COI_OUT_OF_RANGE if in_SourceOffset + in_Length exceeds the size of in_SourceBuffer.
COI_ARGUMENT_MISMATCH if the in_pDependencies is non NULL but in_NumDependencies is 0.
COI_ARGUMENT_MISMATCH if in_pDependencies is NULL but in_NumDependencies is not 0.
COI_NOT_SUPPORTED if the source or destination buffers are of type COI_BUFFER_STREAMING_TO_SINK or COI_BUFFER_STREAMING_TO_SOURCE.
COI_NOT_SUPPORTED if either buffer is of type COI_BUFFER_STREAMING_TO_SINK or COI_BUFFER_STREAMING_TO_SOURCE.
COI_RETRY if in_DestBuffer or in_SourceBuffer are mapped and not COI_BUFFER_PINNED buffers or COI_BUFFER_OPENCL buffers.

5.13.4.4 COIACCESSAPI COIRESULT COIBufferCreate (

in_Size,

in_Type,

in_Flags,

in_pInitData,

in_NumProcesses,

in_pProcesses,

out_pBuffer)

Creates a buffer that can be used in RunFunctions that are queued in pipelines.

The address space for the buffer is reserved when it is created although the memory may not be committed until the buffer is used for the first time. Please note that the Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) runtime may also allocate space for the source process to use as shadow memory for certain types of buffers. If Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) does allocate this memory it will not be released or reallocated until the COIBuffer is destroyed.

Parameters:

in_Size [in] The number of bytes to allocate for the buffer. If in_Size is not page aligned, it will be rounded up.

in_Type [in] The type of the buffer to create.

in_Flags [in] A bitmask of attributes for the newly created buffer. Some of these flags are required for correctness while others are provided as hints to the runtime system so it can make certain performance optimizations.

in_pInitData [in] If non-NULL the buffer will be initialized with the data pointed to by pInitData. The memory at in_pInitData must hold at least in_Size bytes.

in_NumProcesses [in] The number of processes with which this buffer might be used.

in_pProcesses [in] An array of COIPROCESS handles identifying the processes with which this buffer might be used.

out_pBuffer [out] Pointer to a buffer handle. The handle will be filled in with a value that uniquely identifies the newly created buffer. This handle should be disposed of via COIBufferDestroy() once it is no longer needed.

Returns:

COI_SUCCESS if the buffer was created
COI_ARGUMENT_MISMATCH if the in_Type and in_Flags parameters are not compatible with one another. Please see the COI_VALID_BUFFER_TYPES_-AND_FLAGS map above for information about which flags and types are compatible.

COI_OUT_OF_RANGE if in_Size is zero, if the bits set in the in_Flags parameter are not recognized flags, or if in_NumProcesses is zero.
COI_INVALID_POINTER if the in_pProcesses or out_pBuffer parameter is NULL.
COI_NOT_SUPPORTED if one of the in_Flags is COI_SINK_MEMORY.
COI_NOT_SUPPORTED if the flags include either COI_SAME_ADDRESS_-SINKS or COI_SAME_ADDRESS_SINKS_AND_SOURCE and COI_-OPTIMIZE_HUGE_PAGE_SIZE.
COI_INVALID_HANDLE if one of the COIPROCESS handles in the in_-pProcesses array does not identify a valid process.
COI_OUT_OF_MEMORY if allocating the buffer fails.
COI_RESOURCE_EXHAUSTED if the sink is out of buffer memory. This error can also be thrown from Windows 7 operating systems if COI_BUFFER_PINNED and a size larger than 4KB is requested. This is due to a limitation of the Windows 7 memory management unit.

5.13.4.5 COIACCESSAPI COIRESULT COIBufferCreateFromMemory (

in_Size,

in_Type,

in_Flags,

in_Memory,

in_NumProcesses,

in_pProcesses,

out_pBuffer)

Creates a buffer from some existing memory that can be used in RunFunctions that are queued in pipelines.

If the flag COI_SINK_MEMORY is specified then Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) will use that memory for the buffer on the sink. If that flag isn't set then the memory provided is used as backing store for the buffer on the source. In either case the memory must not be freed before the buffer is destroyed. While the user still owns the memory passed in they must use the appropriate access flags when accessing the buffer in COIPipelinRunFunction or COIBufferMap calls so that the runtime knows when the memory has been modified. If the user just writes directly to the memory location then those changes may not be visible when the corresponding buffer is accessed. Whatever values are already present in the memory location when this call is made are preserved. The memory values are also preserved when COIBufferDestroy is called.

Warning:

: Use of this function is highly discouraged if the calling program forks at all

(including calls to system(3), popen(3), or similar functions) during the life of this buffer. See the discussion around the in_Memory parameter below regarding this.

Parameters:

in_Size [in] The size of in_Memory in bytes. If in_Size is not page aligned, it will be rounded up.

in_Type [in] The type of the buffer to create. Note that streaming buffers can not be created from user memory. Only COI_BUFFER_NORMAL and COI_BUFFER_PINNED buffer types are supported.

in_Flags [in] A bitmask of attributes for the newly created buffer. Some of these flags are required for correctness while others are provided as hints to the runtime system so it can make certain performance optimizations. Note that the flag COI_SAME_ADDRESS_SINKS_AND_SOURCE is still valid but may fail if the same address as in_Memory can not be allocated on the sink.

in_Memory [in] A pointer to an already allocated memory region that should be turned into a COIBUFFER. Although the user still owns this memory they should not free it before calling COIBufferDestroy. They must also only access the memory using COIBUFFER semantics, for example using COIBufferMap/COIBufferUnmap when they wish to read or write the data. There are no alignment or size requirements for this memory region.

WARNING: Since the backing memory passed in can be the target of a DMA the caller must ensure that there is no call to clone(2) (without the CLONE_VM argument) during the life of this buffer. This includes higher level functions that call clone such as fork(2), system(3), popen(3), among others).

For forked processes, Linux uses copy-on-write semantics for performances reasons. Conseqeuently, if the parent forks and then writes to this memory, the physical page mapping changes causing the DMA to fail (and thus data corruption).

In Linux you can mark a set of pages to not be copied across across the clone by calling madvise(2) with an argument of MADV_DONTFORK and then safely use that memory in this scenario. Alternately, if the memory is from a region marked MAP_-SHARED, this will work.

Parameters:

in_NumProcesses [in] The number of processes with which this buffer might be used. If the flag COI_SINK_MEMORY is specified then this must be 1.

in_pProcesses [in] An array of COIPROCESS handles identifying the processes with which this buffer might be used.

out_pBuffer [out] Pointer to a buffer handle. The handle will be filled in with a value that uniquely identifies the newly created buffer. This handle should be disposed of via COIBufferDestroy() once it is no longer needed.

Returns:

COI_SUCCESS if the buffer was created
COI_NOT_SUPPORTED if the in_Type value is not COI_BUFFER_NORMAL or COI_BUFFER_PINNED.
COI_NOT_SUPPORTED if in_Memory is read-only memory
COI_NOT_SUPPORTED if one of the in_Flags is COI_SINK_MEMORY and in_Type is not COI_BUFFER_NORMAL
COI_NOT_SUPPORTED if the flag COI_SAME_ADDRESS_SINKS is set
COI_NOT_SUPPORTED if the flag COI_SAME_ADDRESS_SINKS_AND_SOURCE is set
COI_ARGUMENT_MISMATCH if the in_Type and in_Flags parameters are not compatible with one another. Please see the COI_VALID_BUFFER_TYPES_AND_FLAGS map above for information about which flags and types are compatible.
COI_ARGUMENT_MISMATCH if the flag COI_SINK_MEMORY is specified and in_NumProcesses > 1.
COI_ARGUMENT_MISMATCH if the flags COI_SINK_MEMORY and COI_OPTIMIZE_HUGE_PAGE_SIZE are both set.
COI_OUT_OF_RANGE if in_Size is zero, if the bits set in the in_Flags parameter are not recognized flags, or if in_NumProcesses is zero.
COI_INVALID_POINTER if in_Memory, in_pProcesses or out_pBuffer parameter is NULL.
COI_INVALID_HANDLE if one of the COIPROCESS handles in the in_pProcesses array does not identify a valid process.

5.13.4.6 COIACCESSAPI COIRESULT COIBufferCreateSubBuffer (

in_Buffer,

in_Length,

in_Offset,

out_pSubBuffer)

Creates a sub-buffer that is a reference to a portion of an existing buffer.

The returned buffer handle can be used in all API calls that the original buffer handle could be used in except COIBufferCreateSubBuffer. Sub buffers out of Huge Page Buffer are also supported but the original buffer needs to be a OPENCL buffer created with COI_OPTIMIZE_HUGE_PAGE_SIZE flag.

When the sub-buffer is used only the corresponding sub-section of the original buffer is used or affected.

Parameters:

in_Buffer [in] The original buffer that this new sub-buffer is a reference to.

in_Length [in] The length of the sub-buffer in number of bytes.

in_Offset [in] Where in the original buffer to start this sub-buffer.

out_pSubBuffer [out] Pointer to a buffer handle that is filled in with the newly created sub-buffer.

Returns:

COI_SUCCESS if the sub-buffer was created
COI_INVALID_HANDLE if in_Buffer is not a valid buffer handle.
COI_OUT_OF_RANGE if in_Length is zero, or if in_Offset + in_Length is greater than the size of the original buffer.
COI_OUT_OF_MEMORY if allocating the buffer fails.
COI_INVALID_POINTER if the out_pSubBuffer pointer is NULL.
COI_NOT_SUPPORTED if the in_Buffer is of any type other than COI_-BUFFER_OPENCL

5.13.4.7 COIACCESSAPI COIRESULT COIBufferDestroy (

in_Buffer)

Destroys a buffer.

Will block on completion of any operations on the buffer, such as COIPipelineRun-Function or COIBufferCopy. Will block until all COIBufferAddRef calls have had a matching COIBufferReleaseRef call made. Will not block on an outstanding COIBufferUnmap but will instead return COI_RETRY.

Parameters:

in_Buffer [in] Handle of the buffer to destroy.

Returns:

COI_SUCCESS if the buffer was destroyed.
COI_INVALID_HANDLE if the buffer handle was invalid.
COI_RETRY if the buffer is currently mapped. The buffer must first be unmapped before it can be destroyed.
COI_RETRY if the sub-buffers created from this buffer are not yet destroyed

5.13.4.8 COIACCESSAPI COIRESULT COIBufferGetSinkAddress (

in_Buffer,

out_pAddress)

Gets the Sink's virtual address of the buffer.

This is the same address that is passed to the run function on the Sink. The virtual address assigned to the buffer for use on the sink is fixed; the buffer will always be present at that virtual address on the sink and will not get a different virtual address across different RunFunctions. This address is only valid on the Sink and should not be dereferenced on the Source (except for the special case of buffers created with the COI_SAME_ADDRESS flag).

Parameters:

in_Buffer [in] Buffer handle

out_pAddress [out] pointer to a uint64_t* that will be filled with the address.

Returns:

COI_SUCCESS upon successful return of the buffer's address.
COI_INVALID_HANDLE if the passed in buffer handle was invalid.
COI_INVALID_POINTER if the out_pAddress parameter was invalid.
COI_NOT_SUPPORTED if the buffer passed in is of type COI_BUFFER_-STREAMING_TO_SOURCE or COI_BUFFER_STREAMING_TO_SINK.

5.13.4.9 COIACCESSAPI COIRESULT COIBufferMap (

in_Buffer,

in_Offset,

in_Length,

in_Type,

in_NumDependencies,

in_pDependencies,

out_pCompletion,

out_pMapInstance,

out_ppData)

This call initiates a request to access a region of a buffer.

Multiple overlapping (or non overlapping) regions can be mapped simultaneously for any given buffer. If a completion event is specified this call will queue a request for

at the beginning of the buffer.

---

in_Length [in] Length of the buffer area to map. This parameter, in combination with in_Offset, allows the caller to specify that only a subset of an entire buffer need be mapped. A value of 0 can be passed in only if in_Offset is 0, to signify that the mapped region is the entire buffer.

---

in_Type [in] The access type that is needed by the application. This will affect how the data can be accessed once the map operation completes. See the COI_MAP_TYPE enum for more details.

---

in_NumDependencies [in] The number of dependencies specified in the in_pDependencies array. This may be 0 if the caller does not want the map call initiation to wait for any events to be signaled before starting the map operations.

---

in_pDependencies [in] An optional array of handles to previously created COIEVENT objects that this map operation will wait for before starting. This allows the user to create dependencies between asynchronous map calls and other operations such as run functions or other asynchronous map calls. The user may pass in NULL if they do not wish to wait for any dependencies to complete before initiating map operations.

---

out_pCompletion [out] An optional pointer to a COIEVENT object that will be signaled when a map call with the passed in buffer would complete immediately, that is, the buffer memory has been allocated on the source and its contents updated. The user may pass in NULL if the user wants COIBufferMap to perform a blocking map operation.

out_pMapInstance [out] A pointer to a COIMAPINSTANCE which represents this mapping of the buffer and must be passed in to COIBufferUnmap when access to this region of the buffer data is no longer needed.

out_ppData [out] Pointer to the buffer data. The data will only be valid when the completion object is signaled, or for a synchronous map operation with the call to map returns.

Returns:

COI_SUCCESS if the map request succeeds.
COI_OUT_OF_RANGE if in_Offset of (in_Offset + in_Length) exceeds the size of the buffer.
COI_OUT_OF_RANGE if in_Length is 0, but in_Offset is not 0.
COI_OUT_OF_RANGE if in_Type is not a valid COI_MAP_TYPE.
COI_ARGUMENT_MISMATCH if in_NumDependencies is non-zero while in_pDependencies was passed in as NULL.
COI_ARGUMENT_MISMATCH if in_pDependencies is non-NULL but in_NumDependencies is zero.
COI_ARGUMENT_MISMATCH if the in_Type of map is not a valid type for in_Buffer's type of buffer.
COI_RESOURCE_EXHAUSTED if could not create a version for TO_SINK streaming buffer. It can fail if enough memory is not available to register. This call will succeed eventually when the registered memory becomes available.
COI_INVALID_HANDLE if in_Buffer is not a valid buffer handle.
COI_INVALID_POINTER if out_pMapInstance or out_ppData is NULL.

5.13.4.10 COIACCESSAPI COIRESULT COIBufferRead (

in_SourceBuffer,

in_Offset,

in_pDestData,

in_Length,

in_Type,

in_NumDependencies,

in_pDependencies,

out_pCompletion)

Copy data from a buffer into local memory.

Note that it is not possible to use this API with any type of Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) Streaming Buffers. Please note that COIBufferRead does not follow implicit buffer dependencies. If a buffer is in use in a run function or has been added to a process using COIBufferAddRef the call to COIBufferRead will not wait, it will still copy data immediately. This is to facilitate a usage model where a buffer is being used outside of a run function, for example in a spawned thread, but data still needs to be transferred to or from the buffer.

Parameters:

in_SourceBuffer [in] Buffer to read from.

in_Offset [in] Location in the buffer to start reading from.

in_pDestData [in] A pointer to local memory that should be written into from the provided buffer.

in_Length [in] The number of bytes to write from in_SourceBuffer into in_pDestData. Must not be larger than the size of in_SourceBuffer and must not over run in_SourceBuffer if an in_Offset is provided.

in_Type [in] The type of copy operation to use, one of either COI_COPY_-UNSPECIFIED, COI_COPY_USE_DMA, COI_COPY_USE_CPU.

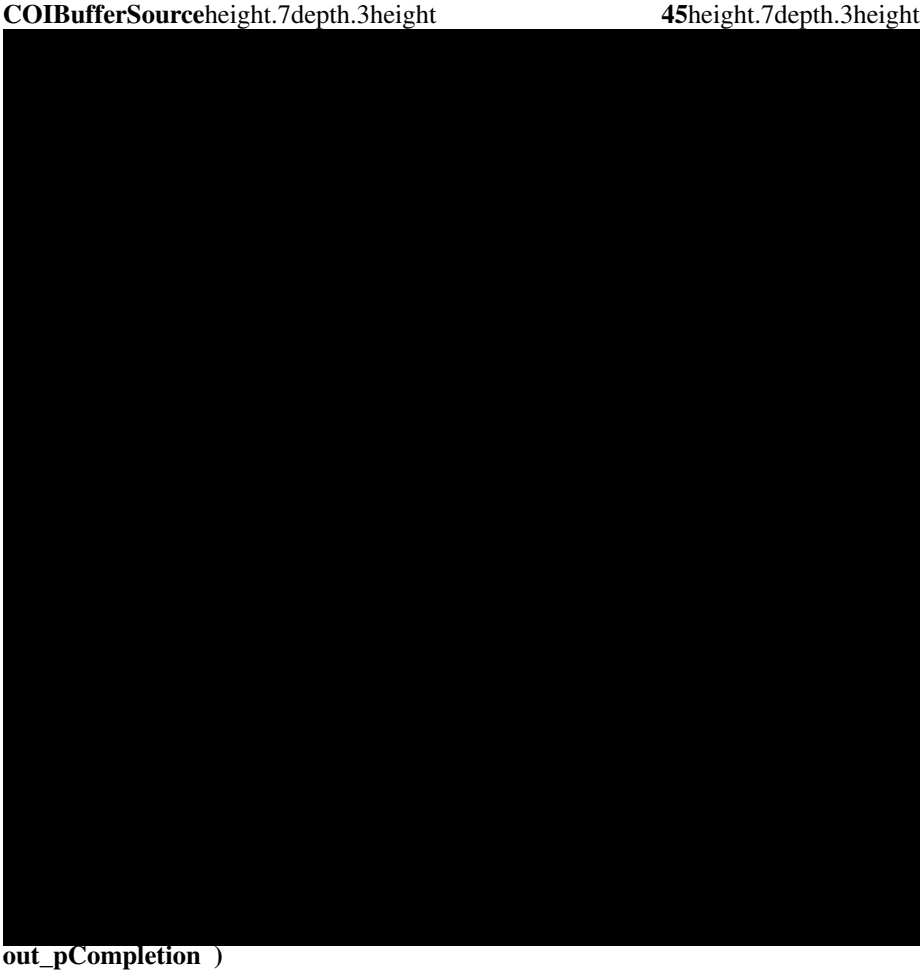
in_NumDependencies [in] The number of dependencies specified in the in_pDependencies array. This may be 0 if the caller does not want the read call to wait for any additional events to be signaled before starting the read operation.

in_pDependencies [in] An optional array of handles to previously created COIEVENT objects that this read operation will wait for before starting. This allows the user to create dependencies between buffer read calls and other operations such as run functions and map calls. The user may pass in NULL if they do not wish to wait for any additional dependencies to complete before doing the read.

out_pCompletion [out] An optional event to be signaled when the read has completed. This event can be used as a dependency to order the read with regard to future operations. If no completion event is passed in then the read is synchronous and will block until the transfer is complete.

Returns:

COI_SUCCESS if the buffer was copied successfully.
COI_INVALID_HANDLE if the buffer handle was invalid.
COI_OUT_OF_RANGE if in_Offset is beyond the end of the buffer.



out_pCompletion)

Copy data specified by multi-dimensional array data structure from an existing COIBUFFER to another multi-dimensional array located in memory.

Arrays with more than 3 dimensions are not supported. Different numbers of elements between source and destination are not supported. Note that it is not possible to use this API with any type of Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) Streaming Buffers. Please note that COIBufferReadMultiD does not follow implicit buffer dependencies. If a buffer is in use in a run function or has been added to a process using COIBufferAddRef the call to COIBufferReadMultiD will not wait, it will still copy data immediately. This is to facilitate a usage model where a buffer is being used outside of a run function, for example in a spawned thread, but data still needs to be transferred to or from the buffer.

Parameters:

in_SourceBuffer [in] Buffer to read from.

in_Offset [in] Start location of the source array within the buffer.

in_DestArray [in] A pointer to a data structure describing the structure of the data array in the buffer. Total size must not be larger than the size of in_DestBuffer. The base field of this structure will be ignored.

in_SrcArray [in] A pointer to a data structure describing the structure of the data array in local memory that should be copied. in_SrcArray and in_DestArry must have the same number of elements. The base field of this structure should be the virtual pointer to the local memory in which this array is located.

in_Type [in] The type of copy operation to use, one of either COI_COPY_UNSPECIFIED, COI_COPY_USE_DMA, COI_COPY_USE_CPU.

in_NumDependencies [in] The number of dependencies specified in the in_pDependencies array. This may be 0 if the caller does not want the write call to wait for any additional events to be signaled before starting the write operation.

in_pDependencies [in] An optional array of handles to previously created COIEVENT objects that this write operation will wait for before starting. This allows the user to create dependencies between buffer write calls and other operations such as run functions and map calls. The user may pass in NULL if they do not wish to wait for any additional dependencies to complete before doing the write.

out_pCompletion [out] An optional event to be signaled when the write has completed. This event can be used as a dependency to order the write with regard to future operations. If no completion event is passed in then the write is synchronous and will block until the transfer is complete.

Returns:

COI_SUCCESS if the buffer was copied successfully.
COI_INVALID_HANDLE if the buffer or process handle was invalid.
COI_OUT_OF_RANGE if in_Offset is beyond the end of the buffer.
COI_ARGUMENT_MISMATCH if the in_pDependencies is non NULL but in_NumDependencies is 0.
COI_ARGUMENT_MISMATCH if in_pDependencies is NULL but in_NumDependencies is not 0.
COI_NOT_SUPPORTED if the source buffer is of type COI_BUFFER_STREAMING_TO_SINK or COI_BUFFER_STREAMING_TO_SOURCE.
COI_NOT_SUPPORTED or dimension of destination or source arrays are greater than 3 or less than 1
COI_INVALID_POINTER if the pointer in_DestArray->base is NULL.
COI_OUT_OF_RANGE if in_Offset + size of in_SourceArray exceeds the size of the buffer.
COI_OUT_OF_MEMORY if any allocation of memory fails
COI_RETRY if in_SourceBuffer is mapped and is not a COI_BUFFER_PINNED buffer or COI_BUFFER_OPENCL buffer.

5.13.4.12 COIACCESSAPI COIRESULT COIBufferReleaseRefcnt (

in_Process,

in_Buffer,

in_ReleaseRefcnt)

Releases the reference count on the specified buffer and process by in_ReleaseRefcnt.

The returned result being COI_SUCCESS indicates that the specified process contains a reference to the specified buffer that has a refcnt that can be decremented. Otherwise, if the buffer or process specified do not exist, then COI_INVALID_HANDLE will be returned. If the process does not contain a reference to the specified buffer then COI_-OUT_OF_RANGE will be returned.

Parameters:

in_Process [in] The COI Process whose reference count for the specified buffer the user wants to decrement.

in_Buffer [in] The buffer used in the specified coi process in which the user wants to decrement the reference count.

in_ReleaseRefcnt [in] The value the reference count will be decremented by.

Returns:

COI_SUCCESS if the reference count was successfully decremented.
COI_INVALID_HANDLE if in_Buffer or in_Process are invalid handles.
COI_OUT_OF_RANGE if the reference for the specified buffer or process does not exist.

5.13.4.13 COIACCESSAPI COIRESULT COIBufferSetState (

in_Buffer,

in_Process,

in_State,

in_DataMove,

in_NumDependencies,

in_pDependencies,

out_pCompletion)

This API allows an experienced Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) developer to set where a COIBUFFER is located and when the COIBUFFER's data is moved.

This functionality is useful when the developer knows when and where a buffer is going to be accessed. It allows the data movement to happen sooner than if the Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) runtime tried to manage the buffer placement itself. The advantage of this API is that the developer knows much more about their own application's data access patterns and can therefore optimize the data access to be much more efficient than the Intel(R)Coprocessor Offload Infrastructure (Intel(R) COI) runtime. Using this API may yield better memory utilization, lower latency and overall improved workload throughput. This API does respect implicit dependencies for buffer read/write hazards. For example, if the buffer is being written in one COIPROCESS and the user requests the buffer be placed in another COIPROCESS then this API will wait for the first access to complete before moving the buffer. This API is not required for program correctness. It is intended solely for advanced Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) developers who wish to fine tune their application performance Cases where "a change in state" is an error condition the change just gets ignored without any error. This is because the SetState can be a non-blocking call and in such cases we can't rely on the state of the buffer at the time of the call. We can do the transition checks only at the time when the actual state change happens (which is something in future). Currently there is no way to report an error from something that happens in future and that is why such state transitions are nop. One example is using VALID_MAY_DROP with COI_SINK_OWNERS when buffer is not valid at source. This operation will be a nop if at the time of actual state change the buffer is not valid at source.

Parameters:

in_Buffer [in] The buffer to modify.

in_Process [in] The process where the state is being modified for this buffer. To modify buffer's state on source process use COI_PROCESS_SOURCE as process handle. To modify buffer's state on all processes where buffer is valid use COI_SINK_OWNERS as the process handle.

in_State [in] The new state for the buffer. The buffer's state could be set to invalid on one of the sink processes where it is being used.

in_DataMove [in] A flag to indicate if the buffer's data should be moved when the state is changed. For instance, a buffer's state may be set to valid on a process and the data move flag may be set to COI_BUFFER_MOVE which would cause the buffer contents to be copied to the process where it is now valid.

in_NumDependencies [in] The number of dependencies specified in the in_pDependencies array. This may be 0 if the caller does not want the SetState call to wait for any additional events to be signaled before starting

5.13.4.14 COIACCESSAPI COIRESULT COIBufferUnmap (

in_MapInstance,

in_NumDependencies,

in_pDependencies,

out_pCompletion)

Disables Source access to the region of the buffer that was provided through the corresponding call to COIBufferMap.

The number of calls to COIBufferUnmap() should always match the number of calls made to COIBufferMap(). The data pointer returned from the COIBufferMap() call will be invalid after this call.

Parameters:

in_MapInstance [in] buffer map instance handle to unmap.

in_NumDependencies [in] The number of dependencies specified in the in_pDependencies array. This may be 0 if the caller does not want the unmap call to wait for any events to be signaled before performing the unmap operation.

in_pDependencies [in] An optional array of handles to previously created COIEVENT objects that this unmap operation will wait for before starting. This allows the user to create dependencies between asynchronous unmap calls and other operations such as run functions or other asynchronous unmap calls. The user may pass in NULL if they do not wish to wait for any dependencies to complete before initiating unmap operations.

out_pCompletion [out] An optional pointer to a COIEVENT object that will be signaled when the unmap is complete. The user may pass in NULL if the user wants COIBufferUnmap to perform a blocking unmap operation.

Returns:

COI_SUCCESS upon successful unmapping of the buffer instance.
COI_INVALID_HANDLE if the passed in map instance handle was NULL.
COI_ARGUMENT_MISMATCH if the in_pDependencies is non NULL but in_NumDependencies is 0.
COI_ARGUMENT_MISMATCH if in_pDependencies is NULL but in_NumDependencies is not 0.

5.13.4.15 COIACCESSAPI COIRESULT COIBufferWrite (

in_DestBuffer,

in_Offset,

in_pSourceData,

in_Length,

in_Type,

in_NumDependencies,

in_pDependencies,

out_pCompletion)

Copy data from a normal virtual address into an existing COIBUFFER.

Note that it is not possible to use this API with any type of Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) Streaming Buffers. Please note that COIBufferWrite does not follow implicit buffer dependencies. If a buffer is in use in a run function or has been added to a process using COIBufferAddRef the call to COIBufferWrite will not

wait, it will still copy data immediately. This is to facilitate a usage model where a buffer is being used outside of a run function, for example in a spawned thread, but data still needs to be transferred to or from the buffer.

Parameters:

in_DestBuffer [in] Buffer to write into.

in_Offset [in] Location in the buffer to start writing to.

in_pSourceData [in] A pointer to local memory that should be copied into the provided buffer.

in_Length [in] The number of bytes to write from in_pSourceData into in_DestBuffer. Must not be larger than the size of in_DestBuffer and must not over run in_DestBuffer if an in_Offset is provided.

in_Type [in] The type of copy operation to use, one of either COI_COPY_UNSPECIFIED, COI_COPY_USE_DMA, COI_COPY_USE_CPU.

in_NumDependencies [in] The number of dependencies specified in the in_pDependencies array. This may be 0 if the caller does not want the write call to wait for any additional events to be signaled before starting the write operation.

in_pDependencies [in] An optional array of handles to previously created COIEVENT objects that this write operation will wait for before starting. This allows the user to create dependencies between buffer write calls and other operations such as run functions and map calls. The user may pass in NULL if they do not wish to wait for any additional dependencies to complete before doing the write.

out_pCompletion [out] An optional event to be signaled when the write has completed. This event can be used as a dependency to order the write with regard to future operations. If no completion event is passed in then the write is synchronous and will block until the transfer is complete.

Returns:

COI_SUCCESS if the buffer was copied successfully.
COI_INVALID_HANDLE if the buffer handle was invalid.
COI_OUT_OF_RANGE if in_Offset is beyond the end of the buffer.
COI_ARGUMENT_MISMATCH if the in_pDependencies is non NULL but in_NumDependencies is 0.
COI_ARGUMENT_MISMATCH if in_pDependencies is NULL but in_NumDependencies is not 0.
COI_NOT_SUPPORTED if the source buffer is of type COI_BUFFER_STREAMING_TO_SINK or COI_BUFFER_STREAMING_TO_SOURCE.
COI_INVALID_POINTER if the in_pSourceData pointer is NULL.
COI_OUT_OF_RANGE if in_Offset + in_Length exceeds the size of the buffer.
COI_OUT_OF_RANGE if in_Length is 0.

not follow implicit buffer dependencies. If a buffer is in use in a run function or has been added to a process using COIBufferAddRef the call to COIBufferWrite will not wait, it will still copy data immediately. This is to facilitate a usage model where a buffer is being used outside of a run function, for example in a spawned thread, but data still needs to be transferred to or from the buffer.

Parameters:

in_DestBuffer [in] Buffer to write into.

in_DestProcess [in] A pointer to the process to which the data will be written. Buffer is updated only in this process and invalidated in other processes. Only a single process can be specified. Can be left NULL and default behavior will be chosen, which chooses the first valid process in which regions are found. Other buffer regions are invalidated if not updated.

in_Offset [in] Location in the buffer to start writing to.

in_pSourceData [in] A pointer to local memory that should be copied into the provided buffer.

in_Length [in] The number of bytes to write from in_pSourceData into in_DestBuffer. Must not be larger than the size of in_DestBuffer and must

not over run in_DestBuffer if an in_Offset is provided.

in_Type [in] The type of copy operation to use, one of either COI_COPY_-UNSPECIFIED, COI_COPY_USE_DMA, COI_COPY_USE_CPU.

in_NumDependencies [in] The number of dependencies specified in the in_pDependencies array. This may be 0 if the caller does not want the write call to wait for any additional events to be signaled before starting the write operation.

in_pDependencies [in] An optional array of handles to previously created COIEVENT objects that this write operation will wait for before starting. This allows the user to create dependencies between buffer write calls and other operations such as run functions and map calls. The user may pass in NULL if they do not wish to wait for any additional dependencies to complete before doing the write.

out_pCompletion [out] An optional event to be signaled when the write has completed. This event can be used as a dependency to order the write with regard to future operations. If no completion event is passed in then the write is synchronous and will block until the transfer is complete.

Returns:

COI_SUCCESS if the buffer was copied successfully.
COI_INVALID_HANDLE if the buffer handle was invalid.
COI_OUT_OF_RANGE if in_Offset is beyond the end of the buffer.
COI_ARGUMENT_MISMATCH if the in_pDependencies is non NULL but in_-NumDependencies is 0.
COI_ARGUMENT_MISMATCH if in_pDependencies is NULL but in_-NumDependencies is not 0.
COI_NOT_SUPPORTED if the source buffer is of type COI_BUFFER_-STREAMING_TO_SINK or COI_BUFFER_STREAMING_TO_SOURCE.
COI_INVALID_POINTER if the in_pSourceData pointer is NULL.
COI_OUT_OF_RANGE if in_Offset + in_Length exceeds the size of the buffer.
COI_OUT_OF_RANGE if in_Length is 0.
COI_RETRY if in_DestBuffer is mapped and is not a COI_BUFFER_PINNED buffer or COI_BUFFER_OPENCL buffer.

5.13.4.17 COIACCESSAPI COIRESULT COIBufferWriteMultiD (

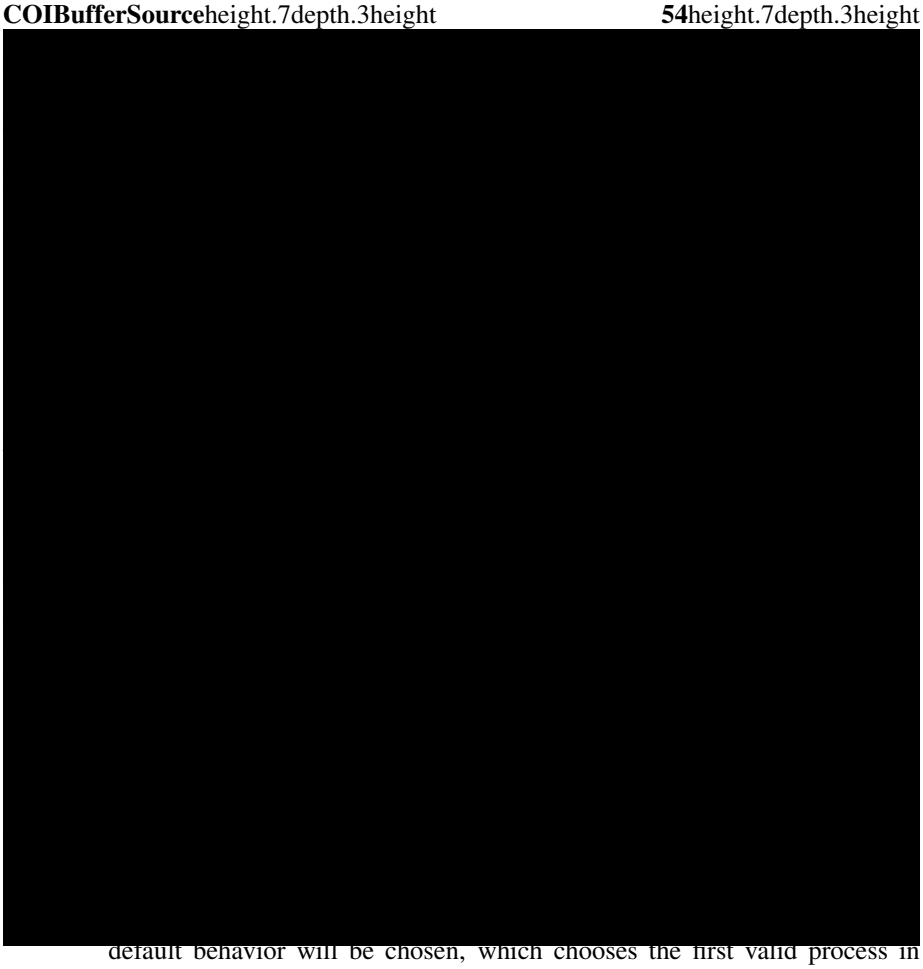
in_DestBuffer,

in_DestProcess,

in_Offset,

in_DestArray,

in_SrcArray,



default behavior will be chosen, which chooses the first valid process in which regions are found. Other buffer regions are invalidated if not updated.

in_Offset [in] Start location of the destination array within the buffer.

in_DestArray [in] A pointer to a data structure describing the structure of the data array in the buffer. Total size must not be larger than the size of in_DestBuffer. The base field of this structure will be ignored.

in_SrcArray [in] A pointer to a data structure describing the structure of the data array in local memory that should be copied. in_SrcArray and in_DestArry must have the same number of elements. The base field of this structure should be the virtual pointer to the local memory in which this array is located.

in_Type [in] The type of copy operation to use, one of either COI_COPY_-UNSPECIFIED, COI_COPY_USE_DMA, COI_COPY_USE_CPU.

in_NumDependencies [in] The number of dependencies specified in the in_pDependencies array. This may be 0 if the caller does not want the write call to wait for any additional events to be signaled before starting the write operation.

in_pDependencies [in] An optional array of handles to previously created COIEVENT objects that this write operation will wait for before starting. This allows the user to create dependencies between buffer write calls and other operations such as run functions and map calls. The user may pass in NULL if they do not wish to wait for any additional dependencies to complete before doing the write.

out_pCompletion [out] An optional event to be signaled when the write has completed. This event can be used as a dependency to order the write with regard to future operations. If no completion event is passed in then the write is synchronous and will block until the transfer is complete.

Returns:

COI_SUCCESS if the buffer was copied successfully.
COI_INVALID_HANDLE if the buffer or process handle was invalid.
COI_OUT_OF_RANGE if in_Offset is beyond the end of the buffer.
COI_ARGUMENT_MISMATCH if the in_pDependencies is non NULL but in_NumDependencies is 0.
COI_ARGUMENT_MISMATCH if in_pDependencies is NULL but in_NumDependencies is not 0.
COI_NOT_SUPPORTED if the destination buffer is of type COI_BUFFER_STREAMING_TO_SINK or COI_BUFFER_STREAMING_TO_SOURCE.
COI_NOT_SUPPORTED or dimension of destination or source arrays are greater than 3 or less than 1
COI_INVALID_POINTER if the pointer in_SrcArray->base is NULL.
COI_OUT_OF_RANGE if in_Offset + size of in_DestArray exceeds the size of the buffer.
COI_OUT_OF_MEMORY if any allocation of memory fails
COI_RETRY if in_DestBuffer is mapped and is not a COI_BUFFER_PINNED buffer or COI_BUFFER_OPENCL buffer.

5.14 COIEngineSource

Data Structures

- struct COI_ENGINE_INFO

  This structure returns information about an Intel(R) Xeon Phi(TM) coprocessor.

Defines

- #define COI_MAX_DRIVER_VERSION_STR_LEN 255
- #define COI_MAX_HW_THREADS 1024

Typedefs

- typedef struct COI_ENGINE_INFO COI_ENGINE_INFO

This structure returns information about an Intel(R) Xeon Phi(TM) coprocessor.

Enumerations

- enum coi_eng_misc {
  COI_ENG_ECC_DISABLED = 0,
  COI_ENG_ECC_ENABLED = 0x00000001,
  COI_ENG_ECC_UNKNOWN = 0x00000002 }

  This enum defines miscellaneous information returned from the COIGetEngineInfo() function.

Functions

- COIACCESSAPI COIRESULT COIEngineGetCount (COI_ISA_TYPE in_ISA, uint32_t *out_pNumEngines)

  Returns the number of engines in the system that match the provided ISA.

- COIACCESSAPI COIRESULT COIEngineGetHandle (COI_ISA_TYPE in_ISA, uint32_t in_EngineIndex, COIENGINE *out_pEngineHandle)

  Returns the handle of a user specified engine.

- COIACCESSAPI COIRESULT COIEngineGetInfo (COIENGINE in_EngineHandle, uint32_t in_EngineInfoSize, COI_ENGINE_INFO *out_pEngineInfo)

  Returns information related to a specified engine.

5.14.1 Define Documentation

5.14.1.1 #define COI_MAX_DRIVER_VERSION_STR_LEN 255

Definition at line 56 of file COIEngine_source.h.

5.14.1.2 #define COI_MAX_HW_THREADS 1024

Definition at line 58 of file COIEngine_source.h.

5.14.4.1 COIACCESSAPI COIRESULT COIEngineGetCount (

in_ISA,

out_pNumEngines)

Returns the number of engines in the system that match the provided ISA.

Note that while it is possible to enumerate different types of Intel(R) Xeon Phi(TM) coprocessors on a single host this is not currently supported. Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) makes an assumption that all Intel(R) Xeon Phi(TM) coprocessors found in the system are the same architecture as the first coprocessor device.

Also, note that this function returns the number of engines that Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) is able to detect. Not all of them may be online.

Parameters:

in_ISA [in] Specifies the ISA type of the engine requested.

out_pNumEngines [out] The number of engines available. This can be used to index into the engines using COIEngineGetHandle().

Returns:

COI_SUCCESS if the function completed without error.
COI_DOES_NOT_EXIST if the in_ISA parameter is not valid.
COI_INVALID_POINTER if the out_pNumEngines parameter is NULL.

5.14.4.2 COIACCESSAPI COIRESULT COIEngineGetHandle (

in_ISA,

in_EngineIndex,

out_pEngineHandle)

Returns the handle of a user specified engine.

Parameters:

in_ISA [in] Specifies the ISA type of the engine requested.

in_EngineIndex [in] A unsigned integer which specifies the zero-based position of the engine in a collection of engines. The makeup of this collection is defined by the in_ISA parameter.

out_pEngineHandle [out] The address of an COIENGINE handle.

Returns:

COI_SUCCESS if the function completed without error.
COI_DOES_NOT_EXIST if the in_ISA parameter is not valid.
COI_OUT_OF_RANGE if in_EngineIndex is greater than or equal to the number of engines that match the in_ISA parameter.
COI_INVALID_POINTER if the out_pEngineHandle parameter is NULL.
COI_VERSION_MISMATCH if the version of Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) on the host is not compatible with the version on the device.
COI_NOT_INITIALIZED if the engine requested exists but is offline.

5.14.4.3 COIACCESSAPI COIRESULT COIEngineGetInfo (

in_EngineHandle,

in_EngineInfoSize,

out_pEngineInfo)

Returns information related to a specified engine.

Note that if Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) is unable to query unable to query a value it will be returned as zero but the call will still succeed.

Parameters:

in_EngineHandle [in] The COIENGINE structure as provided from COIEngineGetHandle() which to query for device level information.

in_EngineInfoSize [in] The size of the structure that out_pEngineInfo points to. Used for version safety of the function call.

out_pEngineInfo [out] The address of a user allocated COI_ENGINE_INFO structure. Upon success, the contents of the structure will be updated to contain information related to the specified engine.

Returns:

COI_SUCCESS if the function completed without error.
COI_INVALID_HANDLE if the in_EngineHandle handle is not valid.
COI_SIZE_MISMATCH if in_EngineInfoSize does not match any current or previous COI_ENGINE_INFO structure sizes.
COI_INVALID_POINTER if the out_pEngineInfo pointer is NULL.

5.15 COIPipelineSource

Files

- file COIPipeline_source.h

Defines

- #define COI_PIPELINE_MAX_IN_BUFFERS 16384
- #define COI_PIPELINE_MAX_IN_MISC_DATA_LEN 32768
- #define COI_PIPELINE_MAX_PIPELINES 512

Typedefs

- typedef enum COI_ACCESS_FLAGS COI_ACCESS_FLAGS

  These flags specify how a buffer will be used within a run function.

Enumerations

- enum COI_ACCESS_FLAGS {
  COI_SINK_READ = 1,
  COI_SINK_WRITE,
  COI_SINK_WRITE_ENTIRE,
  COI_SINK_READ_ADDREF,
  COI_SINK_WRITE_ADDREF,
  COI_SINK_WRITE_ENTIRE_ADDREF }

  These flags specify how a buffer will be used within a run function.

Functions

- COIACCESSAPI COIRESULT COIPipelineClearCPUMask (COI_CPU_MASK *in_Mask)

  Clears a given mask.

- COIACCESSAPI COIRESULT COIPipelineCreate (COIPROCESS in_Process, COI_CPU_MASK in_Mask, uint32_t in_StackSize, COIPIPELINE *out_pPipeline)

  Create a pipeline assoiated with a remote process.

- COIACCESSAPI COIRESULT COIPipelineDestroy (COIPIPELINE in_Pipeline)

  Destroys the inidicated pipeline, releasing its resources.

- COIACCESSAPI COIRESULT COIPipelineGetEngine (COIPIPELINE in_Pipeline, COIENGINE *out_pEngine)

  Retrieve the engine that the pipeline is associated with.

- COIACCESSAPI COIRESULT COIPipelineRunFunction (COIPIPELINE in_Pipeline, COIFUNCTION in_Function, uint32_t in_NumBuffers, const COIBUFFER *in_pBuffers, const COI_ACCESS_FLAGS *in_pBufferAccessFlags, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, const void *in_pMiscData, uint16_t in_MiscDataLen, void *out_pAsyncReturnValue, uint16_t in_AsyncReturnValueLen, COIEVENT *out_pCompletion)

  Enqueues a function in the remote process binary to be executed.

- COIACCESSAPI COIRESULT COIPipelineSetCPUMask (COIPROCESS in_Process, uint32_t in_CoreID, uint8_t in_ThreadID, COI_CPU_MASK *out_pMask)

  Add a particular core:thread pair to a COI_CPU_MASK.

5.15.1 Define Documentation

5.15.1.1 #define COI_PIPELINE_MAX_IN_BUFFERS 16384

Definition at line 95 of file COIPipeline_source.h.

5.15.1.2 #define COI_PIPELINE_MAX_IN_MISC_DATA_LEN 32768

Definition at line 96 of file COIPipeline_source.h.

5.15.1.3 #define COI_PIPELINE_MAX_PIPELINES 512

Definition at line 94 of file COIPipeline_source.h.

5.15.2 Typedef Documentation

5.15.2.1 typedef enum COI_ACCESS_FLAGS COI_ACCESS_FLAGS

These flags specify how a buffer will be used within a run function.

They allow the runtime to make optimizations in how it moves the data around. These flags can affect the correctness of an application, so they must be set properly. For example, if a buffer is used in a run function with the COI_SINK_READ flag and then mapped on the source, the runtime may use a previously cached version of the buffer instead of retrieving data from the sink.

5.15.3 Enumeration Type Documentation

5.15.3.1 enum COI_ACCESS_FLAGS

These flags specify how a buffer will be used within a run function.

They allow the runtime to make optimizations in how it moves the data around. These flags can affect the correctness of an application, so they must be set properly. For example, if a buffer is used in a run function with the COI_SINK_READ flag and then mapped on the source, the runtime may use a previously cached version of the buffer instead of retrieving data from the sink.

Enumerator:

COI_SINK_READ Specifies that the run function will only read the associated buffer.

COI_SINK_WRITE Specifies that the run function will write to the associated buffer.

COI_SINK_WRITE_ENTIRE Specifies that the run function will overwrite the entire associated buffer and therefore the buffer will not be synchronized with the source before execution.

COI_SINK_READ_ADDREF Specifies that the run function will only read the associated buffer and will maintain the reference count on the buffer after run function exit.

COI_SINK_WRITE_ADDREF Specifies that the run function will write to the associated buffer and will maintain the reference count on the buffer after run function exit.

COI_SINK_WRITE_ENTIRE_ADDREF Specifies that the run function will overwrite the entire associated buffer and therefore the buffer will not be synchronized with the source before execution and will maintain the reference count on the buffer after run function exit.

Definition at line 64 of file COIPipeline_source.h.

5.15.4 Function Documentation

5.15.4.1 COIACCESSAPI COIRESULT COIPipelineClearCPUMask (

in_Mask)

Clears a given mask.

Note that the memory contents of COI_CPU_MASK are not guaranteed to be zero when declaring a COI_CPU_MASK variable. Thus, prior to setting a specific affinity to in_Mask it is important to call this function first.

Parameters:

in_Mask [in] Pointer to the mask to clear.

Returns:

COI_SUCCESS if the mask was cleared.
COI_INVALID_POINTER if in_Mask is invalid.

5.15.4.2 COIACCESSAPI COIRESULT COIPipelineCreate (

in_Process,

in_Mask,

in_StackSize,

out_pPipeline)

Create a pipeline assoiated with a remote process.

This pipeline can then be used to execute remote functions and to share data using COIBuffers.

Parameters:

in_Process [in] A handle to an already existing process that the pipeline will be associated with.

in_Mask [in] An optional mask of the set of hardware threads on which the sink pipeline command processing thread could run.

in_StackSize [in] An optional value that will be used when the pipeline processing thread is created on the sink. If the user passes in 0 the OS default stack size will be used. Otherwise the value must be PTHREAD_STACK_-MIN (16384) bytes or larger and must be a multiple of a page (4096 bytes).

out_pPipeline [out] Handle returned to uniquely identify the pipeline that was created for use in later API calls.

Returns:

COI_SUCCESS if the pipeline was successfully created.
COI_INVALID_HANDLE if the in_Process handle passed in was invalid.
COI_INVALID_POINTER if the out_pPipeline pointer was NULL.
COI_RESOURCE_EXHAUSTED if no more COIPipelines can be created. The maximum number of pipelines allowed is COI_PIPELINE_MAX_PIPELINES. It is recommended in most cases to not exceed the number of CPU's that are reported on the offload device, performance will suffer.
COI_OUT_OF_RANGE if the in_StackSize $>$ 0 && in_StackSize $<$ PTHREAD_STACK_MIN or if in_StackSize is not a multiple of a page (4096 bytes).
COI_OUT_OF_RANGE if the in_Mask is set to all zeroes. If no mask is desired then the in_Mask should be passed as NULL, otherwise at least one thread must be set.
COI_TIME_OUT_REACHED if establishing the communication channel with the remote pipeline timed out.
COI_RETRY if the pipeline cannot be created due to the number of source-to-sink connections in use. A subsequent call to COIPipelineCreate may succeed if resources are freed up.
COI_PROCESS_DIED if in_Process died.

5.15.4.3 COIACCESSAPI COIRESULT COIPipelineDestroy (

in_Pipeline)

Destroys the inidicated pipeline, releasing its resources.

Parameters:

in_Pipeline [in] Pipeline to destroy.

Returns:

COI_SUCCESS if the pipeline was destroyed

5.15.4.4 COIACCESSAPI COIRESULT COIPipelineGetEngine (

in_Pipeline,

out_pEngine)

Retrieve the engine that the pipeline is associated with.

Parameters:

in_Pipeline [in] Pipeline to query.

---

out_pEngine [out] The handle of the Engine.

Returns:

COI_SUCCESS if the engine was retrieved.
COI_INVALID_HANDLE if the pipeline handle passed in was invalid.
COI_INVALID_POINTER if the out_pEngine parameter is NULL.
COI_PROCESS_DIED if the process associated with this engine died.

5.15.4.5 COIACCESSAPI COIRESULT COIPipelineRunFunction (

in_Pipeline,

in_Function,

in_NumBuffers,

in_pBuffers,

in_pBufferAccessFlags,

the buffer gets destroyed before the runtime receives the completion notification of the Runfunction, it can cause unexpected behaviour. So it is always recommended to wait for RunFunction completion event before any related destroy event occurs.

The runtime expects users to handle such scenarios. COIPipelineRunFunction returns COI_SUCCESS for above cases because it was queued up successfully. Also if you try to destroy a pipeline with a stalled function then the destroy call will hang. COIPipelineDestroy waits until all the functions enqueued are finished executing.

Parameters:

in_Pipeline [in] Handle to a previously created pipeline that this run function should be enqueued to.

in_Function [in] Previously returned handle from a call to COIPipelineGetFunctionHandle() that represents a function in the application running on the Sink process.

in_NumBuffers [in] The number of buffers that are being passed to the run function. This number must match the number of buffers in the in_pBuffers and in_pBufferAccessFlags arrays. Must be less than COI_PIPELINE_MAX_IN_BUFFERS.

in_pBuffers [in] An array of COIBUFFER handles that the function is expected to use during its execution. Each buffer when it arrives at the Sink process will be at least 4k page aligned, thus, using a very large number of small buffers is memory inefficient and should be avoided.

in_pBufferAccessFlags [in] An array of flag values which correspond to the buffers passed in the in_pBuffers parameter. These flags are used to track dependencies between different run functions being executed from different pipelines.

in_NumDependencies [in] The number of dependencies specified in the in_pDependencies array. This may be 0 if the caller does not want the run function to wait for any dependencies.

in_pDependencies [in] An optional array of COIEVENT objects that this run function will wait for before executing. This allows the user to create dependencies between run functions in different pipelines. The user may pass in NULL if they do not wish to wait for any dependencies to complete.

in_pMiscData [in] Pointer to user defined data, typically used to pass parameters to Sink side functions. Should only be used for small amounts data since the data will be placed directly in the Driver's command buffer. COIBuffers should be used to pass large amounts of data.

in_MiscDataLen [in] Size of the in_pMiscData in bytes. Must be less than COI_PIPELINE_MAX_IN_MISC_DATA_LEN, and should usually be much smaller, see documentation for the parameter in_pMiscData.

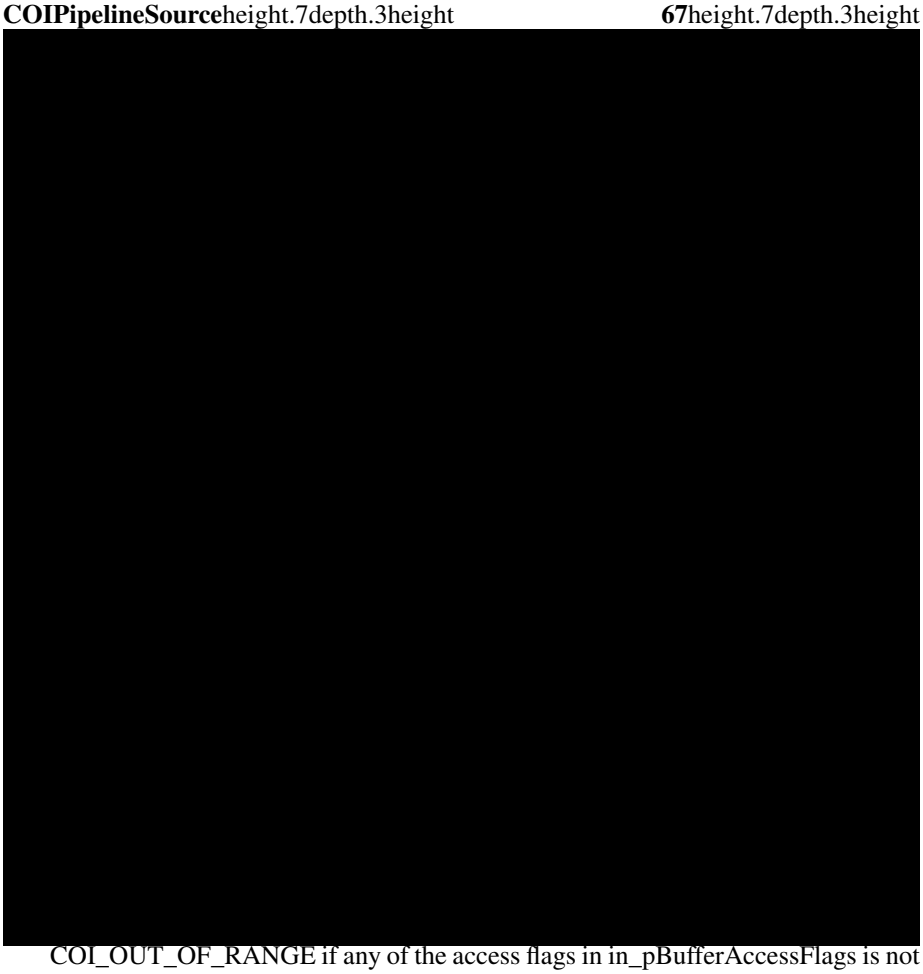
out_pAsyncReturnValue [out] Pointer to user-allocated memory where the return value from the run function will be placed. This memory should not be read until out_pCompletion has been signalled.

in_AsyncReturnValueLen [in] Size of the out_pAsyncReturnValue in bytes.

out_pCompletion [out] An optional pointer to a COIEVENT object that will be signaled when this run function has completed execution. The user may pass in NULL if they wish for this function to be synchronous, otherwise if a COIEVENT object is passed in the function is then asynchronous and closes after enqueuing the RunFunction and passes back the COIEVENT that will be signaled once the RunFunction has completed.

Returns:

COI_SUCCESS if the function was successfully placed in a pipeline for future execution. Note that the actual execution of the function will occur in the future.
COI_OUT_OF_RANGE if in_NumBuffers is greater than COI_PIPELINE_-MAX_IN_BUFFERS or if in_MiscDataLen is greater than COI_PIPELINE_-MAX_IN_MISC_DATA_LEN.
COI_INVALID_HANDLE if the pipeline handle passed in was invalid.
COI_INVALID_HANDLE if the function handle passed in was invalid.
COI_INVALID_HANDLE if any of the buffers passed in are invalid.



COI_OUT_OF_RANGE if any of the access flags in in_pBufferAccessFlags is not a valid COI_ACCESS_FLAGS.

5.15.4.6 COIACCESSAPI COIRESULT COIPipelineSetCPUMask (

in_Process,

in_CoreID,

in_ThreadID,

out_pMask)

Add a particular core:thread pair to a COI_CPU_MASK.

Parameters:

in_Process [in] A handle to an already existing process that the pipeline will be associated with.

in_CoreID [in] Core to affinitize to; must be less than the number of cores on the device.

in_ThreadID [in] Thread on the core to affinitize to (0 - 3).

out_pMask [out] Pointer to the mask to set.

Warning:

Unless it is explicitly done, the contents of the mask may not be zero when creating or declaring a COI_CPU_MASK variable.

Returns:

COI_SUCCESS if the mask was set.
COI_OUT_OF_RANGE if the in_CoreID or in_ThreadID is out of range.
COI_INVALID_POINTER if out_pMask is invalid.
COI_INVALID_HANDLE if in_Process is invalid.

5.16 COIProcessSource

Files

- file COIProcess_source.h

Defines

- #define COI_FAT_BINARY ((uint64_t)-1)

  This is a flag for COIProcessCreateFromMemory that indicates the passed in memory pointer is a fat binary file and should not have regular validation.

- #define COI_MAX_FILE_NAME_LENGTH 256
- #define COI_MAX_FUNCTION_NAME_LENGTH 256
- #define COI_PROCESS_SOURCE ((COIPROCESS)-1)

  This is a special COIPROCESS handle that can be used to indicate that the source process should be used for an operation.

Typedefs

- typedef void(∗ COI_NOTIFICATION_CALLBACK)(COI_NOTIFICATIONS in_Type, COIPROCESS in_Process, COIEVENT in_Event, const void ∗in_UserData)

  A callback that will be invoked to notify the user of an internal Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) event.

- typedef enum COI_NOTIFICATIONS COI_NOTIFICATIONS

The user can choose to have notifications for these internal events so that they can build their own profiling and performance layer on top of Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI).

Enumerations

- enum COI_NOTIFICATIONS {
  RUN_FUNCTION_READY = 0,
  RUN_FUNCTION_START,
  RUN_FUNCTION_COMPLETE,
  BUFFER_OPERATION_READY,
  BUFFER_OPERATION_COMPLETE,
  USER_EVENT_SIGNALED }

  The user can choose to have notifications for these internal events so that they can build their own profiling and performance layer on top of Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI).

Functions

- __asm__ (".symver COIProcessLoadLibraryFromMemory,""COIProcessLoadLibraryFromMemory@COI_1.0")
- __asm__ (".symver COIProcessLoadLibraryFromFile,""COIProcessLoadLibraryFromFile@COI_1.0")
- COIACCESSAPI void COINotificationCallbackSetContext (const void *in_UserData)

  Set the user data that will be returned in the notification callback.

- COIACCESSAPI COIRESULT COIProcessCreateFromFile (COIENGINE in_Engine, const char *in_pBinaryName, int in_Argc, const char **in_ppArgv, uint8_t in_DupEnv, const char **in_ppAdditionalEnv, uint8_t in_ProxyActive, const char *in_Reserved, uint64_t in_InitialBufferSpace, const char *in_LibrarySearchPath, COIPROCESS *out_pProcess)

  Create a remote process on the Sink and start executing its main() function.

- COIACCESSAPI COIRESULT COIProcessCreateFromMemory (COIENGINE in_Engine, const char *in_pBinaryName, const void *in_pBinaryBuffer, uint64_t in_BinaryBufferLength, int in_Argc, const char **in_ppArgv, uint8_t in_DupEnv, const char **in_ppAdditionalEnv, uint8_t in_ProxyActive, const char *in_Reserved, uint64_t in_InitialBufferSpace, const char *in_LibrarySearchPath, const char *in_FileOfOrigin, uint64_t in_FileOfOriginOffset, COIPROCESS *out_pProcess)

  Create a remote process on the Sink and start executing its main() function.

- COIACCESSAPI COIRESULT COIProcessDestroy (COIPROCESS in_Process, int32_t in_WaitForMainTimeout, uint8_t in_ForceDestroy, int8_t *out_pProcessReturn, uint32_t *out_pTerminationCode)

  Destroys the indicated process, releasing its resources.

- COIACCESSAPI COIRESULT COIProcessGetFunctionHandles (COIPROCESS in_Process, uint32_t in_NumFunctions, const char **in_ppFunctionNameArray, COIFUNCTION *out_pFunctionHandleArray)

  Given a loaded native process, gets an array of function handles that can be used to schedule run functions on a pipeline associated with that process.

- COIRESULT COIProcessLoadLibraryFromFile (COIPROCESS in_Process, const char *in_pFileName, const char *in_pLibraryName, const char *in_LibrarySearchPath, COILIBRARY *out_pLibrary)

  Loads a shared library into the specified remote process, akin to using dlopen() on a local process in Linux or LoadLibrary() in Windows.

- COIRESULT COIProcessLoadLibraryFromMemory (COIPROCESS in_Process, const void *in_pLibraryBuffer, uint64_t in_LibraryBufferLength, const char *in_pLibraryName, const char *in_LibrarySearchPath, const char *in_FileOfOrigin, uint64_t in_FileOfOriginOffset, COILIBRARY *out_pLibrary)

  Loads a shared library into the specified remote process, akin to using dlopen() on a local process in Linux or LoadLibrary() in Windows.

- COIACCESSAPI COIRESULT COIProcessRegisterLibraries (uint32_t in_NumLibraries, const void **in_ppLibraryArray, const uint64_t *in_pLibrarySizeArray, const char **in_ppFileOfOriginArray, const uint64_t *in_pFileOfOriginOffSetArray)

  Registers shared libraries that are already in the host process's memory to be used during the shared library dependency resolution steps that take place during subsequent calls to COIProcessCreate and COIProcessLoadLibrary*.*

- COIACCESSAPI COIRESULT COIProcessSetCacheSize (const COIPROCESS in_Process, const uint64_t in_HugePagePoolSize, const uint32_t in_HugeFlags, const uint64_t in_SmallPagePoolSize, const uint32_t in_SmallFlags, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Set the minimum preferred COIProcess cache size.

- COIACCESSAPI COIRESULT COIProcessUnloadLibrary (COIPROCESS in_Process, COILIBRARY in_Library)

  Unloads a a previously loaded shared library from the specified remote process.

- COIACCESSAPI COIRESULT COIRegisterNotificationCallback (COIPROCESS in_Process, COI_NOTIFICATION_CALLBACK in_Callback, const void *in_UserData)

  Register a callback to be invoked to notify that an internal Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) event has occured on the process that is associated with the callback.

- COIACCESSAPI COIRESULT COIUnregisterNotificationCallback (COIPROCESS in_Process, COI_NOTIFICATION_CALLBACK in_Callback)

  Unregisters a callback, notifications will no longer be signaled.

COIProcessSetCacheSize flags.

Flags are divided into two categories: _MODE_ and _ACTION_ only one of each is valid with each call.

ACTIONS and _MODES_ should be bitwised OR'ed together, i.e. |

- #define COI_CACHE_MODE_MASK 0x00000007

  Current set of DEFINED bits for _MODE_, can be used to clear or check fields, not useful to pass into APIs.

- #define COI_CACHE_MODE_NOCHANGE 0x00000001

  Flag to indicate to keep the previous mode of operation.

- #define COI_CACHE_MODE_ONDEMAND_SYNC 0x00000002

  Mode of operation that indicates that COI will allocate physical cache memory exactly when it is is needed.

- #define COI_CACHE_MODE_ONDEMAND_ASYNC 0x00000004

  Not yet implemented.

- #define COI_CACHE_ACTION_MASK 0x00070000

  Current set of DEFINED bits for _ACTION_ can be used to clear fields, but not usefil to pass into API's.

- #define COI_CACHE_ACTION_NONE 0x00010000

  No action requested.

- #define COI_CACHE_ACTION_GROW_NOW 0x00020000

  This _ACTION_ flag will immediately attempt to increase the cache physical memory size to the current set pool size(s).

- #define COI_CACHE_ACTION_FREE_UNUSED 0x00040000

  Not yet implemented.

5.16.1 Define Documentation

5.16.1.1 #define COI_CACHE_ACTION_FREE_UNUSED 0x00040000

Not yet implemented.

Future _ACTION_ that will attempt to find unused allocated cache and free it, with the express goal of reducing the footprint on the remote process down to the value of the currently set pool size(s).

Definition at line 1017 of file COIProcess_source.h.

5.16.1.2 #define COI_CACHE_ACTION_GROW_NOW 0x00020000

This _ACTION_ flag will immediately attempt to increase the cache physical memory size to the current set pool size(s).

Used to pre-allocate memory on remote processes, so that runfunction will enqueue faster. Also may prevent unused buffer eviction from process reducing overhead in trade for memory allocation cost.

Definition at line 1011 of file COIProcess_source.h.

5.16.1.3 #define COI_CACHE_ACTION_MASK 0x00070000

Current set of DEFINED bits for _ACTION_ can be used to clear fields, but not usefil to pass into API's.

Used internaly.

Definition at line 998 of file COIProcess_source.h.

5.16.1.4 #define COI_CACHE_ACTION_NONE 0x00010000

No action requested.

With this flag specified it is recommended to NOT provide a out_pCompletion event, as with this flag, modes and values are immediately set. This is valid with _MODE_ flags.

Definition at line 1004 of file COIProcess_source.h.

5.16.1.5 #define COI_CACHE_MODE_MASK 0x00000007

Current set of DEFINED bits for _MODE_, can be used to clear or check fields, not useful to pass into APIs.

Used internally.

Definition at line 975 of file COIProcess_source.h.

5.16.1.6 #define COI_CACHE_MODE_NOCHANGE 0x00000001

Flag to indicate to keep the previous mode of operation.

By default this would be COI_CACHE_MODE_ONDEMAND_SYNC. As of this release This is the only mode available. This mode is valid with _ACTION_ flags.

Definition at line 981 of file COIProcess_source.h.

5.16.1.7 #define COI_CACHE_MODE_ONDEMAND_ASYNC 0x00000004

Not yet implemented.

Future mode that will not stall a COIPipeline but prefer eviction/paging if possible as to immediately execute pipeline. At the same time, enqueue background requests to allocate extra cache so as to provide optimze behavior on subsequent runs.

Definition at line 993 of file COIProcess_source.h.

5.16.1.8 #define COI_CACHE_MODE_ONDEMAND_SYNC 0x00000002

Mode of operation that indicates that COI will allocate physical cache memory exactly when it is is needed.

COIPipeline execution in the given process will momentarily block until the allocation request is completed. This is and has been the default mode.

Definition at line 987 of file COIProcess_source.h.

5.16.1.9 #define COI_FAT_BINARY ((uint64_t)-1)

This is a flag for COIProcessCreateFromMemory that indicates the passed in memory pointer is a fat binary file and should not have regular validation.

Definition at line 67 of file COIProcess_source.h.

5.16.1.10 #define COI_MAX_FILE_NAME_LENGTH 256

Definition at line 61 of file COIProcess_source.h.

5.16.1.11 #define COI_MAX_FUNCTION_NAME_LENGTH 256

Definition at line 408 of file COIProcess_source.h.

5.16.1.12 #define COI_PROCESS_SOURCE ((COIPROCESS)-1)

This is a special COIPROCESS handle that can be used to indicate that the source process should be used for an operation.

Definition at line 59 of file COIProcess_source.h.

5.16.2 Typedef Documentation

5.16.2.1 typedef void(∗ COI_NOTIFICATION_CALLBACK)(COI_NOTIFICATIONS in_Type, COIPROCESS in_Process, COIEVENT in_Event, const void ∗in_UserData)

A callback that will be invoked to notify the user of an internal Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) event.

Note that the callback is registered per process so any of the above notifications that happen on the registered process will receive the callback. As with any callback mechanism it is up to the user to make sure that there are no possible deadlocks due to reentrancy (ie the callback being invoked in the same context that triggered the notification) and also that the callback does not slow down overall processing. If the user performs too much work within the callback it could delay further Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) processing. Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) promises to invoke the callback for an internal event prior to signaling the corresponding COIEvent. For example, if a user is waiting for a COIEvent associated with a run function completing they will receive the callback before the COIEvent is marked as signaled.

Parameters:

in_Type [in] The type of internal event that has occurred.

in_Process [in] The process associated with the operation.

in_Event [in] The completion event that is associated with the operation that is being notified.

in_UserData [in] Opaque data that was provided when the callback was registered. Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) simply passes this back to the user so that they can interpret it as they choose.

Definition at line 872 of file COIProcess_source.h.

5.16.2.2 typedef enum COI_NOTIFICATIONS COI_NOTIFICATIONS

The user can choose to have notifications for these internal events so that they can build their own profiling and performance layer on top of Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI).

5.16.3 Enumeration Type Documentation

5.16.3.1 enum COI_NOTIFICATIONS

The user can choose to have notifications for these internal events so that they can build their own profiling and performance layer on top of Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI).

Enumerator:

RUN_FUNCTION_READY This event occurs when all explicit and implicit dependencies are satisified and Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) schedules the run function to begin execution.

RUN_FUNCTION_START This event occurs just before the run function actually starts executing. There may be some latency between the ready and start events if other run functions are already queued and ready to run.

RUN_FUNCTION_COMPLETE This event occurs when the run function finishes. This is when the completion event for that run function would be signaled.

BUFFER_OPERATION_READY This event occurs when all explicit and implicit dependencies are met for the pending buffer operation. Assuming buffer needs to be moved, copied, read, etc... Will not be invoked if no actual memory is moved, copied, read, etc. This means that COIBufferUnmap will never result in a callback as it simply updates the status of the buffer but doesn't initiate any data movement. COIBufferMap, COIBufferSetState, COIBufferWrite, COIBufferRead and COIBufferCopy do initiate data movement and therefore will invoke the callback.

BUFFER_OPERATION_COMPLETE This event occurs when the buffer operation is completed.

USER_EVENT_SIGNALED This event occurs when a user event is signaled from the remotely a sink process. Local (source triggered) events do not trigger this.

Definition at line 802 of file COIProcess_source.h.

5.16.4 Function Documentation

5.16.4.1 __asm__ (

COIProcessLoadLibraryFromMemory,

)

5.16.4.2 __asm__ (

COIProcessLoadLibraryFromFile,

)

5.16.4.3 COIACCESSAPI void COINotificationCallbackSetContext (

in_UserData)

Set the user data that will be returned in the notification callback.

This data is sticky and per thread so must be set prior to the Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) operation being invoked. If you wish to set the context to be returned for a specific instance of a user event notification then the context must be set using this API prior to registering that user event with COIEventRegisterUserEvent. The value may be set prior to each Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) operation being called to effectively have a unique UserData per callback. Setting this value overrides any value that was set when the callback was registered and will also override any future registrations that occur.

Parameters:

in_UserData [in] Opaque data to pass to the callback when it is invoked. Note that this data is set per thread.

5.16.4.4 COIACCESSAPI COIRESULT COIProcessCreateFromFile (

in_Engine,

in_pBinaryName,

in_Argc,

in_ppArgv,

in_DupEnv,

in_ppAdditionalEnv,

in_ppArgv parameter.

in_ppArgv [in] An array of strings that represent the arguments being passed in. The system will auto-generate argv[0] using in_pBinaryName and thus that parameter cannot be passed in using in_ppArgv. Instead, in_ppArgv contains the rest of the parameters being passed in.

in_DupEnv [in] A boolean that indicates whether the process that is being created should inherit the environment of the caller.

in_ppAdditionalEnv [in] An array of strings that represent additional environment variables. This parameter must terminate the array with a NULL string. For convenience it is also allowed to be NULL if there are no additional environment variables that need adding. Note that any environment variables specified here will be in addition to but override those that were inherited via in_DupEnv.

in_ProxyActive [in] A boolean that specifies whether the process that is to be created wants I/O proxy support. If this flag is enabled, then stdout and stderr are forwarded back to the calling process's output and error streams.

in_Reserved Reserved for future use, best set at NULL.

in_InitialBufferSpace [in] The initial memory (in bytes) that will be pre-allocated at process creation for use by buffers associated with this remote process. In addition to allocating, Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) will also fault in the memory during process creation. If the total size of the buffers in use by this process exceed this initial size, memory on the sink may continue to be allocated on demand, as needed, subject to the system constraints on the sink.

in_LibrarySearchPath [in] a path to locate dynamic libraries dependencies for the sink application. If not NULL, this path will override the environment variable SINK_LD_LIBRARY_PATH. If NULL it will use SINK_LD_-LIBRARY_PATH to locate dependencies.

out_pProcess [out] Handle returned to uniquely identify the process that was created for use in later API calls.

Returns:

COI_SUCCESS if the remote process was successfully created.
COI_INVALID_POINTER if in_pBinaryName was NULL.
COI_INVALID_FILE if in_pBinaryName is not a "regular file" as determined by stat or if its size is 0.
COI_DOES_NOT_EXIST if in_pBinaryName cannot be found.
See COIProcessCreateFromMemory for additional errors.

5.16.4.5 COIACCESSAPI COIRESULT COIProcessCreateFromMemory (

in_Engine,

in_pBinaryName,

in_pBinaryBuffer,

in_BinaryBufferLength,

in_Argc,

in_ppArgv,

in_DupEnv,

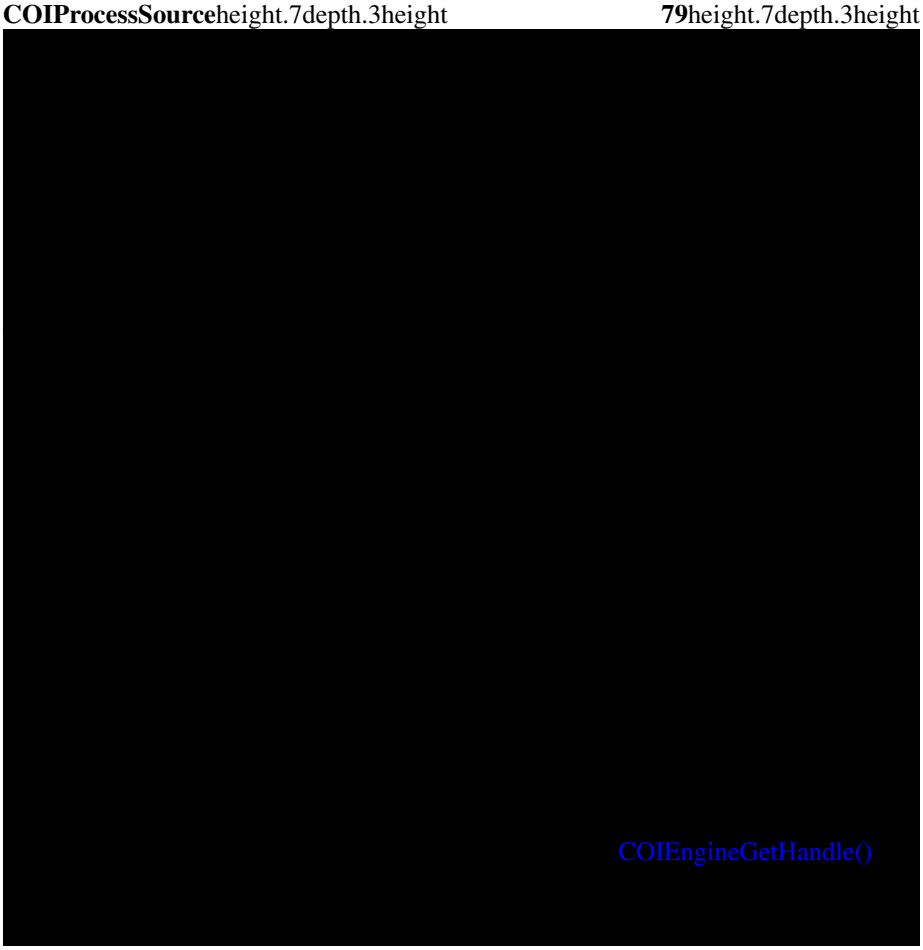
in_ppAdditionalEnv,

in_ProxyActive,

in_Reserved,

in_InitialBufferSpace,

in_LibrarySearchPath,



in_pBinaryName [in] Pointer to a null-terminated string that contains the name to give the process that will be created. Note that the final name will strip out any directory information from in_pBinaryName and use the file information to generate an argv[0] for the new process.

in_pBinaryBuffer [in] Pointer to a buffer whose contents represent the sink-side process that we want to create.

in_BinaryBufferLength [in] Number of bytes in in_pBinaryBuffer.

in_Argc [in] The number of arguments being passed in to the process in the in_ppArgv parameter.

in_ppArgv [in] An array of strings that represent the arguments being passed in. The system will auto-generate argv[0] using in_pBinaryName and thus that parameter cannot be passed in using in_ppArgv. Instead, in_ppArgv contains the rest of the parameters being passed in.

in_DupEnv [in] A boolean that indicates whether the process that is being created should inherit the environment of the caller.

in_ppAdditionalEnv [in] An array of strings that represent additional environment variables. This parameter must terminate the array with a

NULL string. For convenience it is also allowed to be NULL if there are no additional environment variables that need adding. Note that any environment variables specified here will be in addition to but override those that were inherited via in_DupEnv.

in_ProxyActive [in] A boolean that specifies whether the process that is to be created wants I/O proxy support.

in_Reserved Reserved for future use, best set to NULL.

in_InitialBufferSpace [in] The initial memory (in bytes) that will be pre-allocated at process creation for use by buffers associated with this remote process. In addition to allocating, Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) will also fault in the memory during process creation. If the total size of the buffers in use by this process exceed this initial size, memory on the sink may continue to be allocated on demand, as needed, subject to the system constraints on the sink.

in_LibrarySearchPath [in] A path to locate dynamic libraries dependencies for the sink application. If not NULL, this path will override the environment variable SINK_LD_LIBRARY_PATH. If NULL it will use SINK_LD_-LIBRARY_PATH to locate dependencies.

in_FileOfOrigin [in] If not NULL, this parameter indicates the file from which the in_pBinaryBuffer was obtained. This parameter is optional.

in_FileOfOriginOffset [in] If in_FileOfOrigin is not NULL, this parameter indicates the offset within that file where in_pBinaryBuffer begins.

out_pProcess [out] Handle returned to uniquely identify the process that was created for use in later API calls.

Returns:

COI_SUCCESS if the remote process was successfully created.
COI_INVALID_HANDLE if the in_Engine handle passed in was invalid.
COI_INVALID_POINTER if out_pProcess was NULL.
COI_INVALID_POINTER if in_pBinaryName or in_pBinaryBuffer was NULL.
COI_MISSING_DEPENDENCY if a dependent library is missing from either SINK_LD_LIBRARY_PATH or the in_LibrarySearchPath parameter.
COI_BINARY_AND_HARDWARE_MISMATCH if in_pBinaryName or any of its recursive dependencies were built for a target machine that does not match the engine specified.
COI_RESOURCE_EXHAUSTED if no more COIProcesses can be created, possibly, but not necessarily because in_InitialBufferSpace is too large.
COI_ARGUMENT_MISMATCH if in_Argc is 0 and in_ppArgv is not NULL.
COI_ARGUMENT_MISMATCH if in_Argc is greater than 0 and in_ppArgv is NULL.
COI_OUT_OF_RANGE if in_Argc is less than 0.
COI_OUT_OF_RANGE if the length of in_pBinaryName is greater than or equal to COI_MAX_FILE_NAME_LENGTH.

out_pProcessReturn,

out_pTerminationCode)

Destroys the indicated process, releasing its resources.

Note, this will destroy any outstanding pipelines created in this process as well.

Parameters:

in_Process [in] Process to destroy.

in_WaitForMainTimeout [in] The number of milliseconds to wait for the main() function to return in the sink process before timing out. -1 means to wait indefinitely.

in_ForceDestroy [in] If this flag is set to true, then the sink process will be forcibly terminated after the timeout has been reached. A timeout value of 0 will kill the process immediately, while a timeout of -1 is invalid. If the flag is set to false then a message will be sent to the sink process requesting a clean shutdown. A value of false along with a timeout of 0 does not send a shutdown message, instead simply polls the

process to see if it is alive. In most cases this flag should be set to false. If a sink process is not responding then it may be necessary to set this flag to true.

out_pProcessReturn [out] The value returned from the main() function executing in the sink process. This is an optional parameter. If the caller is not interested in the return value from the remote process they may pass in NULL for this parameter. The output value of this pointer is only meaningful if COI_SUCCESS is returned.

out_pTerminationCode [out] This parameter specifies the termination code. This will be 0 if the remote process exited cleanly. If the remote process exited abnormally this will contain the termination code given by the operating system of the remote process. This is an optional parameter and the caller may pass in NULL if they are not interested in the termination code. The output value of this pointer is only meaningful if COI_SUCCESS is returned.

Returns:

COI_SUCCESS if the process was destroyed.
COI_INVALID_HANDLE if the process handle passed in was invalid.
COI_OUT_OF_RANGE for any negative in_WaitForMainTimeout value except -1.
COI_ARGUMENT_MISMATCH if in_WaitForMainTimeout is -1 and in_ForceDestroy is true.
COI_TIME_OUT_REACHED if the sink process is still running after waiting in_WaitForMainTimeout milliseconds and in_ForceDestroy is false. This is true even if in_WaitForMainTimeout was 0. In this case, out_pProcessReturn and out_pTerminationCode are undefined.

5.16.4.7 COIACCESSAPI COIRESULT COIProcessGetFunctionHandles (

in_Process,

in_NumFunctions,

in_ppFunctionNameArray,

out_pFunctionHandleArray)

Given a loaded native process, gets an array of function handles that can be used to schedule run functions on a pipeline associated with that process.

See the documentation for COIPipelineRunFunction() for additional information. All functions that are to be retrieved in this fashion must have the define COINATIVEPROCESSEXPORT preceeding their type specification. For functions that are written in C++, either the entries in in_pFunctionNameArray in must be pre-mangled, or the functions must be declared as extern "C". It is also necessary to link the binary containing the exported functions with the -rdynamic linker flag. It is possible for this call to

successfully find function handles for some of the names passed in but not all of them. If this occurs COI_DOES_NOT_EXIST will return and any handles not found will be returned as NULL.

Parameters:

in_Process [in] Process handle previously returned via COIProcessCreate().

in_NumFunctions [in] Number of function names passed in to the in_pFunctionNames array.

in_ppFunctionNameArray [in] Pointer to an array of null-terminated strings that match the name of functions present in the code of the binary previously loaded via COIProcessCreate(). Note that if a C++ function is used, then the string passed in must already be properly name-mangled, or extern "C" must be used for where the function is declared.

out_pFunctionHandleArray [in out] Pointer to a location created by the caller large enough to hold an array of COIFUNCTION sized elements that has in_numFunctions entries in the array.

Returns:

COI_SUCCESS if all function names indicated were found.
COI_INVALID_HANDLE if the in_Process handle passed in was invalid.
COI_OUT_OF_RANGE if in_NumFunctions is zero.
COI_INVALID_POINTER if the in_ppFunctionNameArray or out_pFunctionHandleArray pointers was NULL.
COI_DOES_NOT_EXIST if one or more function names were not found. To determine the function names that were not found, check which elements in the out_pFunctionHandleArray are set to NULL.
COI_OUT_OF_RANGE if any of the null-terminated strings passed in via in_ppFunctionNameArray were more than COI_MAX_FUNCTION_NAME_LENGTH characters in length including the null.

Warning:

This operation can take several milliseconds so it is recommended that it only be be done at load time.

5.16.4.8 COIRESULT COIProcessLoadLibraryFromFile (

in_Process,

in_pFileName,

in_pLibraryName,

in_LibrarySearchPath,

out_pLibrary)

Loads a shared library into the specified remote process, akin to using dlopen() on a local process in Linux or LoadLibrary() in Windows.

For more details, see COIProcessLoadLibraryFromMemory.

Parameters:

in_Process [in] Process to load the library into.

in_pFileName [in] The name of the shared library file on the source's file system that is being loaded. If the file name is not an absolute path, the file is searched for in the same manner as dependencies.

in_pLibraryName [in] Name for the shared library. This optional parameter can be specified in case the dynamic library doesn't have an SO_NAME field. If specified, it will take precedence over the SO_NAME if it exists. If it is not specified then the library must have a valid SO_NAME field.

in_LibrarySearchPath [in] a path to locate dynamic libraries dependencies for the library being loaded. If not NULL, this path will override the environment variable SINK_LD_LIBRARY_PATH. If NULL it will use SINK_LD_LIBRARY_PATH to locate dependencies.

out_pLibrary [out] If COI_SUCCESS or COI_ALREADY_EXISTS is returned, the handle that uniquely identifies the loaded library.

Returns:

COI_SUCCESS if the library was successfully loaded.
COI_INVALID_POINTER if in_pFileName is NULL.
COI_DOES_NOT_EXIST if in_pFileName cannot be found.
COI_INVALID_FILE if the file is not a valid shared library.
See COIProcessLoadLibraryFromMemory for additional errors.

5.16.4.9 COIRESULT COIProcessLoadLibraryFromMemory (

in_Process,

in_pLibraryBuffer,

in_LibraryBufferLength,

in_pLibraryName,

in_LibrarySearchPath,

in_FileOfOrigin,

in_FileOfOriginOffset,

out_pLibrary)

Loads a shared library into the specified remote process, akin to using dlopen() on a local process in Linux or LoadLibrary() in Windows.

Dependencies for this library that are not listed with absolute paths are searched for first in current working directory, then in the colon-delimited paths in the environment variable SINK_LD_LIBRARY_PATH, and finally on the sink in the standard search paths as defined by the sink's operating system / dynamic loader.

Parameters:

in_Process [in] Process to load the library into.

in_pLibraryBuffer [in] The memory buffer containing the shared library to load.

in_LibraryBufferLength [in] The number of bytes in the memory buffer in_pLibraryBuffer.

in_pLibraryName [in] Name for the shared library. This optional parameter can be specified in case the dynamic library doesn't have an SO_NAME field. If specified, it will take precedence over the SO_NAME if it exists. If it is not specified then the library must have a valid SO_NAME field.

in_LibrarySearchPath [in] A path to locate dynamic libraries dependencies for the library being loaded. If not NULL, this path will override the environment variable SINK_LD_LIBRARY_PATH. If NULL it will use SINK_LD_LIBRARY_PATH to locate dependencies.

in_LibrarySearchPath [in] A path to locate dynamic libraries dependencies for the sink application. If not NULL, this path will override the environment variable SINK_LD_LIBRARY_PATH. If NULL it will use SINK_LD_-LIBRARY_PATH to locate dependencies.

in_FileOfOrigin [in] If not NULL, this parameter indicates the file from which the in_pBinaryBuffer was obtained. This parameter is optional.

in_FileOfOriginOffset [in] If in_FileOfOrigin is not NULL, this parameter indicates the offset within that file where in_pBinaryBuffer begins.

out_pLibrary [out] If COI_SUCCESS or COI_ALREADY_EXISTS is returned, the handle that uniquely identifies the loaded library.

Returns:

COI_SUCCESS if the library was successfully loaded.
COI_INVALID_HANDLE if the process handle passed in was invalid.
COI_OUT_OF_RANGE if in_LibraryBufferLength is 0.

COI_INVALID_FILE if in_pLibraryBuffer does not represent a valid shared library file.
COI_MISSING_DEPENDENCY if a dependent library is missing from either SINK_LD_LIBRARY_PATH or the in_LibrarySearchPath parameter.
COI_ARGUMENT_MISMATCH if the shared library is missing an SONAME and in_pLibraryName is NULL.
COI_ARGUMENT_MISMATCH if in_pLibraryName is the same as that of any of the dependencies (recursive) of the library being loaded.
COI_ALREADY_EXISTS if there is an existing COILIBRARY handle that identifies this library, and this COILIBRARY hasn't been unloaded yet.
COI_BINARY_AND_HARDWARE_MISMATCH if the target machine of the binary or any of its recursive dependencies does not match the engine associated with in_Process.
COI_UNDEFINED_SYMBOL if we are unable to load the library due to an undefined symbol.
COI_PROCESS_DIED if loading the library on the device caused the remote process to terminate.
COI_DOES_NOT_EXIST if in_FileOfOrigin is not NULL and does not exist.
COI_ARGUMENT_MISMATCH if in_FileOfOrigin is NULL and in_FileOfOriginOffset is not 0.
COI_INVALID_FILE if in_FileOfOrigin is not a "regular file" as determined by stat or if its size is 0.
COI_OUT_OF_RANGE if in_FileOfOrigin exists but its size is less than in_FileOfOriginOffset + in_BinaryBufferLength.
COI_INVALID_POINTER if out_pLibrary or in_pLibraryBuffer are NULL.

5.16.4.10 COIACCESSAPI COIRESULT COIProcessRegisterLibraries (

in_NumLibraries,

in_ppLibraryArray,

in_pLibrarySizeArray,

in_ppFileOfOriginArray,

in_pFileOfOriginOffSetArray)

Registers shared libraries that are already in the host process's memory to be used during the shared library dependency resolution steps that take place during subsequent calls to COIProcessCreate∗ and COIProcessLoadLibrary∗.

If listed as a dependency, the registered library will be used to satisfy the dependency, even if there is another library on disk that also satisfies that dependency.

Addresses registered must remain valid during subsequent calls to COIProcessCreate∗ and COIProcessLoadLibrary∗.

If the Sink is Linux, the shared libraries must have a library name (DT_SONAME field). On most compilers this means built with -soname.

If successful, this API registers all the libraries. Otherwise none are registered.

Parameters:

in_NumLibraries [in] The number of libraries that are being registered.

in_ppLibraryArray [in] An array of pointers that point to the starting addresses of the libraries.

in_pLibrarySizeArray [in] An array of pointers that point to the number of bytes in each of the libraries.

in_ppFileOfOriginArray [in] An array of strings indicating the file from which the library was obtained. This parameter is optional. Elements in the array may be set to NULL.

in_pFileOfOriginOffSetArray [in] If the corresponding entry in in_ppFileOfOriginArray is not NULL, this parameter indicates the offsets within those files where the corresponding libraries begin.

Returns:

COI_SUCCESS if the libraries were registered successfully.
COI_OUT_OF_RANGE if in_NumLibraries is 0.
COI_INVALID_POINTER if in_ppLibraryArray or in_pLibrarySizeArray are NULL.
COI_INVALID_POINTER if any of the pointers in in_ppLibraryArray are NULL.
COI_OUT_OF_RANGE if any of the values in in_pLibrarySizeArray is 0.
COI_ARGUMENT_MISMATCH if either one of in_ppFileOfOriginArray and in_pFileOfOriginOffSetArray is NULL and the other is not.
COI_OUT_OF_RANGE if one of the addresses being registered does not represent a valid library.

5.16.4.11 COIACCESSAPI COIRESULT COIProcessSetCacheSize (

in_Process,

in_HugePagePoolSize,

in_HugeFlags,

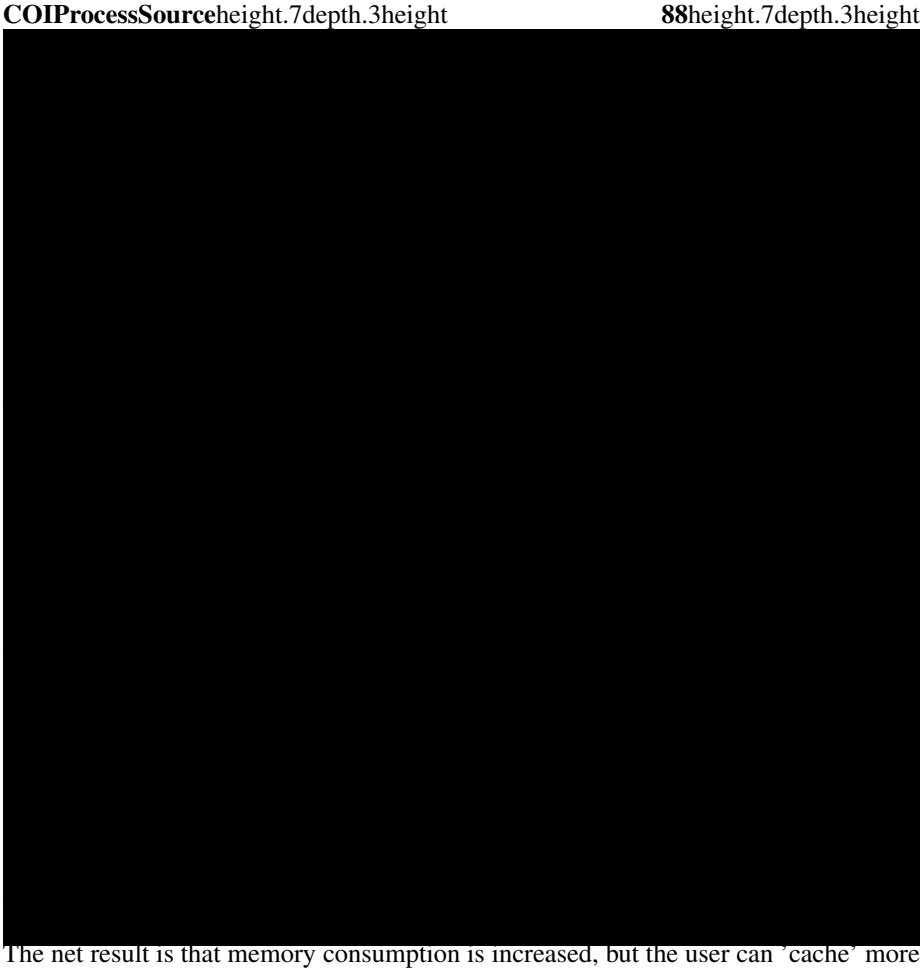
in_SmallPagePoolSize,

in_SmallFlags,

in_NumDependencies,

in_pDependencies,

out_pCompletion)



The net result is that memory consumption is increased, but the user can 'cache' more buffers on the remote process. More time may be spent during first use of run functions as more memory may be allocated, but subsequent run functions will likely see an increase in queueing performance as the data is already valid in the remote process.

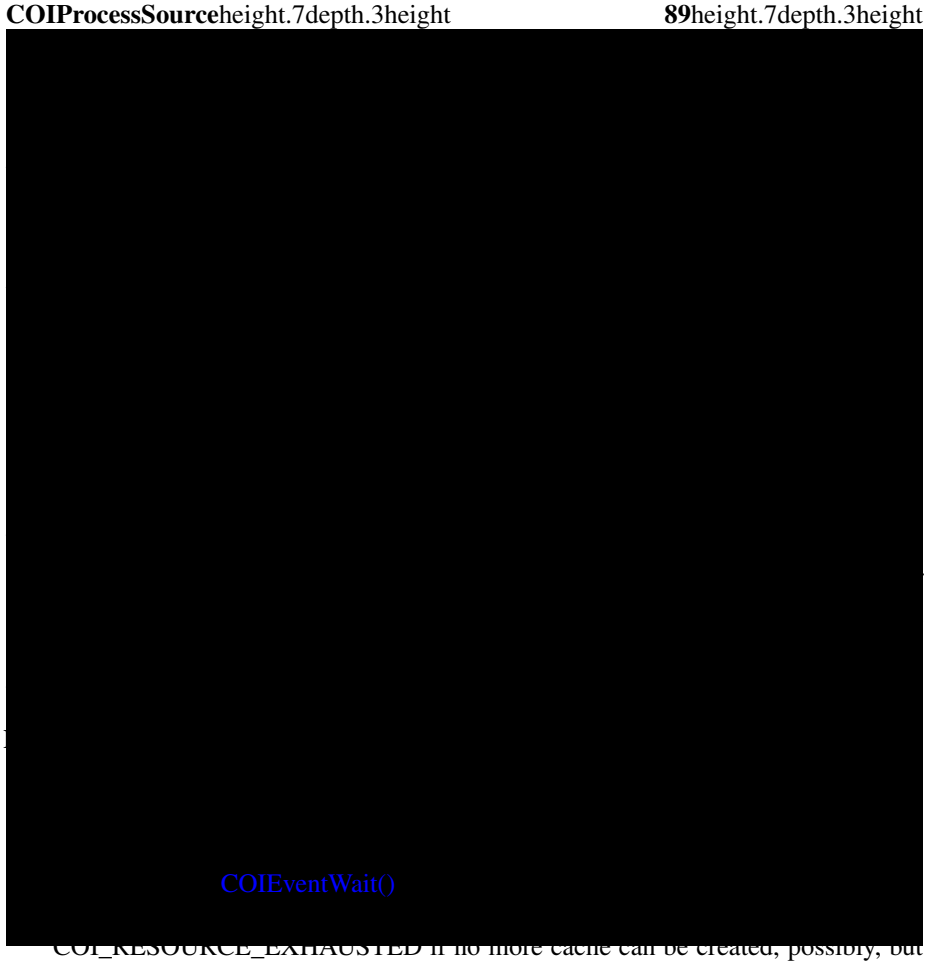
Users should tune this value for optimum performance balanced against memory consumption. This value does not affect 4K page cache. Please use in_SmallPagePoolSize for 4K pages.

Parameters:

in_HugeFlags [in] Flags to select mode or action for huge page cache. One _MODE_ and one _ACTION_ flag are specified together. Default _MODE_ is COI_CACHE_MODE_ONDEMAND_SYNC. See all COI_CACHE_-MODE_∗ and COI_CACHE_ACTION_∗ for other modes and actions. Default _ACTION_ is COI_CACHE_ACTION_NONE.

in_SmallPagePoolSize [in] The suggested size of the remote 4K cache in bytes. Same function as in_HugePagePoolSize but affecting only 4K page cache. Defaults to 1GB.

in_SmallFlags [in] Flags to select mode or action for 4K page cache. One _MODE_ and one _ACTION_ flag are be specified together. Default



COI_RESOURCE_EXHAUSTED if no more cache can be created, possibly, but not necessarily because a pool size was set to large and COI_CACHE_ACTION_-GROW_NOW was specified.
COI_NOT_SUPPORTED if more than one _MODE_ or _ACTION_ was specified.
COI_NOT_SUPPORTED if an invalid _MODE_ or _ACTION_ was specified.
COI_ARGUMENT_MISMATCH if in_NumDependencies is non-zero while in_-pDependencies was passed in as NULL.
COI_OUT_OF_RANGE if one of the pool sizes was invalid.
COI_PROCESS_DIED if at some point during the mode or action the remote process terminated abnormally. Possible due to an out of memory condition.

5.16.4.12 COIACCESSAPI COIRESULT COIProcessUnloadLibrary (

in_Process,

in_Library)

Unloads a a previously loaded shared library from the specified remote process.

Parameters:

in_Process [in] Process that we are unloading a library from.

in_Library [in] Library that we want to unload.

Returns:

COI_SUCCESS if the library was successfully loaded.
COI_INVALID_HANDLE if the process or library handle were invalid.

5.16.4.13 COIACCESSAPI COIRESULT COIRegisterNotificationCallback (

in_Process,

in_Callback,

in_UserData)

Register a callback to be invoked to notify that an internal Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) event has occured on the process that is associated with the callback.

Note that it is legal to have more than one callback registered with a given process but those must all be unique callback pointers. Note that setting a UserData value with COINotificationCallbackSetContext will override a value set when registering the callback.

Parameters:

in_Process [in] Process that the callback is associated with. The callback will only be invoked to notify an event for this specific process.

in_Callback [in] Pointer to a user function used to signal a notification.

in_UserData [in] Opaque data to pass to the callback when it is invoked.

Returns:

COI_SUCCESS if the callback was registered successfully.
COI_INVALID_HANDLE if the in_Process parameter does not identify a valid process.
COI_INVALID_POINTER if the in_Callback parameter is NULL.
COI_ALREADY_EXISTS if the user attempts to reregister the same callback for a process.

5.16.4.14 COIACCESSAPI COIRESULT COIUnregisterNotificationCallback (in_Process, in_Callback)

Unregisters a callback, notifications will no longer be signaled.

Parameters:

in_Process [in] Process that we are unregistering.

in_Callback [in] The specific callback to unregister.

Returns:

COI_SUCCESS if the callback was unregistered.
COI_INVALID_HANDLE if the in_Process parameter does not identify a valid process.
COI_INVALID_POINTER if the in_Callback parameter is NULL.
COI_DOES_NOT_EXIST if in_Callback was not previously registered for in_Process.

5.17 COIBufferSink

Functions

- COIRESULT COIBufferAddRef (void *in_pBuffer)
  Adds a reference to the memory of a buffer.
- COIRESULT COIBufferReleaseRef (void *in_pBuffer)
  Removes a reference to the memory of a buffer.

5.17.1 Function Documentation

5.17.1.1 COIRESULT COIBufferAddRef (in_pBuffer)

Adds a reference to the memory of a buffer.

The memory of the buffer will remain on the device until both a corresponding COIBufferReleaseRef() call is made and the run function that delivered the buffer returns.

Running this API in a thread spawned within the run function is not supported and will cause unpredictable results and may cause data corruption.

Intel® Coprocessor Offload Infrastructure (Intel® COI) streaming buffers should not be AddRef'd. Doing so may result in unpredictable results or may cause the sink process to crash.

Warning:

1.It is possible for enqueued run functions to be unable to execute due to all card memory being occupied by addref'ed buffers. As such, it is important that whenever a buffer is addref'd that there be no dependencies on future run functions for progress to be made towards releasing the buffer. 2.It is important that AddRef is called within the scope of run function that carries the buffer to be addref'ed.

Parameters:

in_pBuffer [in] Pointer to the start of a buffer being addref'ed, that was passed in at the start of the run function.

Returns:

COI_SUCCESS if the buffer ref count was successfully incremented.
COI_INVALID_POINTER if the buffer pointer is NULL.
COI_INVALID_HANDLE if the buffer pointer is invalid.

5.17.1.2 COIRESULT COIBufferReleaseRef (

in_pBuffer)

Removes a reference to the memory of a buffer.

The memory of the buffer will be eligible for being freed on the device when the following conditions are met: the run function that delivered the buffer returns, and the number of calls to COIBufferReleaseRef() matches the number of calls to COIBufferAddRef(). Running this API in a thread spawned within the run function is not supported and will cause unpredictable results and may cause data corruption.

Warning:

When a buffer is addref'ed it is assumed that it is in use and all other operations on that buffer waits for ReleaseRef() to happen. So you cannot pass the addref'ed buffer's handle to RunFunction that calls ReleaseRef(). This is a circular dependency and will cause a deadlock. Buffer's pointer (buffer's sink side address/-pointer which is different than source side BUFFER handle) needs to be stored somewhere to retrieve it later to use in ReleaseRef.

Parameters:

in_pBuffer [in] Pointer to the start of a buffer previously addref'ed, that was passed in at the start of the run function.

Returns:

COI_SUCCESS if the buffer refcount was successfully decremented.
COI_INVALID_POINTER if the buffer pointer was invalid.
COI_INVALID_HANDLE if the buffer did not have COIBufferAddRef() previously called on it.

5.18 COIPipelineSink

Files

- file COIPipeline_sink.h

Typedefs

- typedef void(∗ RunFunctionPtr_t)(uint32_t in_BufferCount, void ∗∗in_ppBufferPointers, uint64_t ∗in_pBufferLengths, void ∗in_pMiscData, uint16_t in_MiscDataLength, void ∗in_pReturnValue, uint16_t in_ReturnValueLength)

  This is the prototype that run functions should follow.

Functions

- COIRESULT COIPipelineStartExecutingRunFunctions ()

  Start processing pipelines on the Sink.

5.18.1 Typedef Documentation

5.18.1.1 typedef void(∗ RunFunctionPtr_t)(uint32_t in_BufferCount, void ∗∗in_ppBufferPointers, uint64_t ∗in_pBufferLengths, void ∗in_pMiscData, uint16_t in_MiscDataLength, void ∗in_pReturnValue, uint16_t in_ReturnValueLength)

This is the prototype that run functions should follow.

Parameters:

in_BufferCount The number of buffers passed to the run function.

in_ppBufferPointers An array that is in_BufferCount in length that contains the sink side virtual addresses for each buffer passed in to the run function.

in_pBufferLengths An array that is in_BufferCount in length of uint32_t integers describing the length of each passed in buffer in bytes.

in_pMiscData Pointer to the MiscData passed in when the run function was enqueued on the source.

in_MiscDataLen Length in bytes of the MiscData passed in when the run function was enqueued on the source.

in_pReturnValue Pointer to the location where the return value from this run function will be stored.

in_ReturnValueLength Length in bytes of the user-allocated ReturnValue pointer.

Returns:

A uint64_t that can be retrieved in the out_UserData parameter from the COIPipelineWaitForEvent function.

Definition at line 103 of file COIPipeline_sink.h.

5.18.2 Function Documentation

5.18.2.1 COIRESULT COIPipelineStartExecutingRunFunctions ( )

Start processing pipelines on the Sink.

This should be done after any required initialization in the Sink's application has finished. No run functions will actually be executed (although they may be queued) until this function is called.

Returns:

COI_SUCCESS if the pipelines were successfully started.

5.19 COIProcessSink

Files

- file COIProcess_sink.h

Functions

- COIRESULT COIProcessProxyFlush ()

  This call will block until all stdout and stderr output has been proxied to and written by the source.

- COIRESULT COIProcessWaitForShutdown ()

  This call will block while waiting for the source to send a process destroy message.

5.19.1 Function Documentation

5.19.1.1 COIRESULT COIProcessProxyFlush ( )

This call will block until all stdout and stderr output has been proxied to and written by the source.

This call guarantees that any output in a run function is transmitted to the source before the run function signals its completion event back to the source.

Note that having an additional thread printing forever while another calls COIProxyFlush may lead to a hang because the process will be forced to wait until all that output can be flushed to the source before returning from this call.

Returns:

COI_SUCCESS once the proxy output has been flushed to and written written by the host. Note that Intel® Coprocessor Offload Infrastructure (Intel® COI) on the source writes to stdout and stderr, but does not flush this output.
COI_SUCCESS if the process was created without enabling proxy IO this function.

5.19.1.2 COIRESULT COIProcessWaitForShutdown ( )

This call will block while waiting for the source to send a process destroy message.

This provides the sink side application with an event to keep the main() function from exiting until it is directed to by the source. When the shutdown message is received this function will stop any future run functions from executing but will wait for any current run functions to complete. All Intel® Coprocessor Offload Infrastructure (Intel® COI) resources will be cleaned up and no additional Intel® Coprocessor Offload Infrastructure (Intel® COI) APIs should be called after this function returns. This function does not invoke exit() so the application can perform any of its own cleanup once this call returns.

Returns:

COI_SUCCESS once the process receives the shutdown message.

6 Data Structure Documentation

6.1 arr_desc Struct Reference

Data Fields

- int64_t base

- dim_desc dim [3]
- int64_t rank

6.1.1 Detailed Description

Definition at line 383 of file COIBuffer_source.h.

6.1.2 Field Documentation

6.1.2.1 int64_t arr_desc::base

Definition at line 384 of file COIBuffer_source.h.

6.1.2.2 dim_desc arr_desc::dim[3]

Definition at line 386 of file COIBuffer_source.h.

6.1.2.3 int64_t arr_desc::rank

Definition at line 385 of file COIBuffer_source.h.

6.2 COI_ENGINE_INFO Struct Reference

This structure returns information about an Intel(R) Xeon Phi(TM) coprocessor.

Data Fields

- uint16_t BoardSKU

  The SKU of the stepping, EB, ED, etc.

- uint16_t BoardStepping

  The stepping of the board, A0, A1, C0, D0 etc.

- uint32_t CoreMaxFrequency

  The maximum frequency (in MHz) of the cores on the engine.

- uint16_t DeviceId

  The pci config device id.

- coi_wchar_t DriverVersion [COI_MAX_DRIVER_VERSION_STR_LEN]

The version string identifying the driver.

- COI_ISA_TYPE ISA
  The ISA supported by the engine.
- uint32_t Load [COI_MAX_HW_THREADS]
  The load percentage for each of the hardware threads on the engine.
- coi_eng_misc MiscFlags
  Miscellaneous fields.
- uint32_t NumCores
  The number of cores on the engine.
- uint32_t NumThreads
  The number of hardware threads on the engine.
- uint64_t PhysicalMemory
  The amount of physical memory managed by the OS.
- uint64_t PhysicalMemoryFree
  The amount of free physical memory in the OS.
- uint16_t SubSystemId
  The pci config subsystem id.
- uint64_t SwapMemory
  The amount of swap memory managed by the OS.
- uint64_t SwapMemoryFree
  The amount of free swap memory in the OS.
- uint16_t VendorId
  The pci config vendor id.

6.2.1 Detailed Description

This structure returns information about an Intel(R) Xeon Phi(TM) coprocessor. A pointer to this structure is passed into the COIGetEngineInfo() function, which fills in the data before returning to the caller.

Definition at line 78 of file COIEngine_source.h.

6.2.2 Field Documentation

6.2.2.1 uint16_t COI_ENGINE_INFO::BoardSKU

The SKU of the stepping, EB, ED, etc.

Definition at line 127 of file COIEngine_source.h.

6.2.2.2 uint16_t COI_ENGINE_INFO::BoardStepping

The stepping of the board, A0, A1, C0, D0 etc.

Definition at line 124 of file COIEngine_source.h.

6.2.2.3 uint32_t COI_ENGINE_INFO::CoreMaxFrequency

The maximum frequency (in MHz) of the cores on the engine.

Definition at line 96 of file COIEngine_source.h.

6.2.2.4 uint16_t COI_ENGINE_INFO::DeviceId

The pci config device id.

Definition at line 118 of file COIEngine_source.h.

6.2.2.5 coi_wchar_t COI_ENGINE_INFO::DriverVersion[COI_MAX_DRIVER_VERSION_STR_LEN]

The version string identifying the driver.

Definition at line 81 of file COIEngine_source.h.

6.2.2.6 COI_ISA_TYPE COI_ENGINE_INFO::ISA

The ISA supported by the engine.

Definition at line 84 of file COIEngine_source.h.

6.2.2.7 uint32_t COI_ENGINE_INFO::Load[COI_MAX_HW_THREADS]

The load percentage for each of the hardware threads on the engine.
Currently this is limited to reporting out a maximum of 1024 HW threads
Definition at line 100 of file COIEngine_source.h.

6.2.2.8 coi_eng_misc COI_ENGINE_INFO::MiscFlags

Miscellaneous fields.
Definition at line 90 of file COIEngine_source.h.

6.2.2.9 uint32_t COI_ENGINE_INFO::NumCores

The number of cores on the engine.
Definition at line 87 of file COIEngine_source.h.

6.2.2.10 uint32_t COI_ENGINE_INFO::NumThreads

The number of hardware threads on the engine.
Definition at line 93 of file COIEngine_source.h.

6.2.2.11 uint64_t COI_ENGINE_INFO::PhysicalMemory

The amount of physical memory managed by the OS.
Definition at line 103 of file COIEngine_source.h.

6.2.2.12 uint64_t COI_ENGINE_INFO::PhysicalMemoryFree

The amount of free physical memory in the OS.
Definition at line 106 of file COIEngine_source.h.

6.2.2.13 uint16_t COI_ENGINE_INFO::SubSystemId

The pci config subsystem id.

Definition at line 121 of file COIEngine_source.h.

6.2.2.14 uint64_t COI_ENGINE_INFO::SwapMemory

The amount of swap memory managed by the OS.

Definition at line 109 of file COIEngine_source.h.

6.2.2.15 uint64_t COI_ENGINE_INFO::SwapMemoryFree

The amount of free swap memory in the OS.

Definition at line 112 of file COIEngine_source.h.

6.2.2.16 uint16_t COI_ENGINE_INFO::VendorId

The pci config vendor id.

Definition at line 115 of file COIEngine_source.h.

6.3 coievent Struct Reference

Data Fields

- uint64_t opaque [2]

6.3.1 Detailed Description

Definition at line 58 of file COITypes_common.h.

6.3.2 Field Documentation

6.3.2.1 uint64_t coievent::opaque[2]

Definition at line 58 of file COITypes_common.h.

6.4 dim_desc Struct Reference

Data Fields

- int64_t lindex
- int64_t lower
- int64_t size
- int64_t stride
- int64_t upper

6.4.1 Detailed Description

Definition at line 373 of file COIBuffer_source.h.

6.4.2 Field Documentation

6.4.2.1 int64_t dim_desc::lindex

Definition at line 375 of file COIBuffer_source.h.

6.4.2.2 int64_t dim_desc::lower

Definition at line 376 of file COIBuffer_source.h.

6.4.2.3 int64_t dim_desc::size

Definition at line 374 of file COIBuffer_source.h.

6.4.2.4 int64_t dim_desc::stride

Definition at line 378 of file COIBuffer_source.h.

6.4.2.5 int64_t dim_desc::upper

Definition at line 377 of file COIBuffer_source.h.

7 File Documentation

7.1 COIBuffer_sink.h File Reference

Functions

- COIRESULT COIBufferAddRef (void *in_pBuffer)

  Adds a reference to the memory of a buffer.

- COIRESULT COIBufferReleaseRef (void *in_pBuffer)

  Removes a reference to the memory of a buffer.

7.2 COIBuffer_source.h File Reference

Data Structures

- struct arr_desc
- struct dim_desc

Defines

- #define COI_SINK_OWNERS ((COIPROCESS)-2)

COIBUFFER creation flags.

Please see the COI_VALID_BUFFER_TYPES_AND_FLAGS matrix below which describes the valid combinations of buffer types and flags.

- #define COI_SAME_ADDRESS_SINKS 0x00000001

  Create the buffer such that it has the same virtual address on all of the sink processes with which it is associated.

- #define COI_SAME_ADDRESS_SINKS_AND_SOURCE 0x00000002

  Create the buffer such that it has the same virtual address on all of the sink processes with which it is associated and in the source process.

- #define COI_OPTIMIZE_SOURCE_READ 0x00000004

  Hint to the runtime that the source will frequently read the buffer.

- #define COI_OPTIMIZE_SOURCE_WRITE 0x00000008

  Hint to the runtime that the source will frequently write the buffer.

- #define COI_OPTIMIZE_SINK_READ 0x00000010

  Hint to the runtime that the sink will frequently read the buffer.

- #define COI_OPTIMIZE_SINK_WRITE 0x00000020

  Hint to the runtime that the sink will frequently write the buffer.

- #define COI_OPTIMIZE_NO_DMA 0x00000040
  Used to delay the pinning of memory into physical pages, until required for DMA.
- #define COI_OPTIMIZE_HUGE_PAGE_SIZE 0x00000080
  Hint to the runtime to try to use huge page sizes for backing store on the sink.
- #define COI_SINK_MEMORY 0x00000100
  Used to tell Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) to create a buffer using memory that has already been allocated on the sink.

Typedefs

- typedef enum COI_BUFFER_TYPE COI_BUFFER_TYPE
  The valid buffer types that may be created using COIBufferCreate.
- typedef enum COI_COPY_TYPE COI_COPY_TYPE
  The valid copy operation types for the COIBufferWrite, COIBufferRead, and COIBufferCopy APIs.
- typedef enum COI_MAP_TYPE COI_MAP_TYPE
  These flags control how the buffer will be accessed on the source after it is mapped.

Enumerations

- enum COI_BUFFER_MOVE_FLAG {
  COI_BUFFER_MOVE = 0,
  COI_BUFFER_NO_MOVE }
  Note: A VALID_MAY_DROP declares a buffer's copy as secondary on a given process.
- enum COI_BUFFER_STATE {
  COI_BUFFER_VALID = 0,
  COI_BUFFER_INVALID,
  COI_BUFFER_VALID_MAY_DROP,
  COI_BUFFER_RESERVED }
  The buffer states are used to indicate whether a buffer is available for access in a COIPROCESS.
- enum COI_BUFFER_TYPE {
  COI_BUFFER_NORMAL = 1,
  COI_BUFFER_STREAMING_TO_SINK,
  COI_BUFFER_STREAMING_TO_SOURCE,
  COI_BUFFER_PINNED,
  COI_BUFFER_OPENCL }

The valid buffer types that may be created using COIBufferCreate.

- enum COI_COPY_TYPE {
  COI_COPY_UNSPECIFIED = 0,
  COI_COPY_USE_DMA,
  COI_COPY_USE_CPU,
  COI_COPY_UNSPECIFIED_MOVE_ENTIRE,
  COI_COPY_USE_DMA_MOVE_ENTIRE,
  COI_COPY_USE_CPU_MOVE_ENTIRE }

  The valid copy operation types for the COIBufferWrite, COIBufferRead, and COIBufferCopy APIs.

- enum COI_MAP_TYPE {
  COI_MAP_READ_WRITE = 1,
  COI_MAP_READ_ONLY,
  COI_MAP_WRITE_ENTIRE_BUFFER }

  These flags control how the buffer will be accessed on the source after it is mapped.

Functions

- COIACCESSAPI COIRESULT COIBufferAddRefcnt (COIPROCESS in_Process, COIBUFFER in_Buffer, uint64_t in_AddRefcnt)

  Increments the reference count on the specified buffer and process by in_AddRefcnt.

- COIACCESSAPI COIRESULT COIBufferCopy (COIBUFFER in_DestBuffer, COIBUFFER in_SourceBuffer, uint64_t in_DestOffset, uint64_t in_SourceOffset, uint64_t in_Length, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data between two buffers.

- COIACCESSAPI COIRESULT COIBufferCopyEx (COIBUFFER in_DestBuffer, const COIPROCESS in_DestProcess, COIBUFFER in_SourceBuffer, uint64_t in_DestOffset, uint64_t in_SourceOffset, uint64_t in_Length, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data between two buffers.

- COIACCESSAPI COIRESULT COIBufferCreate (uint64_t in_Size, COI_BUFFER_TYPE in_Type, uint32_t in_Flags, const void *in_pInitData, uint32_t in_NumProcesses, const COIPROCESS *in_pProcesses, COIBUFFER *out_pBuffer)

  Creates a buffer that can be used in RunFunctions that are queued in pipelines.

- COIACCESSAPI COIRESULT COIBufferCreateFromMemory (uint64_t in_Size, COI_BUFFER_TYPE in_Type, uint32_t in_Flags, void *in_Memory, uint32_t in_NumProcesses, const COIPROCESS *in_pProcesses, COIBUFFER *out_pBuffer)

  Creates a buffer from some existing memory that can be used in RunFunctions that are queued in pipelines.

- COIACCESSAPI COIRESULT COIBufferCreateSubBuffer (COIBUFFER in_Buffer, uint64_t in_Length, uint64_t in_Offset, COIBUFFER *out_pSubBuffer)

  Creates a sub-buffer that is a reference to a portion of an existing buffer.

- COIACCESSAPI COIRESULT COIBufferDestroy (COIBUFFER in_Buffer)

  Destroys a buffer.

- COIACCESSAPI COIRESULT COIBufferGetSinkAddress (COIBUFFER in_Buffer, uint64_t *out_pAddress)

  Gets the Sink's virtual address of the buffer.

- COIACCESSAPI COIRESULT COIBufferMap (COIBUFFER in_Buffer, uint64_t in_Offset, uint64_t in_Length, COI_MAP_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion, COIMAPINSTANCE *out_pMapInstance, void **out_ppData)

  This call initiates a request to access a region of a buffer.

- COIACCESSAPI COIRESULT COIBufferRead (COIBUFFER in_SourceBuffer, uint64_t in_Offset, void *in_pDestData, uint64_t in_Length, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data from a buffer into local memory.

- COIACCESSAPI COIRESULT COIBufferReadMultiD (COIBUFFER in_SourceBuffer, uint64_t in_Offset, struct arr_desc *in_DestArray, struct arr_desc *in_SrcArray, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data specified by multi-dimensional array data structure from an existing COIBUFFER to another multi-dimensional array located in memory.

- COIACCESSAPI COIRESULT COIBufferReleaseRefcnt (COIPROCESS in_Process, COIBUFFER in_Buffer, uint64_t in_ReleaseRefcnt)

  Releases the reference count on the specified buffer and process by in_ReleaseRefcnt.

- COIACCESSAPI COIRESULT COIBufferSetState (COIBUFFER in_Buffer, COIPROCESS in_Process, COI_BUFFER_STATE in_State, COI_BUFFER_MOVE_FLAG in_DataMove, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

This API allows an experienced Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) developer to set where a COIBUFFER is located and when the COIBUFFER's data is moved.

- COIACCESSAPI COIRESULT COIBufferUnmap (COIMAPINSTANCE in_MapInstance, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Disables Source access to the region of the buffer that was provided through the corresponding call to COIBufferMap.

- COIACCESSAPI COIRESULT COIBufferWrite (COIBUFFER in_DestBuffer, uint64_t in_Offset, const void *in_pSourceData, uint64_t in_Length, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data from a normal virtual address into an existing COIBUFFER.

- COIACCESSAPI COIRESULT COIBufferWriteEx (COIBUFFER in_DestBuffer, const COIPROCESS in_DestProcess, uint64_t in_Offset, const void *in_pSourceData, uint64_t in_Length, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data from a normal virtual address into an existing COIBUFFER.

- COIACCESSAPI COIRESULT COIBufferWriteMultiD (COIBUFFER in_DestBuffer, const COIPROCESS in_DestProcess, uint64_t in_Offset, struct arr_desc *in_DestArray, struct arr_desc *in_SrcArray, COI_COPY_TYPE in_Type, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Copy data specified by multi-dimensional array data structure into another multi-dimensional array in an existing COIBUFFER.

7.3 COIEngine_common.h File Reference

Defines

- #define COI_MAX_ISA_KNC_DEVICES COI_MAX_ISA_MIC_DEVICES
- #define COI_MAX_ISA_KNF_DEVICES COI_MAX_ISA_MIC_DEVICES
- #define COI_MAX_ISA_MIC_DEVICES 128
- #define COI_MAX_ISA_x86_64_DEVICES 1

Enumerations

- enum COI_ISA_TYPE {
  COI_ISA_INVALID = 0,
  COI_ISA_x86_64,
  COI_ISA_MIC,

COI_ISA_KNF,

COI_ISA_KNC }

List of ISA types of supported engines.

Functions

- COIACCESSAPI COIRESULT COIEngineGetIndex (COI_ISA_TYPE *out_pType, uint32_t *out_pIndex)

  Get the information about the COIEngine executing this function call.

7.3.1 Detailed Description

Definition in file COIEngine_common.h.

7.4 COIEngine_source.h File Reference

Data Structures

- struct COI_ENGINE_INFO

  This structure returns information about an Intel(R) Xeon Phi(TM) coprocessor.

Defines

- #define COI_MAX_DRIVER_VERSION_STR_LEN 255
- #define COI_MAX_HW_THREADS 1024

Typedefs

- typedef struct COI_ENGINE_INFO COI_ENGINE_INFO

  This structure returns information about an Intel(R) Xeon Phi(TM) coprocessor.

Enumerations

- enum coi_eng_misc {

  COI_ENG_ECC_DISABLED = 0,

  COI_ENG_ECC_ENABLED = 0x00000001,

  COI_ENG_ECC_UNKNOWN = 0x00000002 }

  This enum defines miscellaneous information returned from the COIGetEngineInfo() function.

Functions

- COIACCESSAPI COIRESULT COIEngineGetCount (COI_ISA_TYPE in_ISA, uint32_t *out_pNumEngines)

  Returns the number of engines in the system that match the provided ISA.

- COIACCESSAPI COIRESULT COIEngineGetHandle (COI_ISA_TYPE in_ISA, uint32_t in_EngineIndex, COIENGINE *out_pEngineHandle)

  Returns the handle of a user specified engine.

- COIACCESSAPI COIRESULT COIEngineGetInfo (COIENGINE in_EngineHandle, uint32_t in_EngineInfoSize, COI_ENGINE_INFO *out_pEngineInfo)

  Returns information related to a specified engine.

7.5 COIEvent_common.h File Reference

Functions

- COIACCESSAPI COIRESULT COIEventSignalUserEvent (COIEVENT in_Event)

  Signal one shot user event.

7.5.1 Detailed Description

Definition in file COIEvent_common.h.

7.6 COIEvent_source.h File Reference

Defines

- #define COI_EVENT_ASYNC ((COIEVENT*)1)

  Special case event values which can be passed in to APIs to specify how the API should behave.

- #define COI_EVENT_SYNC ((COIEVENT*)2)

Functions

- COIACCESSAPI COIRESULT COIEventRegisterUserEvent (COIEVENT *out_pEvent)

  Register a User COIEVENT so that it can be fired.

- COIACCESSAPI COIRESULT COIEventUnregisterUserEvent (COIEVENT in_Event)

  Unregister a User COIEVENT.

- COIACCESSAPI COIRESULT COIEventWait (uint16_t in_NumEvents, const COIEVENT *in_pEvents, int32_t in_TimeoutMilliseconds, uint8_t in_WaitForAll, uint32_t *out_pNumSignaled, uint32_t *out_pSignaledIndices)

  Wait for an arbitrary number of COIEVENTs to be signaled as completed, eg when the run function or asynchronous map call associated with an event has finished execution.

7.6.1 Detailed Description

Definition in file COIEvent_source.h.

7.7 COIMacros_common.h File Reference

Commonly used macros.

Defines

- #define SYMBOL_VERSION(SYMBOL, VERSION) SYMBOL ## VERSION
- #define UNREFERENCED_CONST_PARAM(P)
- #define UNREFERENCED_PARAM(P) (P = P)
- #define UNUSED_ATTR __attribute__((unused))

Functions

- static int __COI_CountBits (uint64_t n)
- static void COI_CPU_MASK_AND (COI_CPU_MASK dst, const COI_CPU_MASK src1, const COI_CPU_MASK src2)
- static int COI_CPU_MASK_COUNT (const COI_CPU_MASK cpu_mask)
- static int COI_CPU_MASK_EQUAL (const COI_CPU_MASK cpu_mask1, const COI_CPU_MASK cpu_mask2)
- static uint64_t COI_CPU_MASK_ISSET (int bitNumber, const COI_CPU_MASK cpu_mask)
- static void COI_CPU_MASK_OR (COI_CPU_MASK dst, const COI_CPU_MASK src1, const COI_CPU_MASK src2)
- static void COI_CPU_MASK_SET (int bitNumber, COI_CPU_MASK cpu_mask)
- static void COI_CPU_MASK_XLATE (COI_CPU_MASK dest, const cpu_set_t *src)
- static void COI_CPU_MASK_XLATE_EX (cpu_set_t *dest, const COI_CPU_MASK src)

- static void COI_CPU_MASK_XOR (COI_CPU_MASK dst, const COI_CPU_-MASK src1, const COI_CPU_MASK src2)
- static void COI_CPU_MASK_ZERO (COI_CPU_MASK cpu_mask)

7.7.1 Detailed Description

Commonly used macros.

Definition in file COIMacros_common.h.

7.7.2 Define Documentation

7.7.2.1 #define SYMBOL_VERSION(

SYMBOL,

VERSION) SYMBOL ## VERSION

Definition at line 65 of file COIMacros_common.h.

7.7.2.2 #define UNREFERENCED_CONST_PARAM(

P)

Value:

```
{ void* x UNUSED_ATTR = \
                                                (void*)(uint64_t)P; \
                                          }
```

Definition at line 51 of file COIMacros_common.h.

7.7.2.3 #define UNREFERENCED_PARAM(

P) (P = P)

Definition at line 58 of file COIMacros_common.h.

7.7.2.4 #define UNUSED_ATTR __attribute__((unused))

Definition at line 48 of file COIMacros_common.h.

7.7.3 Function Documentation

7.7.3.1 static int __COI_CountBits (

n) [inline, static]

Definition at line 130 of file COIMacros_common.h.
Referenced by COI_CPU_MASK_COUNT().

7.7.3.2 static void COI_CPU_MASK_AND (

dst,

src1,

src2) [inline, static]

Definition at line 103 of file COIMacros_common.h.

7.7.3.3 static int COI_CPU_MASK_COUNT (

cpu_mask) [inline, static]

Definition at line 140 of file COIMacros_common.h.
References __COI_CountBits().

7.7.3.4 static int COI_CPU_MASK_EQUAL (

cpu_mask1,

cpu_mask2) [inline, static]

Definition at line 153 of file COIMacros_common.h.

7.7.3.5 static uint64_t COI_CPU_MASK_ISSET (

bitNumber,

cpu_mask) [inline, static]

Definition at line 82 of file COIMacros_common.h.
Referenced by COI_CPU_MASK_XLATE_EX().

7.7.3.6 static void COI_CPU_MASK_OR (

dst,

src1,

src2) [inline, static]

Definition at line 121 of file COIMacros_common.h.

7.7.3.7 static void COI_CPU_MASK_SET (

bitNumber,

cpu_mask) [inline, static]

Definition at line 90 of file COIMacros_common.h.
Referenced by COI_CPU_MASK_XLATE().

7.7.3.8 static void COI_CPU_MASK_XLATE (

dest,

src) [inline, static]

Definition at line 167 of file COIMacros_common.h.
References COI_CPU_MASK_SET(), and COI_CPU_MASK_ZERO().

7.7.3.9 static void COI_CPU_MASK_XLATE_EX (

dest,

src) [inline, static]

Definition at line 189 of file COIMacros_common.h.
References COI_CPU_MASK_ISSET().

7.7.3.10 static void COI_CPU_MASK_XOR (

dst,

src1,

src2) [inline, static]

Definition at line 112 of file COIMacros_common.h.

7.7.3.11 static void COI_CPU_MASK_ZERO (

cpu_mask) [inline, static]

Definition at line 97 of file COIMacros_common.h.

Referenced by COI_CPU_MASK_XLATE().

7.8 COIPerf_common.h File Reference

Performance Analysis API.

Functions

- COIACCESSAPI uint64_t COIPerfGetCycleCounter (void)
  Returns a performance counter value.
- COIACCESSAPI uint64_t COIPerfGetCycleFrequency (void)
  Returns the calculated system frequency in hertz.

7.8.1 Detailed Description

Performance Analysis API.

Definition in file COIPerf_common.h.

7.9 COIPipeline_sink.h File Reference

Typedefs

- typedef void(∗ RunFunctionPtr_t)(uint32_t in_BufferCount, void ∗∗in_ppBufferPointers, uint64_t ∗in_pBufferLengths, void ∗in_pMiscData, uint16_t in_MiscDataLength, void ∗in_pReturnValue, uint16_t in_ReturnValueLength)

This is the prototype that run functions should follow.

Functions

- COIRESULT COIPipelineStartExecutingRunFunctions ()

  Start processing pipelines on the Sink.

7.9.1 Detailed Description

Definition in file COIPipeline_sink.h.

7.10 COIPipeline_source.h File Reference

Defines

- #define COI_PIPELINE_MAX_IN_BUFFERS 16384
- #define COI_PIPELINE_MAX_IN_MISC_DATA_LEN 32768
- #define COI_PIPELINE_MAX_PIPELINES 512

Typedefs

- typedef enum COI_ACCESS_FLAGS COI_ACCESS_FLAGS

  These flags specify how a buffer will be used within a run function.

Enumerations

- enum COI_ACCESS_FLAGS {
  COI_SINK_READ = 1,
  COI_SINK_WRITE,
  COI_SINK_WRITE_ENTIRE,
  COI_SINK_READ_ADDREF,
  COI_SINK_WRITE_ADDREF,
  COI_SINK_WRITE_ENTIRE_ADDREF }

  These flags specify how a buffer will be used within a run function.

Functions

- COIACCESSAPI COIRESULT COIPipelineClearCPUMask (COI_CPU_MASK *in_Mask)

  Clears a given mask.

- COIACCESSAPI COIRESULT COIPipelineCreate (COIPROCESS in_Process, COI_CPU_MASK in_Mask, uint32_t in_StackSize, COIPIPELINE *out_pPipeline)

  Create a pipeline assoiated with a remote process.

- COIACCESSAPI COIRESULT COIPipelineDestroy (COIPIPELINE in_Pipeline)

  Destroys the inidicated pipeline, releasing its resources.

- COIACCESSAPI COIRESULT COIPipelineGetEngine (COIPIPELINE in_Pipeline, COIENGINE *out_pEngine)

  Retrieve the engine that the pipeline is associated with.

- COIACCESSAPI COIRESULT COIPipelineRunFunction (COIPIPELINE in_Pipeline, COIFUNCTION in_Function, uint32_t in_NumBuffers, const COIBUFFER *in_pBuffers, const COI_ACCESS_FLAGS *in_pBufferAccessFlags, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, const void *in_pMiscData, uint16_t in_MiscDataLen, void *out_pAsyncReturnValue, uint16_t in_AsyncReturnValueLen, COIEVENT *out_pCompletion)

  Enqueues a function in the remote process binary to be executed.

- COIACCESSAPI COIRESULT COIPipelineSetCPUMask (COIPROCESS in_Process, uint32_t in_CoreID, uint8_t in_ThreadID, COI_CPU_MASK *out_pMask)

  Add a particular core:thread pair to a COI_CPU_MASK.

7.10.1 Detailed Description

Definition in file COIPipeline_source.h.

7.11 COIProcess_sink.h File Reference

Functions

- COIRESULT COIProcessProxyFlush ()

  This call will block until all stdout and stderr output has been proxied to and written by the source.

- COIRESULT COIProcessWaitForShutdown ()

This call will block while waiting for the source to send a process destroy message.

7.11.1 Detailed Description

Definition in file COIProcess_sink.h.

7.12 COIProcess_source.h File Reference

Defines

- #define COI_FAT_BINARY ((uint64_t)-1)

  This is a flag for COIProcessCreateFromMemory that indicates the passed in memory pointer is a fat binary file and should not have regular validation.

- #define COI_MAX_FILE_NAME_LENGTH 256
- #define COI_MAX_FUNCTION_NAME_LENGTH 256
- #define COI_PROCESS_SOURCE ((COIPROCESS)-1)

  This is a special COIPROCESS handle that can be used to indicate that the source process should be used for an operation.

COIProcessSetCacheSize flags.

Flags are divided into two categories: _MODE_ and _ACTION_ only one of each is valid with each call.

ACTIONS and _MODES_ should be bitwised OR'ed together, i.e. |

- #define COI_CACHE_MODE_MASK 0x00000007

  Current set of DEFINED bits for _MODE_, can be used to clear or check fields, not useful to pass into APIs.

- #define COI_CACHE_MODE_NOCHANGE 0x00000001

  Flag to indicate to keep the previous mode of operation.

- #define COI_CACHE_MODE_ONDEMAND_SYNC 0x00000002

  Mode of operation that indicates that COI will allocate physical cache memory exactly when it is is needed.

- #define COI_CACHE_MODE_ONDEMAND_ASYNC 0x00000004

  Not yet implemented.

- #define COI_CACHE_ACTION_MASK 0x00070000

  Current set of DEFINED bits for _ACTION_ can be used to clear fields, but not usefil to pass into API's.

- #define COI_CACHE_ACTION_NONE 0x00010000

  No action requested.

- #define COI_CACHE_ACTION_GROW_NOW 0x00020000

  This _ACTION_ flag will immediately attempt to increase the cache physical memory size to the current set pool size(s).

- #define COI_CACHE_ACTION_FREE_UNUSED 0x00040000

  Not yet implemented.

Typedefs

- typedef void(* COI_NOTIFICATION_CALLBACK)(COI_NOTIFICATIONS in_Type, COIPROCESS in_Process, COIEVENT in_Event, const void *in_UserData)

  A callback that will be invoked to notify the user of an internal Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) event.

- typedef enum COI_NOTIFICATIONS COI_NOTIFICATIONS

  The user can choose to have notifications for these internal events so that they can build their own profiling and performance layer on top of Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI).

Enumerations

- enum COI_NOTIFICATIONS {
  RUN_FUNCTION_READY = 0,
  RUN_FUNCTION_START,
  RUN_FUNCTION_COMPLETE,
  BUFFER_OPERATION_READY,
  BUFFER_OPERATION_COMPLETE,
  USER_EVENT_SIGNALED }

  The user can choose to have notifications for these internal events so that they can build their own profiling and performance layer on top of Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI).

Functions

- __asm__ (".symver COIProcessLoadLibraryFromMemory,""COIProcessLoadLibraryFromMemory@COI_1.0")
- __asm__ (".symver COIProcessLoadLibraryFromFile,""COIProcessLoadLibraryFromFile@COI_1.0")
- COIACCESSAPI void COINotificationCallbackSetContext (const void *in_UserData)

  Set the user data that will be returned in the notification callback.

- COIACCESSAPI COIRESULT COIProcessCreateFromFile (COIENGINE in_Engine, const char *in_pBinaryName, int in_Argc, const char **in_ppArgv, uint8_t in_DupEnv, const char **in_ppAdditionalEnv, uint8_t in_ProxyActive, const char *in_Reserved, uint64_t in_InitialBufferSpace, const char *in_LibrarySearchPath, COIPROCESS *out_pProcess)

  Create a remote process on the Sink and start executing its main() function.

- COIACCESSAPI COIRESULT COIProcessCreateFromMemory (COIENGINE in_Engine, const char *in_pBinaryName, const void *in_pBinaryBuffer, uint64_t in_BinaryBufferLength, int in_Argc, const char **in_ppArgv, uint8_t in_DupEnv, const char **in_ppAdditionalEnv, uint8_t in_ProxyActive, const char *in_Reserved, uint64_t in_InitialBufferSpace, const char *in_LibrarySearchPath, const char *in_FileOfOrigin, uint64_t in_FileOfOriginOffset, COIPROCESS *out_pProcess)

  Create a remote process on the Sink and start executing its main() function.

- COIACCESSAPI COIRESULT COIProcessDestroy (COIPROCESS in_Process, int32_t in_WaitForMainTimeout, uint8_t in_ForceDestroy, int8_t *out_pProcessReturn, uint32_t *out_pTerminationCode)

  Destroys the indicated process, releasing its resources.

- COIACCESSAPI COIRESULT COIProcessGetFunctionHandles (COIPROCESS in_Process, uint32_t in_NumFunctions, const char **in_ppFunctionNameArray, COIFUNCTION *out_pFunctionHandleArray)

  Given a loaded native process, gets an array of function handles that can be used to schedule run functions on a pipeline associated with that process.

- COIRESULT COIProcessLoadLibraryFromFile (COIPROCESS in_Process, const char *in_pFileName, const char *in_pLibraryName, const char *in_LibrarySearchPath, COILIBRARY *out_pLibrary)

  Loads a shared library into the specified remote process, akin to using dlopen() on a local process in Linux or LoadLibrary() in Windows.

- COIRESULT COIProcessLoadLibraryFromMemory (COIPROCESS in_Process, const void *in_pLibraryBuffer, uint64_t in_LibraryBufferLength, const char *in_pLibraryName, const char *in_LibrarySearchPath, const char *in_FileOfOrigin, uint64_t in_FileOfOriginOffset, COILIBRARY *out_pLibrary)

  Loads a shared library into the specified remote process, akin to using dlopen() on a local process in Linux or LoadLibrary() in Windows.

- COIACCESSAPI COIRESULT COIProcessRegisterLibraries (uint32_t in_NumLibraries, const void **in_ppLibraryArray, const uint64_t *in_pLibrarySizeArray, const char **in_ppFileOfOriginArray, const uint64_t *in_pFileOfOriginOffSetArray)

  Registers shared libraries that are already in the host process's memory to be used during the shared library dependency resolution steps that take place during subsequent calls to COIProcessCreate and COIProcessLoadLibrary*.*

- COIACCESSAPI COIRESULT COIProcessSetCacheSize (const COIPROCESS in_Process, const uint64_t in_HugePagePoolSize, const uint32_t in_HugeFlags, const uint64_t in_SmallPagePoolSize, const uint32_t in_SmallFlags, uint32_t in_NumDependencies, const COIEVENT *in_pDependencies, COIEVENT *out_pCompletion)

  Set the minimum preferred COIProcess cache size.

- COIACCESSAPI COIRESULT COIProcessUnloadLibrary (COIPROCESS in_Process, COILIBRARY in_Library)

  Unloads a a previously loaded shared library from the specified remote process.

- COIACCESSAPI COIRESULT COIRegisterNotificationCallback (COIPROCESS in_Process, COI_NOTIFICATION_CALLBACK in_Callback, const void *in_UserData)

  Register a callback to be invoked to notify that an internal Intel(R) Coprocessor Offload Infrastructure (Intel(R) COI) event has occured on the process that is associated with the callback.

- COIACCESSAPI COIRESULT COIUnregisterNotificationCallback (COIPROCESS in_Process, COI_NOTIFICATION_CALLBACK in_Callback)

  Unregisters a callback, notifications will no longer be signaled.

7.12.1 Detailed Description

Definition in file COIProcess_source.h.

7.13 COIResult_common.h File Reference

Typedefs

- typedef enum COIRESULT COIRESULT

Enumerations

- enum COIRESULT {
  COI_SUCCESS = 0,
  COI_ERROR,
  COI_NOT_INITIALIZED,
  COI_ALREADY_INITIALIZED,
  COI_ALREADY_EXISTS,
  COI_DOES_NOT_EXIST,
  COI_INVALID_POINTER,
  COI_OUT_OF_RANGE,

COI_NOT_SUPPORTED,
COI_TIME_OUT_REACHED,
COI_MEMORY_OVERLAP,
COI_ARGUMENT_MISMATCH,
COI_SIZE_MISMATCH,
COI_OUT_OF_MEMORY,
COI_INVALID_HANDLE,
COI_RETRY,
COI_RESOURCE_EXHAUSTED,
COI_ALREADY_LOCKED,
COI_NOT_LOCKED,
COI_MISSING_DEPENDENCY,
COI_UNDEFINED_SYMBOL,
COI_PENDING,
COI_BINARY_AND_HARDWARE_MISMATCH,
COI_PROCESS_DIED,
COI_INVALID_FILE,
COI_EVENT_CANCELED,
COI_VERSION_MISMATCH,
COI_BAD_PORT,
COI_AUTHENTICATION_FAILURE,
COI_NUM_RESULTS }

Functions

- COIACCESSAPI const char * COIResultGetName (COIRESULT in_ResultCode)

  Returns the string version of the passed in COIRESULT.

Variables

- *If you see an error on this line

7.13.1 Variable Documentation

7.13.1.1 * If you see an error on this line

Definition at line 272 of file COIResult_common.h.

7.14 COISysInfo_common.h File Reference

This interface allows developers to query the platform for system level information.

Defines

- #define INITIAL_APIC_ID_BITS 0xFF000000

Functions

- COIACCESSAPI uint32_t COISysGetAPICID (void)
- COIACCESSAPI uint32_t COISysGetCoreCount (void)
- COIACCESSAPI uint32_t COISysGetCoreIndex (void)
- COIACCESSAPI uint32_t COISysGetHardwareThreadCount (void)
- COIACCESSAPI uint32_t COISysGetHardwareThreadIndex (void)
- COIACCESSAPI uint32_t COISysGetL2CacheCount (void)
- COIACCESSAPI uint32_t COISysGetL2CacheIndex (void)

7.14.1 Detailed Description

This interface allows developers to query the platform for system level information.

Definition in file COISysInfo_common.h.

7.15 COITypes_common.h File Reference

Data Structures

- struct coievent

Typedefs

- typedef uint64_t COI_CPU_MASK [16]
- typedef wchar_t coi_wchar_t

  On Windows, coi_wchar_t is a uint32_t.

- typedef struct coibuffer * COIBUFFER
- typedef struct coiengine * COIENGINE
- typedef struct coievent COIEVENT
- typedef struct coifunction * COIFUNCTION
- typedef struct coilibrary * COILIBRARY
- typedef struct coimapinst * COIMAPINSTANCE
- typedef struct coipipeline * COIPIPELINE
- typedef struct coiprocess * COIPROCESS

7.15.1 Detailed Description

Definition in file COITypes_common.h.

Index