heteroStreams 0.9 Programming Guide and API Reference

Intel® Manycore Platform Software Stack 3.6

Legal Disclaimer

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm

Intel and the Intel logo, are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Contents

Chapter 1

Introduction

1.1 Motivation

The versatility and ease of use of Intel® Xeon Phi™ coprocessor family continues to underline the importance and relevance of the offload programming paradigm. This paradigm pairs very well with another programming model, a special form of task-based parallelism, namely the *stream processing* pattern.

Although users of the Intel® Xeon Phi™ could write streaming-based applications using the Intel® Coprocessor Offload Infrastructure bundled with Intel® Manycore Platform Software Stack, the applications were required to implement a considerable amount of plumbing.

heteroStreams (hStreams for short) aims to relieve the developers of the burden of implementing the necessary infrastructure for applications wishing to leverage the streaming programming paradigm on heterogeneous platforms themselves by exposing a concise programming model through a well-defined API available in form of a shared library.

Chapter 2

Programming Overview

2.1 Source and sink

hStreams uses two special terms, *source* and *sink* to mark the difference between the place where the work is produced, i.e. the source and the place where the work is executed, i.e. the sink. For example, in a computer with an Intel® Xeon Phi™coprocessor attached to its PCIe bus a source would be a process running on the host CPU while a sink would be located on the coprocessor.

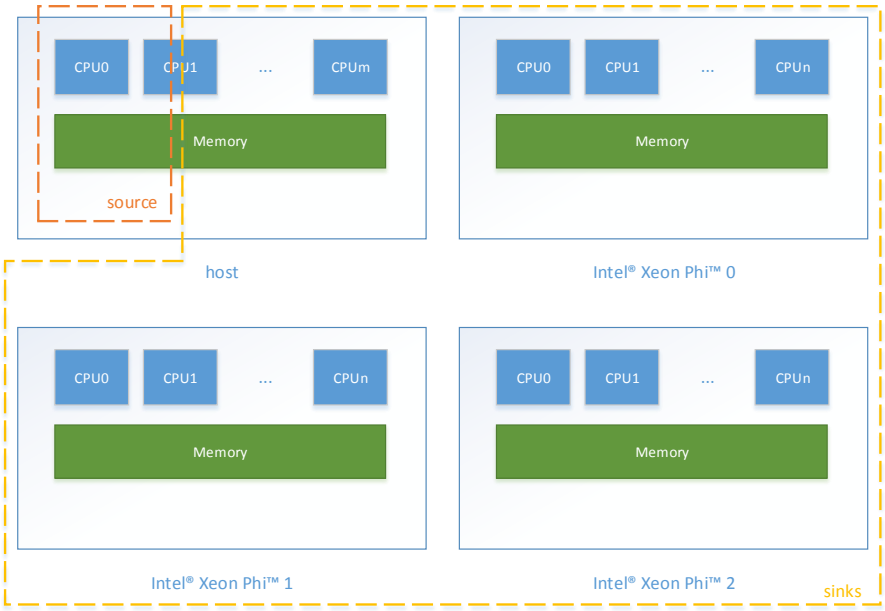

Figure 2.1: An illustration of the *source* and *sink* concepts with a host and three Intel® Xeon Phi™ coprocessors.

2.2 API levels

The hStreams library exposes two levels of API - the *app* API and the *core* API. app API is designed to allow a novice user to quickly start writing programs using the library and offers only a subset of the full functionality of the hStreams library. Moreover, it contains several helper functions, common building blocks which further help boosting the productivity of a beginning user. The core API – on the other hand – exposes the full functionality available in the hStreams library and is targeted at a more advanced user.

It is worth noting that both of those API levels are interoperable – entities created through the app API can be manipulated and used by the core API and *vice-versa*.

2.3 A getting started example

Without getting into too much detail about the library, let us now consider a minimal working example for successfully running a compute command on an Intel® Xeon Phi™, making the use of hStreams through its app API. Listing 2.1 shows an example source-side application written in C++. Listing 2.2 on the other hand shows the corresponding sink-side library which contains a user-defined function – `hello_world` – which is to be invoked.

```cpp
// Main header for app API (source)
#include <hStreams_app_api.h>

int main() {
    uint64_t arg = 3735928559;
    // Create domains and streams
    hStreams_app_init(1,1);
    // Enqueue a computation in stream 0
    hStreams_app_invoke(0, "hello_world",
        1, 0, &arg, NULL, NULL, 0);
    // Finalize the library. Implicitly
    // waits for the completion of
    // enqueued actions
    hStreams_app_fini();
    return 0;
}
```

Listing 2.1: example_src.cpp

```cpp
// Main header for sink API
#include <hStreams_sink.h>
// for printf()
#include <stdio.h>

// Ensure proper name mangling and symbol
// visibility of the user function to be
// invoked on the sink.
HSTREAMS_EXPORT
void hello_world(uint64_t arg)
{
    // This printf will be visible
    // on the host. arg will have
    // the value assigned on the source
    printf("Hello world, %x\n", arg);
}
```

Listing 2.2: example_sink.cpp

In order to run the example, one must set the environment variable `SINK_LD_LIBRARY_PATH` so that hStreams runtime can pick up the appropriate libraries. Listing 2.3 shows a record of a session on the Linux operating system in which both the source and the sink components of the example are compiled, the environment variable is set and finally, the application is executed. Listing 2.4 shows a corresponding session on the Windows operating system.

```
$ # setup Intel compiler variables
$ (...)compilervars.sh intel64
$ icc example_src.cpp -lhstreams_source -o example
$ icc -fPIC -mmic -shared example_sink.cpp -o example_mic.so
$ export SINK_LD_LIBRARY_PATH=/opt/mpss/3.6/sysroots/k1om-mpss-linux/usr/lib64/:\
> $MIC_LD_LIBRARY_PATH:\
> $(pwd)
$ ./example
Hello world, deadbeef
```

Listing 2.3: Compiling and running the example on Linux

```
> rem setup Intel compiler variables
> (...) compilervars.bat intel64
> icl /Qmic -fPIC -shared -o example_mic.so example_sink.cpp
> icl /I"%INTEL_MPSS_HOST_SDK%\include" example_src.cpp /Feexample.exe /link /libpath:"%
    INTEL_MPSS_HOST_SDK%\lib" hstreams_source.lib
> set SINK_LD_LIBRARY_PATH=%cd%;%INTEL_MPSS_HOST_SDK%\..\k1om-mpss-linux\usr\lib64;%
    MIC_LD_LIBRARY_PATH%;
> example.exe
Hello world, deadbeef
```

Listing 2.4: Compiling and running the example on Windows

2.4 Logging and traces in hetero-streams

To aid the development and debugging process of programs which leverage hetero-streams, the library can emit detailed debug output thanks to its internal logging mechanisms. There are two dimensions in which the output of the library can be controlled:

- verbosity
- category of the information

The *verbosity* dimension is controlled through the `hStreams_Cfg_SetLogLevel` API. Using this function, user can instruct hetero-streams to never emit any logs or produce messages *up to* a specific level (refer to `HSTR_LOG_LEVEL_VALUES`, e.g. `HSTR_LOG_LEVEL_NO_LOGGING` or `HSTR_LOG_LEVEL_DEBUG4`).

The other mechanism that hetero-streams employs to further aid the debugging is the *categorisation* of messages which it produces. And so, each message which is produced ongs to a specific category, e.g. synchronization events are described by `HSTR_INFO_TYPE_SYNC` and messages related to memory actions (allocations, deallocations, ...) are handled by `HSTR_INFO_TYPE_MEM`.

Whether messages belonging to a particular category are printed or not is controlled by a *message filter* which can be adjusted by use of the `hStreams_Cfg_SetLogLevel` function. This function accepts a bitmask that determines which messages are to be shown and which are to be dropped. The values for modifying this bitmask are described by the values of the `HSTR_INFO_TYPE_VALUES` enumerated type.

2.5 Infrastructure

2.5.1 hStreams files

hStreams exposes its functionality through dynamic shared objects that user applications can load and then call functions from that object. The binary artifacts provided with hStreams are shown in Table 2.1 for Linux operating system and in Table 2.2 for Windows operating system.

The function entry points and types are declared in the header files listed in Table 2.3.

Binary file name	Description
libhstreams_source.so	A dynamic shared library providing „source” functionality
libhstreams_mic	hStreams runtime, a startup file for Intel® Xeon Phi™ x100 family.

Table 2.1: Binary artifacts of the hStreams library on Linux operating system.

Binary file name	Description
hstreams_source.dll	A dynamic-link library providing „source” functionality
libhstreams_mic	hStreams runtime, a startup file for Intel® Xeon Phi™ x100 family.

Table 2.2: Binary artifacts of the hStreams library on Windows operating system.

2.6 Common issues with setting up the environment

This section attempts to capture most common issues users may encounter with an improper setup of the environment in which they attempt to execute a binary making use of hStreams. To illustrate the possible failures, the example from Section 2.3 will be used.

If the SINK_LD_LIBRARY_PATH variable is not set at all, the following error will be presented:

```
$ unset SINK_LD_LIBRARY_PATH
$ ./example
hStreams_Init_worker: returns: HSTR_RESULT_DEVICE_NOT_INITIALIZED, Did not find
    libhstreams_mic on host.
```

Listing 2.5: Running the example without SINK_LD_LIBRARY_PATH

In the next example, the SINK_LD_LIBRARY_PATH variable contains only the location of the libhstreams_mic binary.

```
$ export SINK_LD_LIBRARY_PATH=/opt/mpss/3.6/sysroots/k1om-mpss-linux/usr/lib64/
$ ./example
The remote process indicated that the following libraries could not be loaded:
    libmkl_intel_lp64.so libmkl_intel_thread.so libmkl_core.so libiomp5.so libimf.so libsvml
    .so libirng.so libintlc.so.5
hStreams_Init_worker: returns: HSTR_RESULT_REMOTE_ERROR, Could not create process on the
    device: COI_MISSING_DEPENDENCY.
```

Listing 2.6: Running the example with incomplete SINK_LD_LIBRARY_PATH

Finally, let us see an example in which all hStreams-required paths are set properly but the library cannot find user's custom kernels library

```
$ export SINK_LD_LIBRARY_PATH=/opt/mpss/3.6/sysroots/k1om-mpss-linux/usr/lib64/:$MKLROOT/lib
    /mic/:$MKLROOT/../compiler/lib/mic
$ ./example
enqueueFunction: returns: HSTR_RESULT_BAD_NAME, hStreams_fetchSinkFuncAddress: returns:
    HSTR_RESULT_BAD_NAME, dlsym() returned error: /tmp/coi_procs/1/99657/libhstreams_mic:
    undefined symbol: hello_world, for function: hello_world.
A sink-compiled version of called function hello_world not found on logical domain 1,
    physical domain 0.
hStreams_app_invoke2: returns: HSTR_RESULT_BAD_NAME, hStreams_EnqueueCompute( in_LogStreamID
    , in_pFuncName, in_NumScalarArgs, in_NumHeapArgs, in_pArgs, out_pEvent, out_pReturnValue
    , in_ReturnValueSize) returned HSTR_RESULT_BAD_NAME
```

Listing 2.7: Running the example with user's custom kernels library location missing from SINK_LD_LIBRARY_PATH

Header file name	Description
hStreams_app_api.h	Declarations for source-side app API
hStreams_common.h	Declarations which are common to both sink and source
hStreams_sink.h	Sink-side declarations, e.g. HSTREAMS_EXPORT
hStreams_source.h	Declarations for source-side core API
hStreams_types.h	Declarations of types used in APIs, both source and sink
hStreams_version.h	Preprocessor definitions for the hStreams library version

Table 2.3: Header files of the hStreams library.

2.7 Execution Model

2.7.1 Streams

The basic building block of the execution model is a *s*tream which has two endpoints – one on the *source* (where actions are placed into the stream and another one on the *sink*) where the actions are executed. The sink endpoint of a stream is defined by specifying a subset of available computing resources (hardware threads) on which the actions may execute. As they are processed, the actions to be executed resemble a *first-in first-out* queue on the sink with the source endpoint of the stream pushing the actions into the queue and the sink endpoint popping them out.

All the actions which can be placed into a stream fall into one of three categories:

- Compute actions
- Memory movement actions
- Wait/synchronization actions

Compute actions are executions of functions queued up in a stream. A compute action consists of the name of the function to be invoked, the parameters to be copied by value for remote invocation, the parameters which belong to pre-registered buffers and the output parameters of the function.

Memory movement actions are transfers of memory contents of pre-registered buffers between arbitrary endpoints of any two streams.

Wait and synchronization actions involve the sink endpoint of a stream waiting on a collection of events.

2.7.2 Logical and physical entities

2.7.2.1 Domains and streams

hStreams introduces an abstraction of the physical resources through the existence of *logical* domains, streams and buffers. To explain the purpose and nature of those, one must introduce a few concepts - *physical* streams, domains and buffers.

A *physical domain* is a representation of all resources within one memory coherence domain - memory and processors. *Physical streams* are entities consisting of threads in the *physical domain*

bounded by a prescribed affinity mask restricted to fully contained inside the domain's processor set.

A *logical domain* is a subset of a given physical domain, defined by its processor mask which must be contained within the owning physical domain's processor set. One or more logical domains may belong to the same physical domain. Additionally, two logical domains are prohibited from partially overlapping. Two logical domains can, however, fully overlap (i.e. be described by identical processor masks), in which case they are considered to be distinct, unrelated entities.

Next, a *logical stream* is an abstraction of a stream which resides in one given logical domain. As with domains, a logical stream is defined by its hardware thread mask. Multiple logical streams are allowed to be partially or fully overlapping. Moreover, logical streams described by identical processor masks are considered to be aliasing the same underlying physical stream. There is no correspondence assumed between logical streams with partially overlapping CPU masks, each of them is mapped to a different physical stream.

Physical streams are entities which are not enumerable via the interface of the hStreams library. They are implicitly created and destroyed when logical streams are created and destroyed.

To better understand these concepts, please refer to Figure 2.2. In this picture you can see how

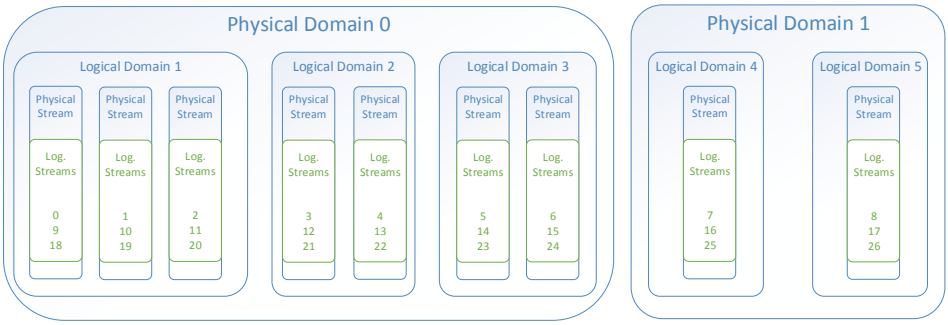

Figure 2.2: An example arrangement of physical domains, logical domains, physical streams and logical streams.

there are three logical domains on physical domain #0 with 21 streams spread across them. In logical domain #2 there are six logical streams constructed with the special property that e.g. streams #3, #12 and #21 have been constructed with exactly overlapping CPU masks and thus alias the same *physical* stream. The physical streams are not given any IDs in the picture to underline the fact that they are not enumerable through the interface of the hStreams library.

2.7.2.2 Buffers

In order to operate on memory resources (e.g. transfer data to and from remote domains or to supply a remote memory address as an operand of a compute action), users must make hStreams aware of such memory resources. In order to do that, user must create a *logical buffer* through a call to the appropriate API supplying the start address and the length of that buffer. Once the hStreams runtime is aware of the buffer, a pointer to a memory location anywhere *inside* of that buffer is recognized as a handle and can be used for performing data transfers or compute actions involving that buffer.

A logical buffer created by the user may have instantiations in many logical domains beside the source. Those instantiations of the buffer are called *physical buffers*. A logical buffer must have a corresponding physical buffer on the logical domain where it is intended to be used (either as an operand of a data transfer or a compute action). E.g. if a buffer is supplied as an argument to a compute action

placed into a stream the sink endpoint of which is located in logical domain #4, then the logical buffer must be already *instantiated* in logical domain #4.

Please note that the contents of multiple buffer instantiations are not synchronized automatically, i.e. if a buffer is instantiated for a logical domain #12, the contents of that instantiation are undefined until they are determined explicitly by the user either writing to the instantiation from a compute action or transferring data to that instantiation.

Chapter 3

Deprecated List

Global HSTR_DEPRECATED("hStreams_SetVerbose() has been deprecated. ""Please refer to hStreams_Cfg_Se
This function has been deprecated in favor of hStreams_Cfg_SetLogLevel() and hStreams_-Cfg_SetLogInfoType().

Global HSTR_DEPRECATED("hStreams_GetVerbose() has been deprecated. ""Please refer to hStreams_Cfg_Se
This function has been deprecated in favor of hStreams_Cfg_SetLogLevel() and hStreams_-Cfg_SetLogInfoType().

Global HSTR_INFO_TYPE_ALWAYSEMIT HSTR_INFO_TYPE_ALWAYSEMIT has been deprecated

Global HSTR_INFO_TYPE_AND16 HSTR_INFO_TYPE_AND16 has been deprecated

Global HSTR_INFO_TYPE_DEPS HSTR_INFO_TYPE_DEPS has been deprecated

Global HSTR_INFO_TYPE_GENERAL HSTR_INFO_TYPE_GENERAL has been deprecated

Global HSTR_INFO_TYPE_INVOKE HSTR_INFO_TYPE_INVOKE has been deprecated

Global HSTR_OPTIONS::HSTR_DEPRECATED("HSTR_OPTIONS::_hStreams_EmitMessage has been deprecate
_hStreams_EmitMessage has been deprecated in favor of a new logging mechanism. For details, consult the documentation of `hStreams_Cfg_SetLogLevel()` and `hStreams_-Cfg_SetLogInfoType()`.

Global HSTR_OPTIONS::HSTR_DEPRECATED("HSTR_OPTIONS::verbose has been deprecated. ""Please refer t
This option has been deprecated in favor of `hStreams_Cfg_SetLogLevel()` and `hStreams_Cfg_SetLogInfoType()`.

Global HSTR_SEVERITY_ERROR HSTR_SEVERITY_ERROR has been deprecated

Global HSTR_SEVERITY_FATAL_ERROR HSTR_SEVERITY_FATAL_ERROR has been deprecated

Global HSTR_SEVERITY_INFO HSTR_SEVERITY_INFO has been deprecated

Global HSTR_SEVERITY_WARNING HSTR_SEVERITY_WARNING has been deprecated

Chapter 4

Module Index

4.1 Modules

Here is a list of all modules:

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Module Documentation

7.1 app API (source)

Modules

- Wrapped and simplified core functions
- Common building blocks

7.1.1 Detailed Description

The app API is set of simplified functions covering only a subset of hStreams' functionality. It is designed to help an inexperienced hStreams user quickly start developing their own applications using the library.

Apart from the library's core functionality, several common building blocks are offered to further help new users quickly develop their code.

7.2 Wrapped and simplified core functions

Functions

- HSTR_RESULT hStreams_app_init_in_version (uint32_t in_StreamsPerDomain, uint32_t in_LogStreamOversubscription, const char *interface_version)

  Initialize hStreams homogenously across all available Intel(R) Xeon Phi(TM) coprocessors.

- HSTR_RESULT hStreams_app_init_domains_in_version (uint32_t in_NumLogDomains, uint32_t *in_pStreamsPerDomain, uint32_t in_LogStreamOversubscription, const char *interface_version)

  Initialize hStreams state, allowing for non-heterogeneity and more control then `hStreams_app_init()`.

- HSTR_RESULT hStreams_app_fini ()

  Finalization of hStreams state.

- HSTR_RESULT hStreams_app_create_buf (void *in_BufAddr, const uint64_t in_NumBytes)

  Allocate 1-dimensional buffer on each currently existing logical domains.

- HSTR_RESULT hStreams_app_invoke (HSTR_LOG_STR in_LogStreamID, const char *in_pFuncName, uint32_t in_NumScalarArgs, uint32_t in_NumHeapArgs, uint64_t *in_pArgs, HSTR_EVENT *out_pEvent, void *out_pReturnValue, uint16_t in_ReturnValueSize)

  Enqueue an execution of a user-defined function in a stream.

- HSTR_RESULT hStreams_app_stream_sync (HSTR_LOG_STR in_LogStreamID)

  Block until all the operation enqueued in a stream have completed.

- HSTR_RESULT hStreams_app_thread_sync ()

  Block until all the operation enqueued in all the streams have completed.

- HSTR_RESULT hStreams_app_event_wait (uint32_t in_NumEvents, HSTR_EVENT *in_pEvents)

  Wait on a set of events.

- HSTR_RESULT hStreams_app_event_wait_in_stream (HSTR_LOG_STR in_LogStreamID, uint32_t in_NumEvents, HSTR_EVENT *in_pEvents, int32_t in_NumAddresses, void **in_pAddresses, HSTR_EVENT *out_pEvent)

  Aggregate multiple dependences into one event handle and optionally insert that event handle into a logical stream.

- HSTR_RESULT hStreams_Alloc1D (void *in_BaseAddress, uint64_t in_size)

  Allocate 1-dimensional buffer on each currently existing logical domains.

7.2.1 Function Documentation

7.2.1.1 HSTR_RESULT hStreams_Alloc1D (void * *in_BaseAddress*, uint64_t *in_size*)

Allocate 1-dimensional buffer on each currently existing logical domains.

Construct an hStreams buffer out of user-provided memory. This function creates an instantiation of the buffer in all of the currently present logical domains. Note that the contents of those instatiations are considered to be undefined, i.e. the contents of the buffer are not implicitly synchronized across all the instantations.

A buffer constructed by a call to `hStreams_Alloc1D()` may be later supplied as an operand to memory transfer or a compute action. In order to do that, user should supply a valid address falling anywhere *inside* the buffer.

Parameters

in_BufAddr [in] pointer to the beginning of the memory in the source logical domain

in_NumBytes [in] size of the memory to create the buffer for, in bytes

Returns

If successful, `hStreams_Alloc1D()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized prior to this call.
- `HSTR_RESULT_NULL_PTR` if `in_BaseAddress` is `NULL`
- `HSTR_RESULT_OUT_OF_RANGE` if `in_NumBytes == 0`
- `HSTR_RESULT_OUT_OF_MEMORY` if there's not enough memory on one or more domains and therefore the buffer cannot be instantiatied

Thread safety:

Thread safe.

7.2.1.2 HSTR_RESULT hStreams_app_create_buf (void * *in_BufAddr*, const uint64_t *in_NumBytes*)

Allocate 1-dimensional buffer on each currently existing logical domains.

Construct an hStreams buffer out of user-provided memory. This function creates an instantiation of the buffer in all of the currently present logical domains. Note that the contents of those instatiations are considered to be undefined, i.e. the contents of the buffer are not implicitly synchronized across all the instantations.

A buffer constructed by a call to `hStreams_app_create_buf()` may be later supplied as an operand to memory transfer or a compute action. In order to do that, user should supply a valid address falling anywhere *inside* the buffer.

Parameters

in_BufAddr [in] pointer to the beginning of the memory in the source logical domain

in_NumBytes [in] size of the memory to create the buffer for, in bytes

Returns

If successful, `hStreams_app_create_buf()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized prior to this call.
- `HSTR_RESULT_NULL_PTR` if `in_BaseAddress` is `NULL`
- `HSTR_RESULT_OUT_OF_RANGE` if `in_NumBytes == 0`
- `HSTR_RESULT_OUT_OF_MEMORY` if there's not enough memory on one or more domains and therefore the buffer cannot be instantiatied

Thread safety:

Thread safe.

7.2.1.3 HSTR_RESULT hStreams_app_event_wait (uint32_t *in_NumEvents*, HSTR_EVENT * *in_pEvents*)

Wait on a set of events.

Synchronization:

- Every action - data transfer or remote compute - yields a sync event
- Those sync events can be waited on within a given logical stream, with event_wait and event_wait_in_stream.

Parameters

in_NumEvents [in] number of event pointers in the array

in_pEvents [in] array of pointers of events to be waited on

Returns

If successful, `hSterams_app_event_wait()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns on of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.
- `HSTR_RESULT_NULL_PTR` if in_pEvents is NULL
- `HSTR_RESULT_REMOTE_ERROR` if there was a remote error, e.g. the remote process died
- `HSTR_RESULT_TIME_OUT_REACHED` if the time out was reached or the timeout is zero and the event has not been signalled.
- `HSTR_RESULT_EVENT_CANCELED` if the event was cancelled or the process died
- `HSTR_RESULT_OUT_OF_RANGE` if in_NumEvents == 0

Thread safety:

Thread safe.

7.2.1.4 HSTR_RESULT hStreams_app_event_wait_in_stream (HSTR_LOG_STR *in_LogStreamID*, uint32_t *in_NumEvents*, HSTR_EVENT * *in_pEvents*, int32_t *in_NumAddresses*, void ** *in_pAddresses*, HSTR_EVENT * *out_pEvent*)

Aggregate multiple dependences into one event handle and optionally insert that event handle into a logical stream.

The resulting aggregation of multiple dependences is composed of depenences on:

- last actions actions enqueued in the stream which involve buffers specified in the `in_pAddresses` array
- explicit completion events specified in the in_pEvents array The output event handle, `out_pEvent` will represent that aggregation of dependences.

The special pair of values `(-1, NULL)` for `(in_NumAddresses, in_pAddresses)` is used to signify that the dependence to be created should not be inserted into the stream, only an event handle representing that dependence should be returned.

The special pair of values `(0, NULL)` for `(in_NumAddresses, in_pAddresses)` is used to signify that the dependence to be created should not involve any actions related to buffer.

On the other hand, the special pair of values `(0, NULL)` for `(in_NumEvents, in_pEvents)` is used to signify that the dependence to be inserted should include all the actions previously enqueued in the specified stream.

Parameters

in_LogStreamID [in] ID of the logical stream from which take the dependencies and into which to optionally insert the dependence.

in_NumEvents [in] Number of entries in the `in_pEvents` array.

in_pEvents [in] An array of event handles that should be included in the aggregated dependence.

in_NumAddresses [in] Number of entries in the in_pAddresses array.

in_pAddresses [in] Array of source-side proxy addresses to be mapped to buffers, on which the dependencies will be computed for aggregation.

out_pEvent [out] the aggreagated completion event. If no handle is needed, set this to NULL

Returns

If successful, `hStreams_app_event_wait_in_stream()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.
- `HSTR_RESULT_NOT_FOUND` if a logical stream with ID `in_LogStreamID` doesn't exist
- `HSTR_RESULT_NOT_FOUND` if at least one entry in the `in_pAddresses` array does not correspond to a buffer with an instantiation in the logical domain in which the `in_LogStreamID` stream is located is located.
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_NumAddresses` is 0 or -1 and `in_pAddresses` is not NULL and vice versa.
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_NumEvents` is 0 and `in_pEvents` is not NULL and vice versa.

Thread safety:

All actions enqueued through concurrent calls to `hStreams_app_event_wait_in_stream()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.2.1.5 HSTR_RESULT hStreams_app_fini ()

Finalization of hStreams state.

Destroys hStreams internal structures and clears the state of the library. All logical domains, streams and buffers are destroyed as a result of this call.

Returns

If successful, `hStreams_app_fini()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized prior to this call.

Thread safety:

Thread safe.

7.2.1.6 HSTR_RESULT hStreams_app_init_domains_in_version (uint32_t *in_NumLogDomains*, uint32_t * *in_pStreamsPerDomain*, uint32_t *in_LogStreamOversubscription*, const char * *interface_version*)

Initialize hStreams state, allowing for non-heterogeneity and more control then `hStreams_app_init()`.

If no initialization through `hStreams_app_init*` routines had been performed beforehand, this function will detect all available Intel(R) Xeon and attempt to spread an indicated number of logical domain across all of them. In i-th logical domain, `in_StreamsPerDomain[i]` sets of streams will be created, each of them consisting of `in_LogStreamOversubscription` exactly overlapping logical streams. The different sets of streams will not overlap.

The IDs of the logical domains created will start at `1` and end with `in_NumLogDomains`. The IDs of the logical streams created will start at `0` and end at `(in_pStreamsPerDomain[0] + ... + in_pStreamsPerDomain[in_NumLogDomains-1]) * in_LogStreamOversubscription`.

With `in_LogStreamOversubscription` larger than `1` logical streams are numbered across all the physical streams first vs within. This causes noncontinuous logical streams IDs inside single physical stream.

For subsequent invocations of `hStreams_app_init*`, those functions will attempt to reuse the logical domains created by the first app API-level initialization and create an additional distribution of logical streams in those logical domains. The IDs of the newly created logical streams will be enumerated in a similar fashion to what is described above with the exception that the lowest-numbered new logical stream ID will be equal to the highest ID from streams added in previous initializations plus one.

If in_NumLogDomains is not an even multiple of available Intel(R) Xeon Phi(TM) coprocessors or the coprocessors have different number of hardware threads, the logical domains will not be uniform; using as many hardware threads as possible is favored over uniformity of logical domains.

Parameters

in_NumLogDomains [in] number of logical domains to spread across available Intel(R) Xeon Phi(TM) coprocessors

in_pStreamsPerDomain [in] physical streams per logical domain. If some values are equal to 0, the corresponding domains are unused

in_LogStreamOversubscription [in] number of logical streams that should map to each of the physical streams created

Returns

If successful, hStreams_app_init_domains() returns HSTR_RESULT_SUCCESS. Otherwise, it returns one of the following errors:

- HSTR_RESULT_DEVICE_NOT_INITIALIZED if hStreams cannot be initialized. Possible causes include incorrect values of environment variables or no Intel(R) Xeon Phi(TM) coprocessor being available
- HSTR_RESULT_NULL_PTR if in_pStreamsPerDomain is NULL
- HSTR_RESULT_OUT_OF_RANGE if in_NumLogDomains is 0
- HSTR_RESULT_OUT_OF_RANGE if this call is *not* the first initialization and the desired number of logical domains does not match what was created during the first initialization.
- HSTR_RESULT_OUT_OF_RANGE if in_LogStreamOversubscription is 0
- HSTR_RESULT_OUT_OF_RANGE if a value of an entry in in_pStreamsPerDomain exceeds the number of hardware threads available in that specific domain
- HSTR_RESULT_OUT_OF_RANGE if all entries of in_pStreamsPerDomain are 0
- HSTR_RESULT_ALREADY_FOUND if an attempt is made to add a stream that is already present. This can be returned if the initialization has already happened.
- HSTR_RESULT_BAD_NAME if dynamic-link dependences cannot be located

Thread safety:

Not thread safe.

7.2.1.7 HSTR_RESULT hStreams_app_init_in_version (uint32_t *in_StreamsPerDomain*, uint32_t *in_LogStreamOversubscription*, const char * *interface_version*)

Initialize hStreams homogenously across all available Intel(R) Xeon Phi(TM) coprocessors.

If no initialization through hStreams_app_init* routines had been performed beforehand, this function will detect all available Intel(R) Xeon Phi(TM) coprocessors and attempt to create one logical domain on each of them. In each of those logical domains, in_StreamsPerDomain sets of streams will be created, each of them consisting of in_LogStreamOversubscription exactly overlapping logical streams.

The IDs of the logical domains created will start at 1 and end with the number of Intel(R) Xeon Phi(TM) coprocessors available in the system. The IDs of the logical streams created will start at 0 and end at in_StreamsPerDomain * in_LogStreamOversubscription - 1.

None of the first in_StreamsPerDomain streams will overlap each other. If in_LogStreamOversubscription > 0, stream number in_StreamsPerDomain will exactly overlap stream number 0, stream number in_StreamsPerDomain + 1 will exactly overlap stream number 1 and so on.

For subsequent invocations of hStreams_app_init*, those functions will attempt to reuse the logical domains created by the first app API-level initialization and create an additional distribution of logical streams in those logical domains. The IDs of the newly created logical streams will be enumerated in a similar fashion to what is described above with the exception that the lowest-numbered new logical stream ID will be equal to the highest ID from streams added in previous initializations plus one.

Parameters

in_StreamsPerDomain [in] number of physical streams to create in each logical domain

in_LogStreamOversubscription [in] degree of oversubscription for logical streams

Returns

If successful, hStreams_app_init() returns HSTR_RESULT_SUCCESS. Otherwise, it returns one of the following errors:

- HSTR_RESULT_DEVICE_NOT_INITIALIZED if hStreams cannot be initialized. Possible causes include incorrect values of environment variables or no Intel(R) Xeon Phi(TM) coprocessor being available
- HSTR_RESULT_OUT_OF_RANGE if this call is *not* the first initialization and the desired number of logical domains does not match what was created during the first initialization.
- HSTR_RESULT_OUT_OF_RANGE if in_LogStreamOversubscription is 0
- HSTR_RESULT_OUT_OF_RANGE if in_StreamsPerDomain is 0
- HSTR_RESULT_OUT_OF_RANGE if in_StreamsPerDomain exceeds the number of logical CPUs available in any of the Intel(R) Xeon Phi(TM) coprocessors detected
- HSTR_RESULT_ALREADY_FOUND if an attempt is made to add a stream that is already present. This can be returned if the initialization has already happened.
- HSTR_RESULT_BAD_NAME if dynamic-link dependences cannot be located

Thread safety:

Not thread safe.

7.2.1.8 HSTR_RESULT hStreams_app_invoke (HSTR_LOG_STR *in_LogStreamID*, const char * *in_pFuncName*, uint32_t *in_NumScalarArgs*, uint32_t *in_NumHeapArgs*, uint64_t * *in_pArgs*, HSTR_EVENT * *out_pEvent*, void * *out_pReturnValue*, uint16_t *in_ReturnValueSize*)

Enqueue an execution of a user-defined function in a stream.

Places an execution of a user-defined function in the stream's internal queue. The function to be called shall be compiled and loaded to the sink process so that hStreams can locate the appropriate symbol and invoke it on the stream's sink endpoint.

Parameters

in_LogStreamID [in] ID of logical stream associated to enqueue the action in

in_pFuncName [in] Null-terminated string with name of the function to be executed

in_NumScalarArgs [in] Number of arguments to be copied by value for remote invocation

in_NumHeapArgs [in] Number of arguments which are buffer addreses to be translated to sink-side instantiations' addresses

in_pArgs [in] Array of in_NumScalarArgs+in_NumHeapArgs arguments as 64-bit unsigned integers with scalar args first and buffer args second

out_pEvent [out] pointer to event which will be signaled once the action completes

out_pReturnValue [out] pointer to host-side memory the remote invocation can asynchronously write to

in_ReturnValueSize [in] the size of the asynchronous return value memory

Returns

If successful, `hStreams_app_invoke()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if the library had not been initialized properly
- `HSTR_RESULT_NOT_FOUND` if `in_LogStreamID` is not a valid logical stream ID
- `HSTR_RESULT_NOT_FOUND` if at least one of the buffer arguments is not in a buffer that had been instantiated for `in_LogStreamID's` logical domain
- `HSTR_RESULT_BAD_NAME` if `in_pFunctionName` is `NULL`
- `HSTR_RESULT_BAD_NAME` if no symbol named `in_pFunctionName` is found on the streams's sink endpoint
- `HSTR_RESULT_BAD_NAME` if `in_pFunctionName` is longer than `HSTR_MAX_FUNC_NAME_SIZE`
- `HSTR_RESULT_OUT_OF_RANGE` if `in_ReturnValueSize` exceeds `HSTR_RETURN_SIZE_LIMIT`
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_ReturnValueSize != 0` and `in_pReturnValue` is `NULL` or `in_ReturnValueSize == 0` and `in_pReturnValue` is not `NULL`
- `HSTR_RESULT_TOO_MANY_ARGS` if `in_NumScalarArgs + in_NumHeapArgs > HSTR_ARGS_SUPPORTED`
- `HSTR_RESULT_NULL_PTR` `in_numScalarArgs + in_numHeapArgs > 0` but `in_pArgs` is `NULL`

Thread safety:

All actions enqueued through concurrent calls to `hStreams_app_invoke()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.2.1.9 HSTR_RESULT hStreams_app_stream_sync (HSTR_LOG_STR *in_LogStreamID*)

Block until all the operation enqueued in a stream have completed.

Parameters

in_LogStreamID [in] ID of the logical stream

Returns

If successful, `hStreams_app_stream_sync()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly
- `HSTR_RESULT_NOT_FOUND` if `in_LogStreamID` is not a valid logical stream ID
- `HSTR_RESULT_TIME_OUT_REACHED` if timeout was reached while waiting on the completion of stream's actions.
- `HSTR_RESULT_EVENT_CANCELED` if one of the events in the stream was canceled
- `HSTR_RESULT_REMOTE_ERROR` if there was a remote error, e.g. the remote process died

See also

HSTR_OPTIONS.time_out_ms_val

Thread safety:

Thread safe.

7.2.1.10 HSTR_RESULT hStreams_app_thread_sync ()

Block until all the operation enqueued in all the streams have completed.

Returns

If successful, `hStreams_app_thread_sync()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly
- `HSTR_RESULT_TIME_OUT_REACHED` if timeout was reached while waiting on the completion of actions.
- `HSTR_RESULT_EVENT_CANCELED` if one of the events was canceled
- `HSTR_RESULT_REMOTE_ERROR` if there was a remote error, e.g. the remote process died

See also

HSTR_OPTIONS.time_out_ms_val

Thread safety:

Thread safe.

7.3 Common building blocks

Functions

- HSTR_RESULT hStreams_app_memset (HSTR_LOG_STR in_LogStreamID, void *in_pWriteAddr, int in_Value, uint64_t in_NumBytes, HSTR_EVENT *out_pEvent)

  Set remote memory to a value, using a named stream.

- HSTR_RESULT hStreams_app_memcpy (HSTR_LOG_STR in_LogStreamID, void *in_pWriteAddr, void *in_pReadAddr, uint64_t in_NumBytes, HSTR_EVENT *out_pEvent)

  copy remote memory, using a named stream

- HSTR_RESULT hStreams_app_sgemm (HSTR_LOG_STR in_LogStreamID, const CBLAS_ORDER Order, const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int64_t M, const int64_t N, const int64_t K, const float alpha, const float *A, const int64_t ldA, const float *B, const int64_t ldB, const float beta, float *C, const int64_t ldC, HSTR_EVENT *out_pEvent)

  perform a remote cblas sgemm

- HSTR_RESULT hStreams_app_dgemm (HSTR_LOG_STR in_LogStreamID, const CBLAS_ORDER Order, const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int64_t M, const int64_t N, const int64_t K, const double alpha, const double *A, const int64_t ldA, const double *B, const int64_t ldB, const double beta, double *C, const int64_t ldC, HSTR_EVENT *out_pEvent)

  perform a remote cblas dgemm

- HSTR_RESULT hStreams_app_cgemm (HSTR_LOG_STR in_LogStreamID, const CBLAS_ORDER Order, const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int64_t M, const int64_t N, const int64_t K, const void *alpha, const void *A, const int64_t ldA, const void *B, const int64_t ldB, const void *beta, void *C, const int64_t ldC, HSTR_EVENT *out_pEvent)

  perform a remote cblas cgemm

- HSTR_RESULT hStreams_app_zgemm (HSTR_LOG_STR in_LogStreamID, const CBLAS_ORDER Order, const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int64_t M, const int64_t N, const int64_t K, const void *alpha, const void *A, const int64_t ldA, const void *B, const int64_t ldB, const void *beta, void *C, const int64_t ldC, HSTR_EVENT *out_pEvent)

  perform a remote cblas zgemm

7.3.1 Detailed Description

These functions are provided as examples of common building blocks for an application making use of the hStreams library. Two memory-related functions are provided - hStreams_app_memset() and hStreams_app_memcpy(). There are also four functions which perform remote matrix multiplication using kernels from the Intel(R) Math Kernel Library (Intel(R) MKL). Their parameters correspond to those used by the Intel(R) MKL routines.

7.3.2 Function Documentation

7.3.2.1 HSTR_RESULT hStreams_app_cgemm (HSTR_LOG_STR *in_LogStreamID*, const CBLAS_ORDER *Order*, const CBLAS_TRANSPOSE *TransA*, const CBLAS_TRANSPOSE *TransB*, const int64_t *M*, const int64_t *N*, const int64_t *K*, const void * *alpha*, const void * *A*, const int64_t *ldA*, const void * *B*, const int64_t *ldB*, const void * *beta*, void * *C*, const int64_t *ldC*, HSTR_EVENT * *out_pEvent*)

perform a remote cblas cgemm

Parameters

in_LogStreamID [in] 0-based index of logical stream

CBLAS-related parameters [in] MKL CBLAS input parameters, in their API order

NOTE: the actual types of A, B, C, alpha and beta are MKL_Complex8 *.

Parameters

out_pEvent [out] opaque event handle used for synchronization

Returns

HSTR_RESULT_NOT_INITIALIZED if hStreams had not been initialized properly.
HSTR_RESULT_NULL_PTR if A, B or C is NULL
HSTR_RESULT_SUCCESS if successful

Thread safety:

All actions enqueued through concurrent calls to `hStreams_app_cgemm()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.3.2.2 HSTR_RESULT hStreams_app_dgemm (HSTR_LOG_STR *in_LogStreamID*, const CBLAS_ORDER *Order*, const CBLAS_TRANSPOSE *TransA*, const CBLAS_TRANSPOSE *TransB*, const int64_t *M*, const int64_t *N*, const int64_t *K*, const double *alpha*, const double * *A*, const int64_t *ldA*, const double * *B*, const int64_t *ldB*, const double *beta*, double * *C*, const int64_t *ldC*, HSTR_EVENT * *out_pEvent*)

perform a remote cblas dgemm

Parameters

in_LogStreamID [in] 0-based index of logical stream

CBLAS-related parameters [in] MKL CBLAS input parameters, in their API order

out_pEvent [out] opaque event handle used for synchronization

Returns

HSTR_RESULT_NOT_INITIALIZED if hStreams had not been initialized properly.
HSTR_RESULT_NULL_PTR if A, B or C is NULL
HSTR_RESULT_SUCCESS if successful

Thread safety:

All actions enqueued through concurrent calls to `hStreams_app_dgemm()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.3.2.3 HSTR_RESULT hStreams_app_memcpy (HSTR_LOG_STR *in_LogStreamID*, void * *in_pWriteAddr*, void * *in_pReadAddr*, uint64_t *in_NumBytes*, HSTR_EVENT * *out_pEvent*)

copy remote memory, using a named stream

Parameters

in_LogStreamID [in] 0-based index of logical stream

in_pWriteAddr [in] Host proxy address pointer to the base of a memory area to write the copy to. This address gets mapped to a corresponding address in the sink domain associated with in_LogStreamID

in_pReadAddr [in] Host proxy address pointer to the base of a memory area to read the copy from. This address gets mapped to a corresponding address in the sink domain associated with in_LogStreamID

in_NumBytes [in] the number of bytes of the sink buffer to be set or copied

out_pEvent [out] opaque event handle used for synchronization

Returns

HSTR_RESULT_NOT_INITIALIZED if hStreams had not been initialized properly.
HSTR_RESULT_NULL_PTR if in_pWriteAddr or in_pReadAddr is NULL
HSTR_RESULT_NOT_FOUND if in_LogStreamID is not found to have an associated hStream, or at least one of the heap arguments is not in an allocated buffer.
HSTR_RESULT_SUCCESS if successful

Thread safety:

All actions enqueued through concurrent calls to `hStreams_app_memcpy()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.3.2.4 HSTR_RESULT hStreams_app_memset (HSTR_LOG_STR *in_LogStreamID*, void * *in_pWriteAddr*, int *in_Value*, uint64_t *in_NumBytes*, HSTR_EVENT * *out_pEvent*)

Set remote memory to a value, using a named stream.

Parameters

in_LogStreamID [in] 0-based index of logical stream

in_pWriteAddr [in] Host proxy address pointer to the base of a memory area to write in_Value to. This address gets mapped to a corresponding address in the sink domain associated with in_LogStreamID

in_Value [in] the byte-sized value that memory is set to

in_NumBytes [in] the number of bytes of the sink buffer to be set or copied

out_pEvent [out] opaque event handle used for synchronization

Returns

HSTR_RESULT_NOT_INITIALIZED if hStreams had not been initialized properly.
HSTR_RESULT_NULL_PTR if in_pWriteAddr is NULL
HSTR_RESULT_NOT_FOUND if in_LogStreamID is not found to have an associated hStream, or at least one of the heap arguments is not in an allocated buffer.
HSTR_RESULT_SUCCESS if successful

Thread safety:

All actions enqueued through concurrent calls to `hStreams_app_memset()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.3.2.5 HSTR_RESULT hStreams_app_sgemm (HSTR_LOG_STR *in_LogStreamID*, const CBLAS_ORDER *Order*, const CBLAS_TRANSPOSE *TransA*, const CBLAS_TRANSPOSE *TransB*, const int64_t *M*, const int64_t *N*, const int64_t *K*, const float *alpha*, const float * *A*, const int64_t *ldA*, const float * *B*, const int64_t *ldB*, const float *beta*, float * *C*, const int64_t *ldC*, HSTR_EVENT * *out_pEvent*)

perform a remote cblas sgemm

Parameters

in_LogStreamID [in] 0-based index of logical stream

CBLAS-related parameters [in] MKL CBLAS input parameters, in their API order

out_pEvent [out] opaque event handle used for synchronization

Returns

HSTR_RESULT_NOT_INITIALIZED if hStreams had not been initialized properly.
HSTR_RESULT_NULL_PTR if A, B or C is NULL
HSTR_RESULT_SUCCESS if successful

Thread safety:

All actions enqueued through concurrent calls to `hStreams_app_sgemm()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.3.2.6 HSTR_RESULT hStreams_app_zgemm (HSTR_LOG_STR *in_LogStreamID*, const CBLAS_ORDER *Order*, const CBLAS_TRANSPOSE *TransA*, const CBLAS_TRANSPOSE *TransB*, const int64_t *M*, const int64_t *N*, const int64_t *K*, const void * *alpha*, const void * *A*, const int64_t *ldA*, const void * *B*, const int64_t *ldB*, const void * *beta*, void * *C*, const int64_t *ldC*, HSTR_EVENT * *out_pEvent*)

perform a remote cblas zgemm

Parameters

in_LogStreamID [in] 0-based index of logical stream

CBLAS-related parameters [in] MKL CBLAS input parameters, in their API order

NOTE: the actual types of A, B, C, alpha and beta are MKL_Complex16 *.

Parameters

out_pEvent [out] opaque event handle used for synchronization

Returns

HSTR_RESULT_NOT_INITIALIZED if hStreams had not been initialized properly.
HSTR_RESULT_NULL_PTR if A, B or C is NULL
HSTR_RESULT_SUCCESS if successful

Thread safety:

All actions enqueued through concurrent calls to `hStreams_app_zgemm()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.4 hStreams AppApiSink

Functions

- HSTREAMS_EXPORT void hStreams_memcpy_sink (uint64_t byte_len, uint64_t *src, uint64_t *dest)

  Calls memcpy from string.h from (remote) sink side.

- HSTREAMS_EXPORT void hStreams_memset_sink (uint64_t byte_len, uint64_t char_value, uint64_t *buf)

  Calls memset from string.h from (remote) sink side.

- HSTREAMS_EXPORT void hStreams_sgemm_sink (uint64_t arg0, uint64_t arg1, uint64_t arg2, uint64_t arg3, uint64_t arg4, uint64_t arg5, uint64_t arg6, uint64_t arg7, uint64_t arg8, uint64_t arg9, uint64_t arg10, uint64_t arg11, uint64_t arg12, uint64_t arg13)

  Calls sgemm from (remote) sink side.

- HSTREAMS_EXPORT void hStreams_dgemm_sink (uint64_t arg0, uint64_t arg1, uint64_t arg2, uint64_t arg3, uint64_t arg4, uint64_t arg5, uint64_t arg6, uint64_t arg7, uint64_t arg8, uint64_t arg9, uint64_t arg10, uint64_t arg11, uint64_t arg12, uint64_t arg13)

  Calls dgemm from (remote) sink side.

- HSTREAMS_EXPORT void hStreams_cgemm_sink (uint64_t arg0, uint64_t arg1, uint64_t arg2, uint64_t arg3, uint64_t arg4, uint64_t arg5, uint64_t arg6, uint64_t arg7, uint64_t arg8, uint64_t arg9, uint64_t arg10, uint64_t arg11, uint64_t arg12, uint64_t arg13)

  Calls cgemm from (remote) sink side.

- HSTREAMS_EXPORT void hStreams_zgemm_sink (uint64_t arg0, uint64_t arg1, uint64_t arg2, uint64_t arg3, uint64_t arg4, uint64_t arg5, uint64_t arg6, uint64_t arg7, uint64_t arg8, uint64_t arg9, uint64_t arg10, uint64_t arg11, uint64_t arg12, uint64_t arg13, uint64_t arg14, uint64_t arg15)

  Calls zgemm from (remote) sink side.

7.4.1 Function Documentation

7.4.1.1 HSTREAMS_EXPORT void hStreams_cgemm_sink (uint64_t *arg0*, uint64_t *arg1*, uint64_t *arg2*, uint64_t *arg3*, uint64_t *arg4*, uint64_t *arg5*, uint64_t *arg6*, uint64_t *arg7*, uint64_t *arg8*, uint64_t *arg9*, uint64_t *arg10*, uint64_t *arg11*, uint64_t *arg12*, uint64_t *arg13*)

Calls cgemm from (remote) sink side.

- For use on sink side only

Parameters

arg0 [in] const CBLAS_ORDER arg0, Order

arg1 [in] const CBLAS_TRANSPOSE arg1, TransA

arg2 [in] const CBLAS_TRANSPOSE arg2, TransB

arg3 [in] const MKL_INT arg3, M

arg4 [in] const MKL_INT arg4, N

arg5 [in] const MKL_INT arg5, K

arg6 [in] MKL_Complex8 arg6, alpha (actually a MKL_Complex8 passed in a uint64_t)

arg7 [in] const MKL_INT arg7, lda

arg8 [in] const MKL_INT arg8, ldb

arg9 [in] MKL_Complex8 arg9 beta (actually a MKL_Complex8 passed in a uint64_t)

arg10 [in] const MKL_INT arg10, ldc

arg11 [in] const void ∗arg11, A

arg12 [in] const void ∗arg12, B

arg13 [in] void ∗arg13); C

Returns

void

Thread safety:

Thread safe for calls on different data.

7.4.1.2 HSTREAMS_EXPORT void hStreams_dgemm_sink (uint64_t *arg0*, uint64_t *arg1*, uint64_t *arg2*, uint64_t *arg3*, uint64_t *arg4*, uint64_t *arg5*, uint64_t *arg6*, uint64_t *arg7*, uint64_t *arg8*, uint64_t *arg9*, uint64_t *arg10*, uint64_t *arg11*, uint64_t *arg12*, uint64_t *arg13*)

Calls dgemm from (remote) sink side.

- For use on sink side only

Parameters

arg0 [in] const CBLAS_ORDER arg0, Order

arg1 [in] const CBLAS_TRANSPOSE arg1, TransA

arg2 [in] const CBLAS_TRANSPOSE arg2, TransB

arg3 [in] const MKL_INT arg3, M

arg4 [in] const MKL_INT arg4, N

arg5 [in] const MKL_INT arg5, K

arg6 [in] double arg6, alpha (actually a double passed in a uint64_t)

arg7 [in] const MKL_INT arg7, lda

arg8 [in] const MKL_INT arg8, ldb

arg9 [in] double arg9, beta (actually a double passed in a uint64_t)

arg10 [in] const MKL_INT arg10, ldc

arg11 [in] const double *arg11, A

arg12 [in] const double *arg12, B

arg13 [in] double *arg13); C

Returns

void

Thread safety:

Thread safe for calls on different data.

7.4.1.3 HSTREAMS_EXPORT void hStreams_memcpy_sink (uint64_t *byte_len*, uint64_t * *src*, uint64_t * *dest*)

Calls memcpy from string.h from (remote) sink side.

For use on sink side only

Parameters

byte_len [in] number of bytes to copy

src [in] source address to copy from

dest [in] destination address to copy to

Returns

void

Thread safety:

Thread safe for calls on different data.

7.4.1.4 HSTREAMS_EXPORT void hStreams_memset_sink (uint64_t *byte_len*, uint64_t *char_value*, uint64_t * *buf*)

Calls memset from string.h from (remote) sink side.

For use on sink side only

Parameters

byte_len [in] number of bytes to copy

char_value [in] character-sized value to fill with

buf [in] starting address to fill at

Returns

void

Thread safety:

Thread safe for calls on different data.

7.4.1.5 HSTREAMS_EXPORT void hStreams_sgemm_sink (uint64_t *arg0*, uint64_t *arg1*, uint64_t *arg2*, uint64_t *arg3*, uint64_t *arg4*, uint64_t *arg5*, uint64_t *arg6*, uint64_t *arg7*, uint64_t *arg8*, uint64_t *arg9*, uint64_t *arg10*, uint64_t *arg11*, uint64_t *arg12*, uint64_t *arg13*)

Calls sgemm from (remote) sink side.

- For use on sink side only

Parameters

arg0 [in] const CBLAS_ORDER arg0, Order

arg1 [in] const CBLAS_TRANSPOSE arg1, TransA

arg2 [in] const CBLAS_TRANSPOSE arg2, TransB

arg3 [in] const MKL_INT arg3, M

arg4 [in] const MKL_INT arg4, N

arg5 [in] const MKL_INT arg5, K

arg6 [in] float arg6, alpha (actually a float passed in a uint64_t)

arg7 [in] const MKL_INT arg7, lda

arg8 [in] const MKL_INT arg8, ldb

arg9 [in] float arg9, beta (actually a float passed in a uint64_t)

arg10 [in] const MKL_INT arg10, ldc

arg11 [in] const float ∗arg11, A

arg12 [in] const float ∗arg12, B

arg13 [in] float ∗arg13); C

Returns

void

Thread safety:

Thread safe for calls on different data.

7.4.1.6 HSTREAMS_EXPORT void hStreams_zgemm_sink (uint64_t *arg0*, uint64_t *arg1*, uint64_t *arg2*, uint64_t *arg3*, uint64_t *arg4*, uint64_t *arg5*, uint64_t *arg6*, uint64_t *arg7*, uint64_t *arg8*, uint64_t *arg9*, uint64_t *arg10*, uint64_t *arg11*, uint64_t *arg12*, uint64_t *arg13*, uint64_t *arg14*, uint64_t *arg15*)

Calls zgemm from (remote) sink side.

- For use on sink side only

Parameters

arg0 [in] const CBLAS_ORDER arg0, Order

arg1 [in] const CBLAS_TRANSPOSE arg1, TransA

arg2 [in] const CBLAS_TRANSPOSE arg2, TransB

arg3 [in] const MKL_INT arg3, M

arg4 [in] const MKL_INT arg4, N

arg5 [in] const MKL_INT arg5, K

arg6 [in] MKL_Complex16 in arg6 and arg7, alpha (actually a MKL_Complex16 passed in two uint64_t's) and arg7,

arg7 [in] MKL_Complex16 in arg6 and arg7, alpha (actually a MKL_Complex16 passed in two uint64_t's) and arg7,

arg8 [in] const MKL_INT arg8, lda

arg9 [in] const MKL_INT arg9, ldb

arg10 [in] MKL_Complex16 in arg10 and arg11, beta (actually a MKL_Complex16 passed in two uint64_t's) and arg11,

arg11 [in] MKL_Complex16 in arg10 and arg11, beta (actually a MKL_Complex16 passed in two uint64_t's) and arg11,

arg12 [in] const MKL_INT arg12, ldc

arg13 [in] const void ∗arg13, A

arg14 [in] const void ∗arg14, B

arg15 [in] void ∗arg15); C

Returns

void

Thread safety:

Thread safe for calls on different data.

7.5 hStreams Source

Modules

- hStreams Source - General
- hStreams Source - Domains
- hStreams Source - Stream management
- hStreams Source - Stream usage
- hStreams Source - Sync
- hStreams Source - Memory management
- hStreams Source - Error handling
- hStreams Source - Configuration
- hStreams Utilities

7.6 hStreams Source - General

Functions

- HSTR_RESULT hStreams_InitInVersion (const char *interface_version)
  Initialize hStreams-related state.
- HSTR_RESULT hStreams_IsInitialized ()
  Check if hStreams has been initialised properly.
- HSTR_RESULT hStreams_Fini ()
  Finalize hStreams-related state.

7.6.1 Function Documentation

7.6.1.1 HSTR_RESULT hStreams_Fini ()

Finalize hStreams-related state.

Destroys hStreams internal structures and clears the state of the library. All logical domains, streams and buffers are destroyed as a result of this call.

Returns

If successful, `hStreams_Fini()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized prior to this call.

Thread safety:

Thread safe.

7.6.1.2 HSTR_RESULT hStreams_InitInVersion (const char * *interface_version*)

Initialize hStreams-related state.

Must be called before all other hStreams functions, else HSTR_RESULT_NOT_INITIALIZED will result.

Note that hStreams_Init() will load the sink-side libraries during initialization. To load the sink-side libraries, hStreams_Init() first, attempts to find a sibling file as the source-side executable with suffix "_mic.so". For example if the source-side executable is called 'test_app', then, hStreams_Init() attempts to find a file named 'test_app_mic.so' somewhere in the list of directories defined in the SINK_LD_LIBRARY_PATH environment variable. (On windows, the '.exe' extension is removed from the source-side executable file name before the sibling file is sought). Next, if the HSTR_OPTIONS struct contains a number of libraries, they will be loaded, again searching the SINK_LD_LIBRARY_PATH for each of the libraries specified there.

Returns

If successful, `hStreams_InitInVersion()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_DEVICE_NOT_INITIALIZED` if the library cannot be initialized properly, e.g. because no MICs are available
- `HSTR_RESULT_OUT_OF_RANGE` if the `interface_version` argument does not correspond to a valid interface version supported by the library

Thread safety:

Concurrent calls to `hStreams_InitInVersion()` are serialized internally by the implementation. The first thread to acquire exclusivity will attempt to initialize the library. If the initialisation is successfull, the first thread will return `HSTR_RESULT_SUCCESS` as well as all of the other threads that were serialized. If the initialisation of the library by the first thread fails, the first thread shall return an error code and the second thread to acquire exclusivity will again try to initialize the library.

7.6.1.3 HSTR_RESULT hStreams_IsInitialized ()

Check if hStreams has been initialised properly.

Returns

If successful, `hStreams_IsInitialized()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.

Thread safety:

Thread safe.

7.7 hStreams Source - Domains

Functions

- HSTR_RESULT hStreams_GetNumPhysDomains (uint32_t *out_pNumPhysDomains, uint32_t *out_pNumActivePhysDomains, bool *out_pHomogeneous)

  Returns number of discovered and active physical domains.

- HSTR_RESULT hStreams_GetPhysDomainDetails (HSTR_PHYS_DOM in_PhysDomain, uint32_t *out_pNumThreads, HSTR_ISA_TYPE *out_pISA, uint32_t *out_pCoreMaxMHz, HSTR_CPU_MASK out_MaxCPUmask, HSTR_CPU_MASK out_AvoidCPUmask, uint64_t *out_pSupportedMemTypes, uint64_t out_pPhysicalBytesPerMemType[HSTR_MEM_TYPE_SIZE])

  Returns information about specified physical domain.

- HSTR_RESULT hStreams_GetAvailable (HSTR_PHYS_DOM in_PhysDomainID, HSTR_CPU_MASK out_AvailableCPUmask)

  Returns unused yet cpu threads.

- HSTR_RESULT hStreams_AddLogDomain (HSTR_PHYS_DOM in_PhysDomainID, HSTR_CPU_MASK in_CPUmask, HSTR_LOG_DOM *out_pLogDomainID, HSTR_OVERLAP_TYPE *out_pOverlap)

  Create a new logical domain in a physical domain.

- HSTR_RESULT hStreams_RmLogDomains (uint32_t in_NumLogDomains, HSTR_LOG_DOM *in_pLogDomainIDs)

  Remove logical domains.

- HSTR_RESULT hStreams_GetNumLogDomains (HSTR_PHYS_DOM in_PhysDomainID, uint32_t *out_pNumLogDomains)

  Return number logical domains associated with a physical domain.

- HSTR_RESULT hStreams_GetLogDomainIDList (HSTR_PHYS_DOM in_PhysDomainID, uint32_t in_NumLogDomains, HSTR_LOG_DOM *out_pLogDomainIDs)

  Returns list of logical domains attached to provided physical domain.

- HSTR_RESULT hStreams_GetLogDomainDetails (HSTR_LOG_DOM in_LogDomainID, HSTR_PHYS_DOM *out_pPhysDomainID, HSTR_CPU_MASK out_CPUmask)

  Returns associated cpu mask and physical domain to provided logical domain.

7.7.1 Function Documentation

7.7.1.1 HSTR_RESULT hStreams_AddLogDomain (HSTR_PHYS_DOM *in_PhysDomainID*, HSTR_CPU_MASK *in_CPUmask*, HSTR_LOG_DOM * *out_pLogDomainID*, HSTR_OVERLAP_TYPE * *out_pOverlap*)

Create a new logical domain in a physical domain.

The CPU resources specified by a logical domain's CPU mask are not allowed to partially overlap within the resources of another logical domain's in the same physical domain. They must be either disjoint or fully overlap each other. Please note that logical domains with exactly overlapping CPU resources are considered to be distinct entities, i.e. they do not alias the same entity as is the case with exactly overlapping logical streams.

Logical domain IDs are generated by hStreams.

Parameters

in_PhysDomainID [in] ID of the physical domain to add logical domain to

in_CPUmask [in] HW threads mask that the logical domain is bound to

out_pLogDomainID [out] Generated logical domain ID

out_pOverlap [out] Resulting overlap status

Returns

If successful, `hStreams_AddLogDomain()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if the library had not been initialized properly
- `HSTR_RESULT_DOMAIN_OUT_OF_RANGE` if there is no valid physical domain with ID `in_PhysDomainID`
- `HSTR_RESULT_NULL_PTR` if `out_pLogDomainID` is `NULL`
- `HSTR_RESULT_NULL_PTR` if `out_pOverlap` is `NULL` correspond to a valid logical domain
- `HSTR_RESULT_OVERLAPPING_RESOURCES`, if `in_CPUmask` partially overlaps any logical domain previously added in physical domain `in_PhysDomainID`
- `HSTR_RESULT_CPU_MASK_OUT_OF_RANGE` if `in_CPUmask` is empty
- `HSTR_RESULT_CPU_MASK_OUT_OF_RANGE` if `in_CPUmask` is outside the range of the `in_PhysDomainID` physical domain
- `HSTR_RESULT_OUT_OF_MEMORY` if any incremental buffer cannot be instantiated because of memory exhaustion or unavailability of requested memory kind, subject to the buffer's memory allocation policy.

Thread safety:

Thread safe.

7.7.1.2 HSTR_RESULT hStreams_GetAvailable (HSTR_PHYS_DOM *in_PhysDomainID*, HSTR_CPU_MASK *out_AvailableCPUmask*)

Returns unused yet cpu threads.

This API covers only dynamic information, to complement GetPhysDomainDetails hStreams_GetAvailable reveals unused threads for a given physical domain.

Parameters

in_PhysDomainID [in] Index of domain Domain numbering starts with 0 for domains, HSTR_SRC_PHYS_DOMAIN for the host.

out_AvailableCPUmask [out] hStreams cpu mask covering all threads on specific domain that are currently not associated with any stream

Returns

If successful, `hStreams_GetAvailable()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_DOMAIN_OUT_OF_RANGE` if `in_PhysDomainID` is out of range or provided physical domain is inactive.
- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.

Thread safety:

Thread safe.

7.7.1.3 HSTR_RESULT hStreams_GetLogDomainDetails (HSTR_LOG_DOM *in_LogDomainID*, HSTR_PHYS_DOM * *out_pPhysDomainID*, HSTR_CPU_MASK *out_CPUmask*)

Returns associated cpu mask and physical domain to provided logical domain.

Parameters

in_LogDomainID [in] ID of logical domain to look up Logical domains are associated with exactly one physical domain.

out_pPhysDomainID [out] physical domain ID associated with provided logical domain

out_CPUmask [out] cpu mask associated with provided logical domain

Returns

If successful, `hStreams_GetLogDomainDetails()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_FOUND` if `in_LogDomainID` is out of range
- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.
- `HSTR_RESULT_NULL_PTR`, if `out_pPhysDomainID` is NULL

Thread safety:

Thread safe.

7.7.1.4 HSTR_RESULT hStreams_GetLogDomainIDList (HSTR_PHYS_DOM *in_PhysDomainID*, uint32_t *in_NumLogDomains*, HSTR_LOG_DOM * *out_pLogDomainIDs*)

Returns list of logical domains attached to provided physical domain.

Note

Before calling `hStreams_GetLogDomainIDList` you should call `hStreams_GetNumLogDomains()` function to get number of logical domains attached to provided physical domain.

Parameters

in_PhysDomainID [in] ID of physical domain for which we are querying

in_NumLogDomains [in] Number of allocated by user elements inside `out_pLogDomainIDs` array

out_pLogDomainIDs [out] Array with `in_NumLogDomains` logical domain IDs

Returns

If successful, `hStreams_GetLogDomainIDList()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.
- `HSTR_RESULT_DOMAIN_OUT_OF_RANGE` if `in_PhysDomainID` is out of range or provided physical domain is inactive.
- `HSTR_RESULT_NULL_PTR`, if `out_pLogDomainIDs` is `NULL`
- `HSTR_RESULT_NOT_FOUND` if fewer than `in_NumLogDomains` mappings found
- `HSTR_RESULT_INCONSISTENT_ARGS` if more than `in_NumLogDomains` mappings are found.

Thread safety:

Thread safe. Please note that subsequent calls to `hStreams_GetNumLogDomains()` and `hStreams_GetLogDomainIDList()` do not form an atomic operation and thus logical domains might have been added or removed between the calls to these APIs.

7.7.1.5 HSTR_RESULT hStreams_GetNumLogDomains (HSTR_PHYS_DOM *in_PhysDomainID*, uint32_t * *out_pNumLogDomains*)

Return number logical domains associated with a physical domain.

Parameters

in_PhysDomainID [in] Physical domain ID: `HSTR_SRC_PHYS_DOMAIN` for source, remote domains start at 0

out_pNumLogDomains [out] Number of logical domains associated with `in_PhysDomainID`

Returns

If successful, `hStreams_GetNumLogDomains()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_DOMAIN_OUT_OF_RANGE` if `in_PhysDomainID` is out of range or provided physical domain is inactive.
- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.

- HSTR_RESULT_NULL_PTR, if out_pNumLogDomains is NULL

Thread safety:

Thread safe.

7.7.1.6 HSTR_RESULT hStreams_GetNumPhysDomains (uint32_t * *out_pNumPhysDomains*, uint32_t * *out_pNumActivePhysDomains*, bool * *out_pHomogeneous*)

Returns number of discovered and active physical domains.

This API covers only static information about physical domains. hStreams_GetAvailable reveals unused threads for a given domain.

Parameters

out_pNumPhysDomains [out] Number of sink physical domains, e.g. MIC accelerator cards. This number cannot be higher phys_domains_limit set in HSTR_OPTIONS. See hStreams_Init() for more details.

out_pNumActivePhysDomains [out] Number of active domains, at initialization time

out_pHomogeneous [out] True if all physical domains besides host have the same resources and capabilities, else false

Returns

If successful, hStreams_GetNumPhysDomains() returns HSTR_RESULT_SUCCESS. Otherwise, it returns one of the following errors:

- HSTR_RESULT_NOT_INITIALIZED if hStreams had not been initialized properly.
- HSTR_RESULT_NULL_PTR, if out_pNumPhysDomains is NULL.
- HSTR_RESULT_NULL_PTR, if out_pNumActivePhysDomains is NULL.
- HSTR_RESULT_NULL_PTR, if out_pHomogeneous is NULL.

Thread safety:

Thread safe.

7.7.1.7 HSTR_RESULT hStreams_GetPhysDomainDetails (HSTR_PHYS_DOM *in_PhysDomain*, uint32_t * *out_pNumThreads*, HSTR_ISA_TYPE * *out_pISA*, uint32_t * *out_pCoreMaxMHz*, HSTR_CPU_MASK *out_MaxCPUmask*, HSTR_CPU_MASK *out_AvoidCPUmask*, uint64_t * *out_pSupportedMemTypes*, uint64_t *out_pPhysicalBytesPerMemType*[HSTR_MEM_TYPE_SIZE])

Returns information about specified physical domain.

This API covers only static information about a single physical domain. hStreams_GetAvailable reveals unused threads for a given physical domain.

Parameters

in_PhysDomain [in] Physical domain ID: HSTR_SRC_PHYS_DOMAIN for source, remote domains start at 0

out_pNumThreads [out] Number of threads (not cores) on each domain; 0 means not initialized

out_pISA [out] ISA type Anticipated improvement: provide a routine to convert this to text

out_pCoreMaxMHz [out] Maximum core frequence, in MHz

out_MaxCPUmask [out] hStreams cpu mask covering all available threads on each domain

out_AvoidCPUmask [out] hStreams cpu mask covering all threads that are normally reserved for OS use. The nominally-available recommended CPU mask is the XOR of MaxCPUmax and AvoidCPUmask.

out_pSupportedMemTypes [out] bit array mask of memory types supported for this physical domain, That is, bit 0 is HSTR_MEM_TYPE_NORMAL, bit 1 is HSTR_MEM_TYPE_HBW, etc.

out_pPhysicalBytesPerMemType [out] array of physical sizes, in bytes, of each memory type, in this physical domain The size of the array passed in must be HSTR_MEM_TYPE_SIZE The array elements correspond to the enumeration in HSTR_MEM_TYPE The value is 0 for memory types that are defined but unsupported on that domain

Returns

If successful, `hStreams_GetPhysDomainDetails()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.
- `HSTR_RESULT_DOMAIN_OUT_OF_RANGE` if `in_PhysDomain` is out of range or provided physical domain is inactive.
- `HSTR_RESULT_NULL_PTR`, if an out_pNumThreads is `NULL`.
- `HSTR_RESULT_NULL_PTR`, if an out_pISA is `NULL`.
- `HSTR_RESULT_NULL_PTR`, if an out_pCoreMaxMHz is `NULL`.
- `HSTR_RESULT_NULL_PTR`, if an out_pSupportedMemTypes is `NULL`.
- `HSTR_RESULT_NULL_PTR`, if an out_pPhysicalBytesPerMemType is `NULL`.

Thread safety:

Thread safe.

7.7.1.8 HSTR_RESULT hStreams_RmLogDomains (uint32_t *in_NumLogDomains*, HSTR_LOG_DOM * *in_pLogDomainIDs*)

Remove logical domains.

This function will remove listed logical domains from their physical domains, make their logical domain IDs invalid and remove any associated logical streams.

Note

A call to `hStreams_RmLogDomains()` will block until the streams from all logical domains listed in the `in_pLogDomainIDs` array have processed all the enqueued actions.

Parameters

in_NumLogDomains [in] Number of entries in the `in_pLogDomainIDs` array

in_pLogDomainIDs [in] Array of IDs of logical domains to be removed

Returns

If successful, `hStreams_RmLogDomains()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if the library had not been initialized properly
- `HSTR_RESULT_NULL_PTR` if `in_pLogDomainIDs` is NULL
- `HSTR_RESULT_NOT_FOUND` if at least one of the logical domain IDs doesn't correspond to a valid logical domain
- `HSTR_RESULT_OUT_OF_RANGE` if `in_NumLogDomains` equals 0

Thread safety:

Thread safe.

7.8 hStreams Source - Stream management

Functions

- HSTR_RESULT hStreams_StreamCreate (HSTR_LOG_STR in_LogStreamID, HSTR_LOG_DOM in_LogDomainID, const HSTR_CPU_MASK in_CPUmask)

  Register a logical stream and specify its domain and CPU mask.

- HSTR_RESULT hStreams_StreamDestroy (HSTR_LOG_STR in_LogStreamID)

  Destroy a logical stream.

- HSTR_RESULT hStreams_GetNumLogStreams (HSTR_LOG_DOM in_LogDomainID, uint32_t *out_pNumLogStreams)

  Return number of logical streams associated with a logical domain.

- HSTR_RESULT hStreams_GetLogStreamIDList (HSTR_LOG_DOM in_LogDomainID, uint32_t in_NumLogStreams, HSTR_LOG_STR *out_pLogStreamIDs)

  Returns list of logical streams attached to provided logical domain.

- HSTR_RESULT hStreams_GetLogStreamDetails (HSTR_LOG_STR in_LogStreamID, HSTR_LOG_DOM in_LogDomainID, HSTR_CPU_MASK out_CPUmask)

  Returns cpu mask assigned to provided logical stream.

7.8.1 Function Documentation

7.8.1.1 HSTR_RESULT hStreams_GetLogStreamDetails (HSTR_LOG_STR *in_LogStreamID*, HSTR_LOG_DOM *in_LogDomainID*, HSTR_CPU_MASK *out_CPUmask*)

Returns cpu mask assigned to provided logical stream.

Parameters

in_LogSstreamID [in] ID of logical stream to look up

in_LogDomainID [in] ID of logical domain to look up

out_CPUmask [out] cpu mask associated with provided logical stream

Returns

If successful, `hStreams_GetLogStreamDetails()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.
- `HSTR_RESULT_NOT_FOUND` if `in_LogStreamID` is out of range

Thread safety:

Thread safe.

7.8.1.2 HSTR_RESULT hStreams_GetLogStreamIDList (HSTR_LOG_DOM *in_LogDomainID*, uint32_t *in_NumLogStreams*, HSTR_LOG_STR * *out_pLogStreamIDs*)

Returns list of logical streams attached to provided logical domain.

Note

Before calling `hStreams_GetLogStreamIDList` you should call `hStreams_GetNumLogStreams()` function to get number of logical streams attached to provided logical domain.

Parameters

in_LogDomainID [in] ID of logical domain for which we are querying

in_NumLogStreams [in] Number of allocated by user elements inside `out_pLogStreamIDs` array

out_pLogStreamIDs [out] Array with `in_NumLogStreams` logical stream IDs

Returns

If successful, `hStreams_GetLogStreamIDList()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.
- `HSTR_RESULT_DOMAIN_OUT_OF_RANGE` if `in_LogDomainID` is out of range
- `HSTR_RESULT_NULL_PTR`, if `out_pLogStreamIDs` is `NULL`
- `HSTR_RESULT_NOT_FOUND` if fewer than `in_NumLogStreams` mappings found
- `HSTR_RESULT_INCONSISTENT_ARGS` if more than `in_NumLogStreams` mappings are found.

Thread safety:

Thread safe. Please note that subsequent calls to `hStreams_GetNumLogStreams()` and `hStreams_GetLogStreamIDList()` do not form an atomic operation and thus logical streams might have been added or removed between the calls to these APIs.

7.8.1.3 HSTR_RESULT hStreams_GetNumLogStreams (HSTR_LOG_DOM *in_LogDomainID*, uint32_t * *out_pNumLogStreams*)

Return number of logical streams associated with a logical domain.

Parameters

in_logDomainID [in] ID of logical domain to be queried

out_pNumLogStreams [out] Number of logical streams associated with `in_LogDomainID`

Returns

If successful, `hStreams_GetNumLogStreams()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- HSTR_RESULT_NOT_INITIALIZED if hStreams had not been initialized properly.
- HSTR_RESULT_NULL_PTR, if out_pNumLogStreams is NULL
- HSTR_RESULT_NOT_FOUND if in_LogDomainID is not valid

Thread safety:

Thread safe.

7.8.1.4 HSTR_RESULT hStreams_StreamCreate (HSTR_LOG_STR *in_LogStreamID*, HSTR_LOG_DOM *in_LogDomainID*, const HSTR_CPU_MASK *in_CPUmask*)

Register a logical stream and specify its domain and CPU mask.

Note

The logical stream IDs are assigned by the user and do not need to form a contiguous range. Multiple logical streams within the same logical domain are permitted to fully or partially overlap. Partially overlapping streams do not have any association between them - they simply share the same CPU resources. However, logical streams which fully overlap map to the same physical stream.

Parameters

in_LogStreamID [in] The ID of the logical stream to be created.

in_LogDomainID [in] The ID of the logical domain to create the stream in.

in_CPUmask [in] The mask describing the HW threads that are used by this logical stream.

Returns

If successful, hStreams_StreamCreate() returns HSTR_RESULT_SUCCESS. Otherwise, it returns one of the following errors:

- HSTR_RESULT_NOT_INITIALIZED if the library had not been initialized properly
- HSTR_RESULT_ALREADY_FOUND if there already exists a logical stream with ID equal to in_LogStreamID
- HSTR_RESULT_DOMAIN_OUT_OF_RANGE if in_LogDomainID doesn't correspond to a valid logical domain
- HSTR_RESULT_CPU_MASK_OUT_OF_RANGE if in_CPUmask is empty
- HSTR_RESULT_CPU_MASK_OUT_OF_RANGE if in_CPUmask doesn't fall entirely within the CPU mask for logical domain in_LogDomainID

Thread safety:

Thread safe.

7.8.1.5 HSTR_RESULT hStreams_StreamDestroy (HSTR_LOG_STR *in_LogStreamID*)

Destroy a logical stream.

Note

A call to `hStreams_StreamDestroy()` will block until all the actions thus far enqueued in the stream have completed.
If multiple logical streams map to the same physical stream (i.e. they belong to the same logical domain and their CPU masks fully overlap), only the destruction of the last logical stream from that set will result in releasing the underlying resources.

Parameters

in_LogStreamID [in] The ID of the logical stream to be destroyed

Returns

If successful, `hStreams_StreamDestroy()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if the library had not been initialized properly
- `HSTR_RESULT_NOT_FOUND` if `in_LogStreamID` doesn't correspond to a valid logical stream

Thread safety:

Thread safe.

7.9 hStreams Source - Stream usage

Functions

- HSTR_RESULT hStreams_app_xfer_memory (HSTR_LOG_STR in_LogStreamID, void *in_pWriteAddr, void *in_pReadAddr, uint64_t in_NumBytes, HSTR_XFER_DIRECTION in_XferDirection, HSTR_EVENT *out_pEvent)

  Enqueue 1-dimensional data transfers in a logical stream.

- HSTR_RESULT hStreams_GetOversubscriptionLevel (HSTR_PHYS_DOM in_PhysDomainID, uint32_t in_NumThreads, uint32_t *out_pOversubscriptionArray)

  Query the number of streams overlapping for each HW thread.

- HSTR_RESULT hStreams_EnqueueCompute (HSTR_LOG_STR in_LogStreamID, const char *in_pFunctionName, uint32_t in_numScalarArgs, uint32_t in_numHeapArgs, uint64_t *in_pArgs, HSTR_EVENT *out_pEvent, void *out_ReturnValue, uint16_t in_ReturnValueSize)

  Enqueue an execution of a user-defined function in a stream.

- HSTR_RESULT hStreams_EnqueueData1D (HSTR_LOG_STR in_LogStreamID, void *in_pWriteAddr, void *in_pReadAddr, uint64_t in_size, HSTR_XFER_DIRECTION in_XferDirection, HSTR_EVENT *out_pEvent)

  Enqueue 1-dimensional data transfers in a logical stream.

- HSTR_RESULT hStreams_EnqueueDataXDomain1D (HSTR_LOG_STR in_LogStreamID, void *in_pWriteAddr, void *in_pReadAddr, uint64_t in_size, HSTR_LOG_DOM in_destLogDomain, HSTR_LOG_DOM in_srcLogDomain, HSTR_EVENT *out_pEvent)

  Enqueue 1-dimensional data between an arbitrary domain and one of the endpoint domains of this stream.

7.9.1 Function Documentation

7.9.1.1 HSTR_RESULT hStreams_app_xfer_memory (HSTR_LOG_STR *in_LogStreamID*, void * *in_pWriteAddr*, void * *in_pReadAddr*, uint64_t *in_NumBytes*, HSTR_XFER_DIRECTION *in_XferDirection*, HSTR_EVENT * *out_pEvent*)

Enqueue 1-dimensional data transfers in a logical stream.

Note

Data transfers are permitted to execute out-of-order subject to dependence policy and the dependences previously inserted into the stream.

See also

HSTR_OPTIONS.dep_policy

Note

This API allows for memory transfers between the *source* and the *sink* logical domain, not between arbitrary sink logical domain.

See also

hStreams_EnqueueDataXDomain1D() for an API allowing transfers between arbitrary logical domains

Parameters

in_LogStreamID [in] The ID of the logical stream to insert the data transfer action in.

in_pWriteAddr [in] Source proxy pointer to the memory location to write to

in_pReadAddr [in] Source proxy pointer to the memory location to read from

in_size [in] The size, in bytes, of contiguous memory that should be copied

in_XferDirection [in] The direction in which the memory transfer should occur

out_pEvent [out] optional, the pointer for the completion event

Returns

If successful, hStreams_app_xfer_memory() returns HSTR_RESULT_SUCCESS. Otherwise, it returns one of the following errors:

- HSTR_RESULT_NOT_INITIALIZED if the library had not been initialized properly
- HSTR_RESULT_NULL_PTR if in_pWriteAddr is NULL
- HSTR_RESULT_NULL_PTR if in_pReadAddr is NULL
- HSTR_RESULT_NOT_FOUND if there is no logical stream with ID equal to in_LogStreamID
- HSTR_RESULT_NOT_FOUND if in_pWriteAddr does not fall in a buffer with a valid instantiation for the logical domain in_LogStreamID is located in.
- HSTR_RESULT_NOT_FOUND if in_pReadAddr does not fall in a buffer with a valid instantiation for the logical domain in_LogStreamID is located in.
- HSTR_RESULT_OUT_OF_RANGE if the data locations involved in the transfer fall outside of any of the buffers' boundaries.

Thread safety:

All actions enqueued through concurrent calls to hStreams_app_xfer_memory() and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.9.1.2 HSTR_RESULT hStreams_EnqueueCompute (HSTR_LOG_STR *in_LogStreamID*, const char * *in_pFunctionName*, uint32_t *in_numScalarArgs*, uint32_t *in_numHeapArgs*, uint64_t * *in_pArgs*, HSTR_EVENT * *out_pEvent*, void * *out_ReturnValue*, uint16_t *in_ReturnValueSize*)

Enqueue an execution of a user-defined function in a stream.

Places an execution of a user-defined function in the stream's internal queue. The function to be called shall be compiled and loaded to the sink process so that hStreams can locate the appropriate symbol and invoke it on the stream's sink endpoint.

Parameters

in_LogStreamID [in] ID of logical stream associated to enqueue the action in

in_pFuncName [in] Null-terminated string with name of the function to be executed

in_NumScalarArgs [in] Number of arguments to be copied by value for remote invocation

in_NumHeapArgs [in] Number of arguments which are buffer addreses to be translated to sink-side instantiations' addresses

in_pArgs [in] Array of in_NumScalarArgs+in_NumHeapArgs arguments as 64-bit unsigned integers with scalar args first and buffer args second

out_pEvent [out] pointer to event which will be signaled once the action completes

out_pReturnValue [out] pointer to host-side memory the remote invocation can asynchronously write to

in_ReturnValueSize [in] the size of the asynchronous return value memory

Returns

If successful, `hStreams_EnqueueCompute()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if the library had not been initialized properly
- `HSTR_RESULT_NOT_FOUND` if `in_LogStreamID` is not a valid logical stream ID
- `HSTR_RESULT_NOT_FOUND` if at least one of the buffer arguments is not in a buffer that had been instantiated for `in_LogStreamID's` logical domain
- `HSTR_RESULT_BAD_NAME` if `in_pFunctionName` is `NULL`
- `HSTR_RESULT_BAD_NAME` if no symbol named `in_pFunctionName` is found on the streams's sink endpoint
- `HSTR_RESULT_BAD_NAME` if `in_pFunctionName` is longer than `HSTR_MAX_FUNC_NAME_SIZE`
- `HSTR_RESULT_OUT_OF_RANGE` if `in_ReturnValueSize` exceeds `HSTR_RETURN_SIZE_LIMIT`
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_ReturnValueSize != 0` and `in_pReturnValue` is `NULL` or `in_ReturnValueSize == 0` and `in_pReturnValue` is not `NULL`
- `HSTR_RESULT_TOO_MANY_ARGS` if `in_NumScalarArgs + in_NumHeapArgs > HSTR_ARGS_SUPPORTED`
- `HSTR_RESULT_NULL_PTR in_numScalarArgs + in_numHeapArgs > 0` but `in_pArgs` is `NULL`

Thread safety:

All actions enqueued through concurrent calls to `hStreams_EnqueueCompute()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.9.1.3 HSTR_RESULT hStreams_EnqueueData1D (HSTR_LOG_STR *in_LogStreamID*, void * *in_pWriteAddr*, void * *in_pReadAddr*, uint64_t *in_size*, HSTR_XFER_DIRECTION *in_XferDirection*, HSTR_EVENT * *out_pEvent*)

Enqueue 1-dimensional data transfers in a logical stream.

Note

Data transfers are permitted to execute out-of-order subject to dependence policy and the dependences previously inserted into the stream.

See also

`HSTR_OPTIONS.dep_policy`

Note

This API allows for memory transfers between the *source* and the *sink* logical domain, not between arbitrary sink logical domain.

See also

`hStreams_EnqueueDataXDomain1D()` for an API allowing transfers between arbitrary logical domains

Parameters

in_LogStreamID [in] The ID of the logical stream to insert the data transfer action in.

in_pWriteAddr [in] Source proxy pointer to the memory location to write to

in_pReadAddr [in] Source proxy pointer to the memory location to read from

in_size [in] The size, in bytes, of contiguous memory that should be copied

in_XferDirection [in] The direction in which the memory transfer should occur

out_pEvent [out] optional, the pointer for the completion event

Returns

If successful, `hStreams_EnqueueData1D()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if the library had not been initialized properly
- `HSTR_RESULT_NULL_PTR` if `in_pWriteAddr` is `NULL`
- `HSTR_RESULT_NULL_PTR` if `in_pReadAddr` is `NULL`
- `HSTR_RESULT_NOT_FOUND` if there is no logical stream with ID equal to `in_LogStreamID`
- `HSTR_RESULT_NOT_FOUND` if `in_pWriteAddr` does not fall in a buffer with a valid instantiation for the logical domain `in_LogStreamID` is located in.
- `HSTR_RESULT_NOT_FOUND` if `in_pReadAddr` does not fall in a buffer with a valid instantiation for the logical domain `in_LogStreamID` is located in.
- `HSTR_RESULT_OUT_OF_RANGE` if the data locations involved in the transfer fall outside of any of the buffers' boundaries.

Thread safety:

All actions enqueued through concurrent calls to `hStreams_EnqueueData1D()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.9.1.4 HSTR_RESULT hStreams_EnqueueDataXDomain1D (HSTR_LOG_STR *in_LogStreamID*, void * *in_pWriteAddr*, void * *in_pReadAddr*, uint64_t *in_size*, HSTR_LOG_DOM *in_destLogDomain*, HSTR_LOG_DOM *in_srcLogDomain*, HSTR_EVENT * *out_pEvent*)

Enqueue 1-dimensional data between an arbitrary domain and one of the endpoint domains of this stream.

Note

Data transfers are permitted to execute out-of-order subject to dependence policy and the dependences previously inserted into the stream.

See also

`HSTR_OPTIONS.dep_policy`

Parameters

in_LogStreamID [in] The ID of the logical stream to insert the data transfer action in.

in_pWriteAddr [in] Source proxy pointer to the memory location to write to

in_pReadAddr [in] Source proxy pointer to the memory location to read from

in_size [in] The size, in bytes, of contiguous memory that should be copied

in_destLogDomain [in] The logical domain to which to transfer the data

in_srcLogDomain [in] The logical domain from which to transfer the data

out_pEvent [out] optional, the pointer for the completion event

Returns

If successful, `hStreams_EnqueueDataXDomain1D()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if the library had not been initialized properly
- `HSTR_RESULT_NULL_PTR` if `in_pWriteAddr` is `NULL`
- `HSTR_RESULT_NULL_PTR` if `in_pReadAddr` is `NULL`
- `HSTR_RESULT_NOT_FOUND` if there is no logical stream with ID equal to `in_LogStreamID`
- `HSTR_RESULT_NOT_FOUND` if `in_pWriteAddr` does not fall in a buffer with a valid instantiation for the logical domain `in_LogStreamID` is located in.
- `HSTR_RESULT_NOT_FOUND` if `in_pReadAddr` does not fall in a buffer with a valid instantiation for the logical domain `in_LogStreamID` is located in.
- `HSTR_RESULT_OUT_OF_RANGE` if the data locations involved in the transfer fall outside of any of the buffers' boundaries.
- `HSTR_RESULT_OVERLAPPING_RESOURCES` if transfer is enqueued within a buffer that has the aliased buffer property set, source and destination logical domains belong to one physical domain and the source and destination regions partial overlap.

Thread safety:

All actions enqueued through concurrent calls to `hStreams_EnqueueDataXDomain1D()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.9.1.5 HSTR_RESULT hStreams_GetOversubscriptionLevel (HSTR_PHYS_DOM *in_PhysDomainID*, uint32_t *in_NumThreads*, uint32_t * *out_pOversubscriptionArray*)

Query the number of streams overlapping for each HW thread.

Given a physical domain ID, fill an array with oversubscription level for each HW thread. `in_NumThreads` must be equal to the number of HW threads existing on queried physical domain. The number of HW threads can be obtained from `hStreams_GetPhysDomainDetails` by checking the output parameter `out_pNumThreads`.

Parameters

in_PhysDomainID [in] ID of the physical domain to query. Enumerated physical domain IDs can be `HSTR_SRC_PHYS_DOMAIN` or 0 and higher for non-source physical domains.

in_NumThreads [in] The number of entries in the `out_pOversubscriptionArray` array

out_pOversubscriptionArray [out] The array of elements describing a number of streams using a HW thread corresponding to this element index.

Returns

If successful, `hStreams_GetOversubscriptionLevel()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if the library had not been initialized properly
- `HSTR_RESULT_NULL_PTR` if `out_pOversubscriptionArray` is `NULL`.
- `HSTR_RESULT_NOT_FOUND` if `in_NumThreads` is not equal to the number of HW threads existing on the queried physical domain.

Thread safety:

Thread safe for operations of different output arrays.

7.10 hStreams Source - Sync

Functions

- HSTR_RESULT hStreams_StreamSynchronize (HSTR_LOG_STR in_LogStreamID)

  Block until all the operation enqueued in a stream have completed.

- HSTR_RESULT hStreams_ThreadSynchronize ()

  Block until all the operation enqueued in all the streams have completed.

- HSTR_RESULT hStreams_EventWait (uint32_t in_NumEvents, HSTR_EVENT *in_pEvents, bool in_WaitForAll, int32_t in_TimeOutMilliSeconds, uint32_t *out_pNumSignaled, uint32_t *out_pSignaledIndices)

  Wait on a set of events.

- HSTR_RESULT hStreams_EventStreamWait (HSTR_LOG_STR in_LogStreamID, uint32_t in_NumEvents, HSTR_EVENT *in_pEvents, int32_t in_NumAddresses, void **in_pAddresses, HSTR_EVENT *out_pEvent)

  Aggregate multiple dependences into one event handle and optionally insert that event handle into a logical stream.

7.10.1 Function Documentation

7.10.1.1 HSTR_RESULT hStreams_EventStreamWait (HSTR_LOG_STR *in_LogStreamID*, uint32_t *in_NumEvents*, HSTR_EVENT * *in_pEvents*, int32_t *in_NumAddresses*, void ** *in_pAddresses*, HSTR_EVENT * *out_pEvent*)

Aggregate multiple dependences into one event handle and optionally insert that event handle into a logical stream.

The resulting aggregation of multiple dependences is composed of depenences on:

- last actions actions enqueued in the stream which involve buffers specified in the `in_pAddresses` array
- explicit completion events specified in the in_pEvents array The output event handle, `out_pEvent` will represent that aggregation of dependences.

The special pair of values `(-1, NULL)` for `(in_NumAddresses, in_pAddresses)` is used to signify that the dependence to be created should not be inserted into the stream, only an event handle representing that dependence should be returned.

The special pair of values `(0, NULL)` for `(in_NumAddresses, in_pAddresses)` is used to signify that the dependence to be created should not involve any actions related to buffer.

On the other hand, the special pair of values `(0, NULL)` for `(in_NumEvents, in_pEvents)` is used to signify that the dependence to be inserted should include all the actions previously enqueued in the specified stream.

Parameters

in_LogStreamID [in] ID of the logical stream from which take the dependencies and into which to optionally insert the dependence.

in_NumEvents [in] Number of entries in the `in_pEvents` array.

in_pEvents [in] An array of event handles that should be included in the aggregated dependence.

in_NumAddresses [in] Number of entries in the in_pAddresses array.

in_pAddresses [in] Array of source-side proxy addresses to be mapped to buffers, on which the dependencies will be computed for aggregation.

out_pEvent [out] the aggreagated completion event. If no handle is needed, set this to NULL

Returns

If successful, `hStreams_EventStreamWait()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.
- `HSTR_RESULT_NOT_FOUND` if a logical stream with ID `in_LogStreamID` doesn't exist
- `HSTR_RESULT_NOT_FOUND` if at least one entry in the `in_pAddresses` array does not correspond to a buffer with an instantiation in the logical domain in which the `in_LogStreamID` stream is located is located.
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_NumAddresses` is 0 or -1 and `in_pAddresses` is not NULL and vice versa.
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_NumEvents` is 0 and `in_pEvents` is not NULL and vice versa.

Thread safety:

All actions enqueued through concurrent calls to `hStreams_EventStreamWait()` and any other function that enqueues actions into the same stream are guaranteed to be correctly inserted into the stream's queue, although in an unspecified order. Therefore, concurrent calls to these functions which operate on the same data will produce undefined results.

7.10.1.2 HSTR_RESULT hStreams_EventWait (uint32_t *in_NumEvents*, HSTR_EVENT * *in_pEvents*, bool *in_WaitForAll*, int32_t *in_TimeOutMilliSeconds*, uint32_t * *out_pNumSignaled*, uint32_t * *out_pSignaledIndices*)

Wait on a set of events.

Parameters

in_NumEvents [in] number of event pointers in the array

in_pEvents [in] the array of pointers of events to be waited on

in_WaitForAll [in] If true, report success only if all events signaled If false, report success if at least one event signaled

in_TimeOutMilliSeconds [in] Timeout. 0 polls and returns immediately, or HSTR_TIME_INFINITE (-1)

out_pNumSignaled [out] Number of events that signaled as completed

out_pSignaledIndices [out] Packed list of indices into in_pEvents that signaled

Returns

If successful, `hSteramsEventWait()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns on of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly.
- `HSTR_RESULT_NULL_PTR` if in_pEvents is NULL
- `HSTR_RESULT_REMOTE_ERROR` if there was a remote error, e.g. the remote process died
- `HSTR_RESULT_TIME_OUT_REACHED` if the time out was reached or the timeout is zero and the event has not been signalled.
- `HSTR_RESULT_EVENT_CANCELED` if the event was cancelled or the process died
- `HSTR_RESULT_OUT_OF_RANGE` if in_NumEvents == 0
- `HSTR_RESULT_INCONSISTENT_ARGS` if in_NumEvents != 1 && !in_WaitForAll and either of out_pNumSignaled or out_pSignaledIndices are NULL

Thread safety:

Thread safe.

7.10.1.3 HSTR_RESULT hStreams_StreamSynchronize (HSTR_LOG_STR *in_LogStreamID*)

Block until all the operation enqueued in a stream have completed.

Parameters

in_LogStreamID [in] ID of the logical stream

Returns

If successful, `hStreams_StreamSynchronize()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly
- `HSTR_RESULT_NOT_FOUND` if `in_LogStreamID` is not a valid logical stream ID
- `HSTR_RESULT_TIME_OUT_REACHED` if timeout was reached while waiting on the completion of stream's actions.
- `HSTR_RESULT_EVENT_CANCELED` if one of the events in the stream was canceled
- `HSTR_RESULT_REMOTE_ERROR` if there was a remote error, e.g. the remote process died

See also

HSTR_OPTIONS.time_out_ms_val

Thread safety:

Thread safe.

7.10.1.4 HSTR_RESULT hStreams_ThreadSynchronize ()

Block until all the operation enqueued in all the streams have completed.

Returns

If successful, `hStreams_ThreadSynchronize()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized properly
- `HSTR_RESULT_TIME_OUT_REACHED` if timeout was reached while waiting on the completion of actions.
- `HSTR_RESULT_EVENT_CANCELED` if one of the events was canceled
- `HSTR_RESULT_REMOTE_ERROR` if there was a remote error, e.g. the remote process died

See also

HSTR_OPTIONS.time_out_ms_val

Thread safety:

Thread safe.

7.11 hStreams Source - Memory management

Functions

- HSTR_RESULT hStreams_Alloc1DEx (void *in_BaseAddress, uint64_t in_Size, HSTR_BUFFER_PROPS *in_pBufferProps, int64_t in_NumLogDomains, HSTR_LOG_DOM *in_pLogDomainIDs)

  Allocate 1-dimensional buffer with additional properties.

- HSTR_RESULT hStreams_AddBufferLogDomains (void *in_Address, uint64_t in_NumLogDomains, HSTR_LOG_DOM *in_pLogDomainIDs)

  Create instances of the buffer in the logical domains specified as parameters.

- HSTR_RESULT hStreams_RmBufferLogDomains (void *in_Address, int64_t in_NumLogDomains, HSTR_LOG_DOM *in_pLogDomainIDs)

  Deallocate buffer instantiations in the selected logical domains.

- HSTR_RESULT hStreams_DeAlloc (void *in_Address)

  Destroy the buffer and remove all its instantiations.

- HSTR_RESULT hStreams_GetBufferNumLogDomains (void *in_Address, uint64_t *out_pNumLogDomains)

  Return the number of logical domains or which the buffer has been instantiated.

- HSTR_RESULT hStreams_GetBufferLogDomains (void *in_Address, uint64_t in_NumLogDomains, HSTR_LOG_DOM *out_pLogDomains, uint64_t *out_pNumLogDomains)

  Return a list of logical domains for which the buffer is instantiated.

- HSTR_RESULT hStreams_GetBufferProps (void *in_Address, HSTR_BUFFER_PROPS *out_BufferProps)

  Returns buffer properties associated with a buffer.

7.11.1 Function Documentation

7.11.1.1 HSTR_RESULT hStreams_AddBufferLogDomains (void * *in_Address*, uint64_t *in_NumLogDomains*, HSTR_LOG_DOM * *in_pLogDomainIDs*)

Create instances of the buffer in the logical domains specified as parameters.

Parameters

in_Address [in] Any address inside a buffer for which to create instantiations of the buffer.

in_NumLogDomains [in] Number of logical domains for which to create instantiations of the buffer.

in_pLogDomainIDs [in] Array of logical domains IDs for which to instantiate the buffer. The array must not contain `HSTR_SRC_LOG_DOMAIN`.

Returns

If successful, hStreams_AddBufferLogDomains() returns HSTR_RESULT_SUCCESS. Otherwise, it returns one of the following errors:

- HSTR_RESULT_NOT_INITIALIZED if hStreams had not been initialized prior to this call.
- HSTR_RESULT_NULL_PTR if in_Address is NULL
- HSTR_RESULT_NULL_PTR if in_pLogDomainIDs is NULL
- HSTR_RESULT_OUT_OF_RANGE if in_NumLogDomains == 0
- HSTR_RESULT_INCONSISTENT_ARGS if in_pLogDomainIDs array contains duplicate entries.
- HSTR_RESULT_DOMAIN_OUT_OF_RANGE if at least one entry in the in_pLogDomainIDs array is not a valid logical domain ID.
- HSTR_NOT_FOUND if in_Address does not belong to any buffer.
- HSTR_RESULT_ALREADY_FOUND if in_pLogDomainIDs array contains at least one ID of a logical domain for which the buffer is already instantiated.
- HSTR_RESULT_OUT_OF_MEMORY if at least one of the instantiations cannot be created due to memory exhaustion or unavailability of requested memory kind, subject to the buffer's memory allocation policy specified in the buffer's properties

Thread safety:

Thread safe.

7.11.1.2 HSTR_RESULT hStreams_Alloc1DEx (void * *in_BaseAddress*, uint64_t *in_Size*, HSTR_BUFFER_PROPS * *in_pBufferProps*, int64_t *in_NumLogDomains*, HSTR_LOG_DOM * *in_pLogDomainIDs*)

Allocate 1-dimensional buffer with additional properties.

This API is an extended version of hStreams_Alloc1D(), allowing for controlling the properties of the buffer as well as instantiation of the buffer on only selected logical domains.

As with hStreams_Alloc1D(), the buffer must be created from existing source memory, so as to have a source proxy address by which to identify it. The resulting buffer is always instantiated for the special logical domain HSTR_SRC_LOG_DOMAIN. Note that the contents of those instatiations are considered to be undefined, i.e. the contents of the buffer are not implicitly synchronized across all the instantations.

If NULL is supplied to the buffer properties argument (in_pBufferProps), default properties will be used. See HSTR_BUFFER_PROPS documentation for details on what the default properties are.

If 0 is supplied as the number of logical domains (in_NumLogDomains) for which to instantiate the buffer, in_pLogDomainIDs must be NULL - the buffer will not be instantiated for any logical domain except HSTR_SRC_LOG_DOMAIN.

If -1 is supplied as the number of logical domains (in_NumLogDomains) for which to instantiate the buffer, in_pLogDomainIDs must be NULL - the buffer will be instantiated for all logical domains already present.

Parameters

in_BaseAddress [in] pointer to the beginning of the memory in the source logical domain

in_Size [in] size of the memory to create the buffer for, in bytes

in_pBufferProps [in] Buffer properties, optional.

in_NumLogDomains [in] Number of logical domains to instantiate the buffer in.

in_pLogDomainIDs [in] Array of logical domains IDs for which to instantiate the buffer. The array must not contain HSTR_SRC_LOG_DOMAIN.

Returns

If successful, `hStreams_Alloc1DEx()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized prior to this call.
- `HSTR_RESULT_OUT_OF_RANGE` if `in_Size == 0`
- `HSTR_RESULT_OUT_OF_RANGE` if `in_NumLogDomains < -1`
- `HSTR_RESULT_OUT_OF_RANGE` if `in_pBufferProps` contains an invalid value.
- `HSTR_RESULT_NULL_PTR` if `in_BaseAddress` is `NULL`.
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_NumLogDomains == -1 && in_pLogDomainIDs != NULL`
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_NumLogDomains == 0 && in_pLogDomainIDs != NULL`
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_pLogDomainIDs` array contains duplicate entries.
- `HSTR_RESULT_DOMAIN_OUT_OF_RANGE` if at least one entry in the `in_pLogDomainIDs` array is not a valid logical domain ID.
- `HSTR_RESULT_ALREADY_FOUND` if a buffer had been previously created with the same memory.
- `HSTR_RESULT_OVERLAPPING_RESOURCES` if the memory range partially overlaps one or more buffers which had been created previously.
- `HSTR_RESULT_OUT_OF_MEMORY` if one or more of the buffer's instatiations cannot be created due to the sink domain's memory exhaustion or unavailability of the requested memory kind, subject to memory allocation policy specified in the buffer's properties.
- `HSTR_RESULT_NOT_IMPLEMENTED` if requested memory type is other than `HSTR_MEM_TYPE_NORMAL`.
- `HSTR_RESULT_NOT_IMPLEMENTED` if `HSTR_BUF_PROP_AFFINITIZED` flag is set in the buffer's properties.

Thread safety:

Thread safe.

7.11.1.3 HSTR_RESULT hStreams_DeAlloc (void * *in_Address*)

Destroy the buffer and remove all its instantiations.

Parameters

in_Address [in] Any address inside a buffer to destroy.

Returns

If successful, `hStreams_DeAlloc()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized prior to this call.
- `HSTR_RESULT_NULL_PTR` if `in_Address` is `NULL`.
- `HSTR_RESULT_NOT_FOUND` if `in_Address` doesn't refer to a valid buffer.

Thread safety:

Thread safe.

7.11.1.4 HSTR_RESULT hStreams_GetBufferLogDomains (void * *in_Address*, uint64_t *in_NumLogDomains*, HSTR_LOG_DOM * *out_pLogDomains*, uint64_t * *out_pNumLogDomains*)

Return a list of logical domains for which the buffer is instantiated.

< what happens when buffer too small >

Parameters

in_Address [in] Source proxy address anywhere in a buffer.

in_NumLogDomains [in] Number of entries to write to the out_pLogDomains array.

out_pLogDomains [out] List of logical domains' IDs the buffer is instantiated. At most in_NumLogDomains entries will be written.

out_pNumLogDomains [out] Number of logical domains the buffer has been instantiated for. This doesn't include the implicit instantiation for `HSTR_SRC_LOG_DOMAIN`.

Returns

If successful, `hStreams_GetBufferNumLogDomains()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized prior to this call.
- `HSTR_RESULT_NULL_PTR` if `in_Address` is `NULL`.
- `HSTR_RESULT_NULL_PTR` if `out_pLogDomains` is `NULL`.
- `HSTR_RESULT_NULL_PTR` if `out_pNumLogDomains` is `NULL`.
- `HSTR_RESULT_NOT_FOUND` if `in_Address` doesn't refer to a valid buffer.
- `HSTR_RESULT_OUT_OF_RANGE` if `in_NumLogDomains == 0`.
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_NumLogDomains` is less than the actual number of logical domains where buffer is instantiated, not counting `HSTR_SRC_LOG_DOMAIN`.

Thread safety:

Thread safe.

7.11.1.5 HSTR_RESULT hStreams_GetBufferNumLogDomains (void * *in_Address*, uint64_t * *out_pNumLogDomains*)

Return the number of logical domains or which the buffer has been instantiated.

Note

The value written to `out_pNumLogDomains` does not count the implicit instantiation of the buffer for `HSTR_SRC_LOG_DOMAIN`.

Parameters

in_Address [in] Any address inside a buffer for which to create instantiations of the buffer.

out_pNumLogDomains [out] Number of logical domains the buffer has been instantiated for.

Returns

If successful, `hStreams_GetBufferNumLogDomains()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized prior to this call.
- `HSTR_RESULT_NULL_PTR` if `in_Address` is `NULL`.
- `HSTR_RESULT_NULL_PTR` if `out_pNumLogDomains` is `NULL`.
- `HSTR_RESULT_NOT_FOUND` if `in_Address` doesn't refer to a valid buffer.

Thread safety:

Thread safe.

7.11.1.6 HSTR_RESULT hStreams_GetBufferProps (void * *in_Address*, HSTR_BUFFER_PROPS * *out_BufferProps*)

Returns buffer properties associated with a buffer.

This API returns data for buffers of all types, including those not allocated with Alloc1Ex.

Parameters

in_Address [in] Source proxy address anywhere in a buffer.

out_BufferProps [out] Buffer properties.

Returns

If successful, `hStreams_GetBufferProps()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if `hStreams` had not been initialized properly.
- `HSTR_RESULT_NULL_PTR` if `in_Address` is `NULL`.
- `HSTR_RESULT_NULL_PTR` if `out_BufferProps` is `NULL`.
- `HSTR_RESULT_NOT_FOUND` if `in_Address` does not belong to any buffer.

Thread safety:

Thread safe.

7.11.1.7 HSTR_RESULT hStreams_RmBufferLogDomains (void * *in_Address*, int64_t *in_NumLogDomains*, HSTR_LOG_DOM * *in_pLogDomainIDs*)

Deallocate buffer instantiations in the selected logical domains.

If `-1` is suppplied as the number of logical domains, `in_pLogDomainIDs` must be `NULL` - all the instantiations of the buffer will be destroyed.

Note

Removing all instantiations of the buffer does not result in removing the logical buffer as such. That is to say, after removing all instantiations of a buffer, the user is free to add new instantiations to that buffer. In order to remove the logical buffer, a call to `hStreams_DeAlloc()` is required.

Parameters

in_Address [in] Any address inside a buffer for which to create instantiations of the buffer.

in_NumLogDomains [in] Number of logical domains from which to delete the instantiations of the buffer.

in_pLogDomainIDs [in] Array of logical domains IDs from which to delete the buffer's instantiations. The array must not contain `HSTR_SRC_LOG_DOMAIN`.

Returns

If successful, `hStreams_RmBufferLogDomains()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_INITIALIZED` if hStreams had not been initialized prior to this call.
- `HSTR_RESULT_NULL_PTR` if `in_Address` is `NULL`.
- `HSTR_RESULT_OUT_OF_RANGE` if `in_NumLogDomains == 0`
- `HSTR_RESULT_OUT_OF_RANGE` if `in_NumLogDomains < -1`
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_NumLogDomains == -1 && in_pLogDomainIDs != NULL`
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_NumLogDomains != -1 && in_pLogDomainIDs == NULL`
- `HSTR_RESULT_INCONSISTENT_ARGS` if `in_pLogDomainIDs` array contains duplicate entries.
- `HSTR_RESULT_DOMAIN_OUT_OF_RANGE` if any entry in the `in_pLogDomainIDs` array is not a valid logical domain ID.
- `HSTR_RESULT_NOT_FOUND` if `in_Address` doesn't refer to a valid buffer.
- `HSTR_RESULT_NOT_FOUND` if the buffer doesn't have an instantiation for any of the selected logical domains.

7.12 hStreams Source - Error handling

Functions

- HSTR_RESULT hStreams_GetLastError ()
  Get the last error.
- void hStreams_ClearLastError ()
  Clear the last hStreams error across.

7.12.1 Function Documentation

7.12.1.1 void hStreams_ClearLastError ()

Clear the last hStreams error across.

Returns

void

Thread safety:

Thread safe. Last error is recorded across all threads accessing the hStreams library API.

7.12.1.2 HSTR_RESULT hStreams_GetLastError ()

Get the last error.

Returns

Last recorded HSTR_RESULT (different than HSTR_SUCCESS)

Thread safety:

Thread safe. Last error is recorded across all threads accessing the hStreams library API.

7.13 hStreams Source - Configuration

Functions

- HSTR_RESULT hStreams_Cfg_SetLogLevel (HSTR_LOG_LEVEL in_loglevel)
  Set a logging level for the hetero-streams library.
- HSTR_RESULT hStreams_Cfg_SetLogInfoType (uint64_t in_info_type_mask)
  Set a bitmask of message categories that the library should emit.
- HSTR_RESULT hStreams_Cfg_SetMKLInterface (HSTR_MKL_INTERFACE in_MKLInterface)
  Choose used MKL interface version.
- HSTR_RESULT hStreams_SetOptions (const HSTR_OPTIONS *in_options)
  Configure user parameters by setting hStreams Options.
- HSTR_RESULT hStreams_GetCurrentOptions (HSTR_OPTIONS *pCurrentOptions, uint64_t buffSize)
  Query user parameters by getting hStreams Options.

7.13.1 Function Documentation

7.13.1.1 HSTR_RESULT hStreams_Cfg_SetLogInfoType (uint64_t *in_info_type_mask*)

Set a bitmask of message categories that the library should emit.

Parameters

in_info_type_mask [in] A bitmask filter to apply to the logging messages.

Hetero-streams logging mechanism categorises the messages that it can produce. User can apply a filter to the messages that will be produced, based on the message's information type. This filter takes the form of a bitmask with meaning of individual bits defined by the values of the `HSTR_INFO_TYPE` enumerated type.

Note

Adjusting the message filter is only permitted *outside* the intialization-finalization cycle for the hetero-streams library. A value that is set before the first call to any of the intialization functions is used until the finalization of the library.

Returns

If successful, `hStreams_Cfg_SetLogInfoType()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_PERMITTED` if the hetero-streams library has been already initialized

Thread safety:

Not thread safe.

7.13.1.2 HSTR_RESULT hStreams_Cfg_SetLogLevel (HSTR_LOG_LEVEL *in_loglevel*)

Set a logging level for the hetero-streams library.

Parameters

in_loglevel [in] The level at which to start reporting messages

Note

Adjusting the logging level is only permitted *outside* the intialization-finalization cycle for the hetero-streams library. A value that is set before the first call to any of the intialization functions is used until the finalization of the library.

Returns

If successful, `hStreams_Cfg_SetLogLevel()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_PERMITTED` if the hetero-streams library has been already initialized
- `HSTR_RESULT_OUT_OF_RANGE` if the input argument value does not correpond to a valid logging level

Thread safety:

Not thread safe.

7.13.1.3 HSTR_RESULT hStreams_Cfg_SetMKLInterface (HSTR_MKL_INTERFACE *in_MKLInterface*)

Choose used MKL interface version.

Returns

If successful, `hStreams_Cfg_SetMKLInterface()` returns `HSTR_RESULT_SUCCESS`. Otherwise, it returns one of the following errors:

- `HSTR_RESULT_NOT_PERMITTED` if the hetero-streams library has been already initialized
- `HSTR_RESULT_OUT_OF_RANGE` if the input argument value does not correpond to a valid MKL interface

Thread safety:

Not thread safe.

7.13.1.4 HSTR_RESULT hStreams_GetCurrentOptions (HSTR_OPTIONS * *pCurrentOptions*, uint64_t *buffSize*)

Query user parameters by getting hStreams Options.

hStreams_GetCurrentOptions() copies the current collection of hStreams options from the hStreams library to the buffer provided in a thread safe manner.

Parameters

pCurrentOptions [out] The buffer that will receive the current options from the hStreams library. Must be non-NULL or hStreams_GetCurrentOptions() returns HSTR_RESULT_OUT_OF_RANGE.

buffSize [in] Indicates the size of the buffer that pCurrentOptions points to. Must be greater than or equal to sizeof(HSTR_OPTIONS) of hStreams_GetCurrentOptions() returns HSTR_RESULT_OUT_OF_RANGE.

For more details about HSTR_OPTIONS please go directly to HSTR_OPTIONS documentation

Returns

HSTR_RESULT_SUCCESS if getting options is successful
HSTR_RESULT_OUT_OF_RANGE if pCurrentOptions is NULL, or buffSize is less than sizeof(HSTR_OPTIONS)

Thread safety:

Thread safe.

7.13.1.5 HSTR_RESULT hStreams_SetOptions (const HSTR_OPTIONS * *in_options*)

Configure user parameters by setting hStreams Options.

Parameters

in_options - see the definition of HSTR_OPTIONS [in] HSTR_OPTIONS

For more details about HSTR_OPTIONS please go directly to HSTR_OPTIONS documentation

Returns

HSTR_RESULT_SUCCESS if setting options is successful
HSTR_RESULT_NULL_PTR if in_options is NULL.
HSTR_RESULT_INCONSISTENT_ARGS if options are inconsistent.

Thread safety:

Thread safe.

7.14 hStreams Utilities

Modules

- CPU_MASK manipulating

Functions

- HSTR_RESULT hStreams_GetVersionStringLen (uint32_t *out_pVersionStringLen)
  Report the length of the version string, including the null termination character.
- HSTR_RESULT hStreams_Version (char *buff, uint32_t buffLength)
  Report hStreams version info to buffer.
- const char * hStreams_ResultGetName (HSTR_RESULT in_olr)
  Get HSTR_RESULT name.

7.14.1 Function Documentation

7.14.1.1 HSTR_RESULT hStreams_GetVersionStringLen (uint32_t * *out_pVersionStringLen*)

Report the length of the version string, including the null termination character.

Parameters

out_pVersionStringLen [out] The length of the version string, including the null termination character

Returns

HSTR_RESULT_NULL_PTR if out_pVersionStringLen is NULL
HSTR_RESULT_SUCCESS

Thread safety:

Thread safe.

7.14.1.2 const char* hStreams_ResultGetName (HSTR_RESULT *in_olr*)

Get HSTR_RESULT name.

Parameters

in_olr [in] HSTR_RESULT

Returns

Name string

Thread safety:

Thread safe.

7.14.1.3 HSTR_RESULT hStreams_Version (char * *buff*, uint32_t *buffLength*)

Report hStreams version info to buffer.

Parameters

buff [out] The buffer that will receive the version of the hStreams library.

buffLength [in] The length of the buff parameter. hStreamsVersion() copies version data upto buffLength bytes to buff.

Returns

HSTR_RESULT_SUCCESS
HSTR_RESULT_NULL_PTR if the buffer argument is a NULL pointer
HSTR_RESULT_BUFF_TOO_SMALL when the buffer is too small.

Thread safety:

Thread safe for invocations with different buffer arguments.

The version string is in format MAJOR.MINOR[.MICRO]. The MICRO-part is ommited if MICRO == 0.

7.15 CPU_MASK manipulating

Functions

- static uint64_t HSTR_CPU_MASK_ISSET (int bitNumber, const HSTR_CPU_MASK cpu_mask)

  Roughly equivalent to CPU_ISSET().

- static void HSTR_CPU_MASK_SET (int bitNumber, HSTR_CPU_MASK cpu_mask)

  Roughly equivalent to CPU_SET().

- static void HSTR_CPU_MASK_ZERO (HSTR_CPU_MASK cpu_mask)

  Roughly equivalent to CPU_ZERO().

- static void HSTR_CPU_MASK_AND (HSTR_CPU_MASK dst, const HSTR_CPU_MASK src1, const HSTR_CPU_MASK src2)

  Roughly equivalent to CPU_AND().

- static void HSTR_CPU_MASK_XOR (HSTR_CPU_MASK dst, const HSTR_CPU_MASK src1, const HSTR_CPU_MASK src2)

  Roughly equivalent to CPU_XOR().

- static void HSTR_CPU_MASK_OR (HSTR_CPU_MASK dst, const HSTR_CPU_MASK src1, const HSTR_CPU_MASK src2)

  Roughly equivalent to CPU_OR().

- static int HSTR_CPU_MASK_COUNT (const HSTR_CPU_MASK cpu_mask)

  Roughly equivalent to CPU_COUNT().

- static int HSTR_CPU_MASK_EQUAL (const HSTR_CPU_MASK cpu_mask1, const HSTR_CPU_MASK cpu_mask2)

  Roughly equivalent to CPU_EQUAL().

- static void HSTR_CPU_MASK_XLATE (HSTR_CPU_MASK dest, const cpu_set_t *src)

  Utility function to translate from cpu_set to COI_CPU_MASK.*

- static void HSTR_CPU_MASK_XLATE_EX (cpu_set_t *dest, const HSTR_CPU_MASK src)

  Utility function to translate from COI_CPU_MASK to cpu_set.*

7.15.1 Detailed Description

Functions used for manipulating HSTR_CPU_MASK.

7.15.2 Function Documentation

7.15.2.1 static void HSTR_CPU_MASK_AND (HSTR_CPU_MASK *dst*, const HSTR_CPU_MASK *src1*, const HSTR_CPU_MASK *src2*) [inline, static]

Roughly equivalent to CPU_AND().

7.15.2.2 static int HSTR_CPU_MASK_COUNT (const HSTR_CPU_MASK *cpu_mask*) [inline, static]

Roughly equivalent to CPU_COUNT().

7.15.2.3 static int HSTR_CPU_MASK_EQUAL (const HSTR_CPU_MASK *cpu_mask1*, const HSTR_CPU_MASK *cpu_mask2*) [inline, static]

Roughly equivalent to CPU_EQUAL().

7.15.2.4 static uint64_t HSTR_CPU_MASK_ISSET (int *bitNumber*, const HSTR_CPU_MASK *cpu_mask*) [inline, static]

Roughly equivalent to CPU_ISSET().

7.15.2.5 static void HSTR_CPU_MASK_OR (HSTR_CPU_MASK *dst*, const HSTR_CPU_MASK *src1*, const HSTR_CPU_MASK *src2*) [inline, static]

Roughly equivalent to CPU_OR().

7.15.2.6 static void HSTR_CPU_MASK_SET (int *bitNumber*, HSTR_CPU_MASK *cpu_mask*) [inline, static]

Roughly equivalent to CPU_SET().

7.15.2.7 static void HSTR_CPU_MASK_XLATE (HSTR_CPU_MASK *dest*, const cpu_set_t * *src*) [inline, static]

Utility function to translate from cpu_set* to COI_CPU_MASK.

7.15.2.8 static void HSTR_CPU_MASK_XLATE_EX (cpu_set_t * *dest*, const HSTR_CPU_MASK *src*) [inline, static]

Utility function to translate from COI_CPU_MASK to cpu_set*.

7.15.2.9 static void HSTR_CPU_MASK_XOR (HSTR_CPU_MASK *dst*, const HSTR_CPU_MASK *src1*, const HSTR_CPU_MASK *src2*) [inline, static]

Roughly equivalent to CPU_XOR().

7.15.2.10 static void HSTR_CPU_MASK_ZERO (HSTR_CPU_MASK *cpu_mask*) [inline, static]

Roughly equivalent to CPU_ZERO().

7.16 hStreams Types

Data Structures

- struct HSTR_OPTIONS
- struct HSTR_BUFFER_PROPS

Typedefs

- typedef int HSTR_OVERLAP_TYPE

  For managing overlap of cpu masks of partitions.

- typedef int HSTR_RESULT

  Type that is returned from hStream functions.

- typedef int64_t HSTR_INFO_TYPE

  Underlying type large enough to encompass any of HSTR_INFO_TYPE_VALUES.

- typedef enum HSTR_SEVERITY HSTR_SEVERITY
- typedef int32_t HSTR_LOG_LEVEL

  Underlying type large enough to encompass any of HSTR_LOG_LEVEL_VALUES.

- typedef int HSTR_DEP_POLICY

  This is the type associated with hStream dependence policies.

- typedef int HSTR_KMP_AFFINITY

  Type associated with hStream's KMP affinity policy.

- typedef int HSTR_OPENMP_POLICY

  Type associated with hStream OpenMP handling FIXME: This will be changing in the transition to support other threading runtimes.

- typedef int HSTR_MEM_TYPE

  Type associated with hStream physical memory types These are consecutive integers, NOT mask values.

- typedef int HSTR_XFER_DIRECTION

  Type associated with direction of data transfer.

- typedef int HSTR_ISA_TYPE

  This type encapsulates the COI_ISA_TYPE, to enable building hStreams on something other than COI in the future. It is used to indicate the ISA for enumerated domains.

- typedef int HSTR_BUFFER_PROP_FLAGS

  Type associated with mask of flags for buffers.

- typedef int HSTR_MEM_ALLOC_POLICY

Type associated with hStream memory allocation policy regarding the behaviour when either: a. the requested memory type has been exhausted on some node b. the requested memory type does not even exist on some node.

- typedef int HSTR_MKL_INTERFACE
- typedef uint64_t HSTR_LOG_STR
- typedef uint32_t HSTR_LOG_DOM
- typedef int32_t HSTR_PHYS_DOM
- typedef void(* hStreams_FatalError_Prototype_Fptr)(int)
- typedef struct HSTR_OPTIONS HSTR_OPTIONS
- typedef struct HSTR_BUFFER_PROPS HSTR_BUFFER_PROPS

Enumerations

- enum HSTR_OVERLAP_TYPE_VALUES { HSTR_NO_OVERLAP = 0, HSTR_EXACT_OVERLAP, HSTR_PARTIAL_OVERLAP }

  Possible values of `HSTR_OVERLAP_TYPE`.

- enum HSTR_RESULT_VALUES {

  HSTR_RESULT_SUCCESS = 0, HSTR_RESULT_REMOTE_ERROR, HSTR_RESULT_NOT_INITIALIZED, HSTR_RESULT_NOT_FOUND,

  HSTR_RESULT_ALREADY_FOUND, HSTR_RESULT_OUT_OF_RANGE, HSTR_RESULT_DOMAIN_OUT_OF_RANGE, HSTR_RESULT_CPU_MASK_OUT_OF_RANGE,

  HSTR_RESULT_OUT_OF_MEMORY, HSTR_RESULT_INVALID_STREAM_TYPE, HSTR_RESULT_OVERLAPPING_RESOURCES, HSTR_RESULT_DEVICE_NOT_INITIALIZED,

  HSTR_RESULT_BAD_NAME, HSTR_RESULT_TOO_MANY_ARGS, HSTR_RESULT_TIME_OUT_REACHED, HSTR_RESULT_EVENT_CANCELED,

  HSTR_RESULT_INCONSISTENT_ARGS, HSTR_RESULT_BUFF_TOO_SMALL, HSTR_RESULT_MEMORY_OPERAND_INCONSISTENT, HSTR_RESULT_NULL_PTR,

  HSTR_RESULT_INTERNAL_ERROR, HSTR_RESULT_RESOURCE_EXHAUSTED, HSTR_RESULT_NOT_IMPLEMENTED, HSTR_RESULT_NOT_PERMITTED,

  HSTR_RESULT_SIZE }

  Possible values of `HSTR_RESULT`.

- enum HSTR_INFO_TYPE_VALUES {

  HSTR_INFO_TYPE_TRACE = (1ULL << 5), HSTR_INFO_TYPE_SINK_INVOKE = (1ULL << 6), HSTR_INFO_TYPE_MEM = (1ULL << 7), HSTR_INFO_TYPE_SYNC = (1ULL << 8),

  HSTR_INFO_TYPE_MISC = (1ULL << 9) }

  Message type categories.

- enum HSTR_SEVERITY
- enum HSTR_LOG_LEVEL_VALUES {

  HSTR_LOG_LEVEL_NO_LOGGING, HSTR_LOG_LEVEL_FATAL_ERROR, HSTR_LOG_LEVEL_ERROR, HSTR_LOG_LEVEL_WARN,

  HSTR_LOG_LEVEL_LOG, HSTR_LOG_LEVEL_DEBUG1, HSTR_LOG_LEVEL_DEBUG2, HSTR_LOG_LEVEL_DEBUG3,

  HSTR_LOG_LEVEL_DEBUG4 }

Message type categories.

- enum HSTR_DEP_POLICY_VALUES { HSTR_DEP_POLICY_CONSERVATIVE = 0, HSTR_DEP_POLICY_BUFFERS, HSTR_DEP_POLICY_NONE, HSTR_DEP_POLICY_SIZE }

  Possible values of `HSTR_DEP_POLICY`.

- enum HSTR_KMP_AFFINITY_VALUES { HSTR_KMP_AFFINITY_BALANCED = 0, HSTR_KMP_AFFINITY_COMPACT, HSTR_KMP_AFFINITY_SCATTER, HSTR_KMP_AFFINITY_SIZE }

  Possible values of `HSTR_KMP_AFFINITY`.

- enum HSTR_OPENMP_POLICY_VALUES { HSTR_OPENMP_ON_DEMAND = 0, HSTR_OPENMP_PRE_SETUP = 1, HSTR_OPENMP_POLICY_SIZE }

  Possible values of `HSTR_KMP_AFFINITY`.

- enum HSTR_MEM_TYPE_VALUES { HSTR_MEM_TYPE_ANY = -1, HSTR_MEM_TYPE_NORMAL = 0, HSTR_MEM_TYPE_HBW, HSTR_MEM_TYPE_SIZE }

  Possible values of `HSTR_MEM_TYPE`.

- enum HSTR_XFER_DIRECTION_VALUES { HSTR_SINK_TO_SRC = 0, HSTR_SRC_TO_SINK = 1 }

  Possible values of `HSTR_XFER_DIRECTION`.

- enum HSTR_ISA_TYPE_VALUES {
  HSTR_ISA_INVALID = 0, HSTR_ISA_x86_64, HSTR_ISA_MIC, HSTR_ISA_KNF,
  HSTR_ISA_KNC, HSTR_ISA_KNL }

  Possible values of `HSTR_ISA_TYPE`.

- enum HSTR_BUFFER_PROP_FLAGS_VALUES {
  HSTR_BUF_PROP_ALIASED = 1, HSTR_BUF_PROP_SRC_PINNED = 2, HSTR_BUF_PROP_INCREMENTAL = 4, HSTR_BUF_PROP_AFFINITIZED = 8,
  HSTR_BUF_PROP_INVALID_VALUE = 16 }

  Possible values of `HSTR_BUFFER_PROP_FLAGS`.

- enum HSTR_MEM_ALLOC_POLICY_VALUES { HSTR_MEM_ALLOC_PREFERRED = 0, HSTR_MEM_ALLOC_STRICT, HSTR_MEM_ALLOC_POLICY_SIZE }

  Possible values of `HSTR_MEM_ALLOC_POLICY`.

- enum HSTR_MKL_INTERFACE_VALUES { HSTR_MKL_LP64 = 0, HSTR_MKL_ILP64, HSTR_MKL_NONE, HSTR_MKL_INTERFACE_SIZE }

  Possible values of `HSTR_MKL_INTERFACE`.

Variables

- const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_ALWAYSEMIT = DEPRECATED_HSTR_INFO_TYPE_ALWAYSEMIT

- const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_INVOKE = DEPRECATED_HSTR_INFO_TYPE_INVOKE
- const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_DEPS = DEPRECATED_HSTR_INFO_TYPE_DEPS
- const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_GENERAL = DEPRECATED_HSTR_INFO_TYPE_GENERAL
- const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_AND16 = DEPRECATED_HSTR_INFO_TYPE_AND16
- const HSTR_SEVERITY HSTR_SEVERITY_INFO = DEPRECATED_HSTR_SEVERITY_INFO
- const HSTR_SEVERITY HSTR_SEVERITY_WARNING = DEPRECATED_HSTR_SEVERITY_WARNING
- const HSTR_SEVERITY HSTR_SEVERITY_ERROR = DEPRECATED_HSTR_SEVERITY_ERROR
- const HSTR_SEVERITY HSTR_SEVERITY_FATAL_ERROR = DEPRECATED_HSTR_SEVERITY_FATAL_ERROR

7.16.1 Typedef Documentation

7.16.1.1 typedef int HSTR_BUFFER_PROP_FLAGS

Type associated with mask of flags for buffers.

7.16.1.2 typedef struct HSTR_BUFFER_PROPS HSTR_BUFFER_PROPS

Additional properties when creating new buffer via Alloc1DEx. Default properties are:

- Prefer to use normal DRAM.
- Not enable aliasing.
- The instance associated with HSTR_SRC_LOG_DOMAIN is pinned.
- Buffer will not be instantiated for logical domains created after its allocation.

7.16.1.3 typedef int HSTR_DEP_POLICY

This is the type associated with hStream dependence policies.

7.16.1.4 typedef int64_t HSTR_INFO_TYPE

Underlying type large enough to encompass any of HSTR_INFO_TYPE_VALUES.

7.16.1.5 typedef int HSTR_ISA_TYPE

This type encapsulates the COI_ISA_TYPE, to enable building hStreams on something other than COI in the future. It is used to indicate the ISA for enumerated domains.

7.16.1.6 typedef int HSTR_KMP_AFFINITY

Type associated with hStream's KMP affinity policy.

7.16.1.7 typedef uint32_t HSTR_LOG_DOM

7.16.1.8 typedef int32_t HSTR_LOG_LEVEL

Underlying type large enough to encompass any of HSTR_LOG_LEVEL_VALUES.

7.16.1.9 typedef uint64_t HSTR_LOG_STR

HSTR_LOG_STR, HSTR_LOG_DOM and HSTR_PHYS_DOM are aliases in order to abstract the interfaces and hide the implementations from the client to stress that the implementation of this datatype can change w/o notice. It may be used for either physical or logical streams. HSTR_LOG_STR is 64 bit, so that it can map from any 64b address to a logical stream.

7.16.1.10 typedef int HSTR_MEM_ALLOC_POLICY

Type associated with hStream memory allocation policy regarding the behaviour when either: a. the requested memory type has been exhausted on some node b. the requested memory type does not even exist on some node.

7.16.1.11 typedef int HSTR_MEM_TYPE

Type associated with hStream physical memory types These are consecutive integers, NOT mask values.

7.16.1.12 typedef int HSTR_MKL_INTERFACE

7.16.1.13 typedef int HSTR_OPENMP_POLICY

Type associated with hStream OpenMP handling FIXME: This will be changing in the transition to support other threading runtimes.

7.16.1.14 typedef struct HSTR_OPTIONS HSTR_OPTIONS

This type defines functions for the hStreams library to use for error handling, and other future options.

7.16.1.15 typedef int HSTR_OVERLAP_TYPE

For managing overlap of cpu masks of partitions.

7.16.1.16 typedef int32_t HSTR_PHYS_DOM

7.16.1.17 typedef int HSTR_RESULT

Type that is returned from hStream functions.

7.16.1.18 typedef enum HSTR_SEVERITY HSTR_SEVERITY

7.16.1.19 typedef int HSTR_XFER_DIRECTION

Type associated with direction of data transfer.

7.16.1.20 typedef void(* hStreams_FatalError_Prototype_Fptr)(int)

7.16.2 Enumeration Type Documentation

7.16.2.1 enum HSTR_BUFFER_PROP_FLAGS_VALUES

Possible values of `HSTR_BUFFER_PROP_FLAGS`.

Enumerator:

HSTR_BUF_PROP_ALIASED Buffer instances should be aliased when their logical domains are mapped to the same physical domain

HSTR_BUF_PROP_SRC_PINNED The instance associated with `HSTR_SRC_LOG_DOMAIN` is pinned when buffer is created. Otherwise defer pinning until on-access demand.

HSTR_BUF_PROP_INCREMENTAL When a new logical domain is added, an instantiation of this buffer is automatically added for that log domain

HSTR_BUF_PROP_AFFINITIZED The first touch of each instantiation of this buffer is constrained to be performed by a thread that belongs to the CPU set of its logical domain Functionality of this flag is not implemented yet, `Alloc1DEx` return `HSTR_RESULT_NOT_IMPLEMENTED` if that flag is set.

HSTR_BUF_PROP_INVALID_VALUE First invalid value of bitmask. Value of last flag * 2.

7.16.2.2 enum HSTR_DEP_POLICY_VALUES

Possible values of `HSTR_DEP_POLICY`.

Enumerator:

HSTR_DEP_POLICY_CONSERVATIVE Everything submitted to an hStream depends on everything before it.

HSTR_DEP_POLICY_BUFFERS Dependendencies are based on the existence of RAW, WAW, WAE to the same buffer.

HSTR_DEP_POLICY_NONE Dependencies ignored, for perf debug testing only; must be last.

HSTR_DEP_POLICY_SIZE One past the max supported value; = to # of supported values.

7.16.2.3 enum HSTR_INFO_TYPE_VALUES

Message type categories.

Enumerator:

HSTR_INFO_TYPE_TRACE Function invocation traces.

HSTR_INFO_TYPE_SINK_INVOKE Sink-side kernel invocations.

HSTR_INFO_TYPE_MEM Memory-related messages.

HSTR_INFO_TYPE_SYNC Messages related to synchronization events.

HSTR_INFO_TYPE_MISC Miscallenous messages.

7.16.2.4 enum HSTR_ISA_TYPE_VALUES

Possible values of `HSTR_ISA_TYPE`.

Enumerator:

HSTR_ISA_INVALID Represents an invalid ISA.

HSTR_ISA_x86_64 The ISA for an x86_64 host engine.

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

HSTR_ISA_KNF ISA for L1OM devices.

HSTR_ISA_KNC ISA for K1OM devices.

HSTR_ISA_KNL

7.16.2.5 enum HSTR_KMP_AFFINITY_VALUES

Possible values of `HSTR_KMP_AFFINITY`.

Enumerator:

HSTR_KMP_AFFINITY_BALANCED Balanced Since there are as many OpenMP threads as there are HW threads reserved for the stream, this option works exactly the same as `HSTR_KMP_AFFINITY_COMPACT`.

HSTR_KMP_AFFINITY_COMPACT Compact This mode associates openmp threads to cores within the same processor first then moves to adjacent processor.

HSTR_KMP_AFFINITY_SCATTER Scatter This mode associates OpenMP threads to all assigned processors, spreading them across cores first. That is, the first OpenMP thread will be affinitized to the first available HW thread of the first core, the second OpenMP thread will be affinitized to the first available HW thread of the second core. If `N` is the number of cores assigned to a given stream, `N-th` OpenMP thread will be affinitized to the second available HW thread of the first core and so on.

HSTR_KMP_AFFINITY_SIZE One past the max supported value; = to # of supported values.

7.16.2.6 enum HSTR_LOG_LEVEL_VALUES

Message type categories.

Enumerator:

HSTR_LOG_LEVEL_NO_LOGGING
HSTR_LOG_LEVEL_FATAL_ERROR
HSTR_LOG_LEVEL_ERROR
HSTR_LOG_LEVEL_WARN
HSTR_LOG_LEVEL_LOG
HSTR_LOG_LEVEL_DEBUG1
HSTR_LOG_LEVEL_DEBUG2
HSTR_LOG_LEVEL_DEBUG3
HSTR_LOG_LEVEL_DEBUG4

7.16.2.7 enum HSTR_MEM_ALLOC_POLICY_VALUES

Possible values of `HSTR_MEM_ALLOC_POLICY`.

Enumerator:

HSTR_MEM_ALLOC_PREFERRED Try to alloc specified memory type, mem_type is treated as HSTR_MEM_TYPE_ANY when given memory type not available

HSTR_MEM_ALLOC_STRICT Alloc only specified memory type, fail when such memory is not avaliable on any of the specified logical domains.

HSTR_MEM_ALLOC_POLICY_SIZE One past the max supported value; = to # of supported values.

7.16.2.8 enum HSTR_MEM_TYPE_VALUES

Possible values of `HSTR_MEM_TYPE`.

Enumerator:

HSTR_MEM_TYPE_ANY Unspecified, could be any.

HSTR_MEM_TYPE_NORMAL Normal DRAM.

HSTR_MEM_TYPE_HBW High bandwidth DRAM.

HSTR_MEM_TYPE_SIZE One past the max supported value; = to # of supported values.

7.16.2.9 enum HSTR_MKL_INTERFACE_VALUES

Possible values of `HSTR_MKL_INTERFACE`.

Enumerator:

HSTR_MKL_LP64 Use LP64 interface of Intel(R) MKL (MKL_INT size is 32b).

HSTR_MKL_ILP64 Use ILP64 interface of Intel(R) MKL (MKL_INT size is 64b).

HSTR_MKL_NONE Don't load Intel(R) MKL libraries at all.

HSTR_MKL_INTERFACE_SIZE One past the max supported value.

7.16.2.10 enum HSTR_OPENMP_POLICY_VALUES

Possible values of `HSTR_KMP_AFFINITY`.

Enumerator:

HSTR_OPENMP_ON_DEMAND OpenMP handled entirely by user, without involvement from hStreams Warning: Unless the user does explicit affinitization, this is likely to lead to having all partitions oversubscribe the first partition that gets defined

HSTR_OPENMP_PRE_SETUP Set up OpenMP parallel region per partition and affinitize its threads upon StreamCreate.

HSTR_OPENMP_POLICY_SIZE One past the max supported value; = to # of supported values.

7.16.2.11 enum HSTR_OVERLAP_TYPE_VALUES

Possible values of `HSTR_OVERLAP_TYPE`.

Enumerator:

HSTR_NO_OVERLAP No overlap among streams: intersection is null.

HSTR_EXACT_OVERLAP Exact overlap among streams: non-null and intersection == union.

HSTR_PARTIAL_OVERLAP Partial overlap among streams: none of the above.

7.16.2.12 enum HSTR_RESULT_VALUES

Possible values of `HSTR_RESULT`.

Enumerator:

HSTR_RESULT_SUCCESS Successful completion.

HSTR_RESULT_REMOTE_ERROR A remote error (e.g. with COI) resulted in early termination.

HSTR_RESULT_NOT_INITIALIZED Results are not valid due to lack of successful initialization - may not be any MIC cards.

HSTR_RESULT_NOT_FOUND The object that an input key was being used to look up was not found. For example, the in_LogStream was not found to have a corresponding hStream object

HSTR_RESULT_ALREADY_FOUND The object that an input key was being used to look up was already found.

HSTR_RESULT_OUT_OF_RANGE The given input type does not exist or has a bad value.

HSTR_RESULT_DOMAIN_OUT_OF_RANGE The given domain does not exist.

HSTR_RESULT_CPU_MASK_OUT_OF_RANGE The given CPU mask is not within range.

HSTR_RESULT_OUT_OF_MEMORY One or more domains do not have adequate memory.

HSTR_RESULT_INVALID_STREAM_TYPE Specifying the given input type would violate an invariant.

HSTR_RESULT_OVERLAPPING_RESOURCES Resource allocation overlaps when it should not.

HSTR_RESULT_DEVICE_NOT_INITIALIZED The requested device is not available.

HSTR_RESULT_BAD_NAME Bad name, e.g. null function name.

HSTR_RESULT_TOO_MANY_ARGS Too many function arguments.

HSTR_RESULT_TIME_OUT_REACHED Event wait time out.

HSTR_RESULT_EVENT_CANCELED Event canceled.

HSTR_RESULT_INCONSISTENT_ARGS The arguments passed to an hStreams function are inconsistent.

HSTR_RESULT_BUFF_TOO_SMALL The given buffer is too small.

HSTR_RESULT_MEMORY_OPERAND_INCONSISTENT The description of a memory operand is inconsistent with past actions.

HSTR_RESULT_NULL_PTR An argument is NULL that shouldn't be.

HSTR_RESULT_INTERNAL_ERROR Internal error.

HSTR_RESULT_RESOURCE_EXHAUSTED Any resource other than memory exhausted, e.g. number of threads.

HSTR_RESULT_NOT_IMPLEMENTED Not implemented yet.

HSTR_RESULT_NOT_PERMITTED Requested operation is not allowed.

HSTR_RESULT_SIZE Dummy last entry.

7.16.2.13 enum HSTR_SEVERITY

7.16.2.14 enum HSTR_XFER_DIRECTION_VALUES

Possible values of `HSTR_XFER_DIRECTION`.

Enumerator:

HSTR_SINK_TO_SRC Sink (stream endpoint) to source (where command issued from).

HSTR_SRC_TO_SINK Source (where command issued from) to sink (stream endpoint).

7.16.3 Variable Documentation

7.16.3.1 const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_ALWAYSEMIT = DEPRECATED_HSTR_INFO_TYPE_ALWAYSEMIT

Deprecated

HSTR_INFO_TYPE_ALWAYSEMIT has been deprecated

See also

HSTR_INFO_TYPE_VALUES

7.16.3.2 const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_AND16 = DEPRECATED_HSTR_INFO_TYPE_AND16

Deprecated

HSTR_INFO_TYPE_AND16 has been deprecated

See also

HSTR_INFO_TYPE_VALUES

7.16.3.3 const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_DEPS = DEPRECATED_HSTR_INFO_TYPE_DEPS

Deprecated

HSTR_INFO_TYPE_DEPS has been deprecated

See also

HSTR_INFO_TYPE_VALUES

7.16.3.4 const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_GENERAL = DEPRECATED_HSTR_INFO_TYPE_GENERAL

Deprecated

HSTR_INFO_TYPE_GENERAL has been deprecated

See also

HSTR_INFO_TYPE_VALUES

7.16.3.5 const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_INVOKE = DEPRECATED_HSTR_INFO_TYPE_INVOKE

Deprecated

HSTR_INFO_TYPE_INVOKE has been deprecated

See also

HSTR_INFO_TYPE_VALUES

7.16.3.6 const HSTR_SEVERITY HSTR_SEVERITY_ERROR = DEPRECATED_HSTR_SEVERITY_ERROR

Deprecated

HSTR_SEVERITY_ERROR has been deprecated

See also

HSTR_LOG_LEVEL

7.16.3.7 const HSTR_SEVERITY HSTR_SEVERITY_FATAL_ERROR = DEPRECATED_HSTR_SEVERITY_FATAL_ERROR

Deprecated

HSTR_SEVERITY_FATAL_ERROR has been deprecated

See also

HSTR_LOG_LEVEL

7.16.3.8 const HSTR_SEVERITY HSTR_SEVERITY_INFO = DEPRECATED_HSTR_SEVERITY_INFO

Deprecated

HSTR_SEVERITY_INFO has been deprecated

See also

HSTR_LOG_LEVEL

7.16.3.9 const HSTR_SEVERITY HSTR_SEVERITY_WARNING = DEPRECATED_HSTR_SEVERITY_WARNING

Deprecated

HSTR_SEVERITY_WARNING has been deprecated

See also

HSTR_LOG_LEVEL

Chapter 8

Data Structure Documentation

8.1 HSTR_BUFFER_PROPS Struct Reference

`#include <hStreams_types.h>`

Data Fields

- HSTR_MEM_TYPE mem_type
  Memory type.
- HSTR_MEM_ALLOC_POLICY mem_alloc_policy
  Memory allocation policy.
- uint64_t flags
  Bitmask. Allowed values are described in HSTR_BUFFER_PROP_FLAGS.

8.1.1 Detailed Description

Additional properties when creating new buffer via Alloc1DEx. Default properties are:

- Prefer to use normal DRAM.
- Not enable aliasing.
- The instance associated with HSTR_SRC_LOG_DOMAIN is pinned.
- Buffer will not be instantiated for logical domains created after its allocation.

8.1.2 Field Documentation

8.1.2.1 uint64_t HSTR_BUFFER_PROPS::flags

Bitmask. Allowed values are described in HSTR_BUFFER_PROP_FLAGS.

8.1.2.2 HSTR_MEM_ALLOC_POLICY HSTR_BUFFER_PROPS::mem_alloc_policy

Memory allocation policy.

8.1.2.3 HSTR_MEM_TYPE HSTR_BUFFER_PROPS::mem_type

Memory type.

The documentation for this struct was generated from the following file:

- include/hStreams_types.h

8.2 HSTR_OPTIONS Struct Reference

`#include <hStreams_types.h>`

Public Member Functions

- HSTR_DEPRECATED ("HSTR_OPTIONS::_hStreams_EmitMessage has been deprecated. ""It has been replaced by a new logging mechanism.""Please refer to hStreams_Cfg_SetLogLevel() and hStreams_Cfg_SetLogInfoType().") int(*_hStreams_EmitMessage)(HSTR_SEVERITY
- HSTR_DEPRECATED ("HSTR_OPTIONS::verbose has been deprecated. ""Please refer to hStreams_Cfg_SetLogLevel() and hStreams_Cfg_SetLogInfoType().") uint32_t verbose

Data Fields

- const char const char hStreams_FatalError_Prototype_Fptr _hStreams_FatalError
- HSTR_KMP_AFFINITY kmp_affinity

  controls thread affinitization

- HSTR_DEP_POLICY dep_policy

  control deps only or data (default)

- uint32_t phys_domains_limit

  max # of phys domains

- HSTR_OPENMP_POLICY openmp_policy

  controls OpenMP startup

- int time_out_ms_val

  timeout for sync waits

- uint16_t libNameCnt
- int * libFlags
- char ** libNames
- uint16_t libNameCntHost
- char ** libNamesHost

8.2.1 Detailed Description

This type defines functions for the hStreams library to use for error handling, and other future options.

8.2.2 Member Function Documentation

8.2.2.1 HSTR_OPTIONS::HSTR_DEPRECATED ("HSTR_OPTIONS::verbose has been deprecated. ""Please refer to hStreams_Cfg_SetLogLevel() and hStreams_Cfg_SetLogInfoType().")

Deprecated

This option has been deprecated in favor of `hStreams_Cfg_SetLogLevel()` and `hStreams_Cfg_SetLogInfoType()`.

8.2.2.2 HSTR_OPTIONS::HSTR_DEPRECATED ("HSTR_OPTIONS::_hStreams_EmitMessage has been deprecated. ""It has been replaced by a new logging mechanism.""Please refer to hStreams_Cfg_SetLogLevel() and hStreams_Cfg_SetLogInfoType().")

internal_doc

Deprecated

_hStreams_EmitMessage has been deprecated in favor of a new logging mechanism. For details, consult the documentation of `hStreams_Cfg_SetLogLevel()` and `hStreams_Cfg_SetLogInfoType()`.

8.2.3 Field Documentation

8.2.3.1 const char const char hStreams_FatalError_Prototype_Fptr HSTR_OPTIONS::_hStreams_FatalError

8.2.3.2 HSTR_DEP_POLICY HSTR_OPTIONS::dep_policy

control deps only or data (default)

8.2.3.3 HSTR_KMP_AFFINITY HSTR_OPTIONS::kmp_affinity

controls thread affinitization

8.2.3.4 int* HSTR_OPTIONS::libFlags

8.2.3.5 uint16_t HSTR_OPTIONS::libNameCnt

8.2.3.6 uint16_t HSTR_OPTIONS::libNameCntHost

8.2.3.7 char** HSTR_OPTIONS::libNames

8.2.3.8 char** HSTR_OPTIONS::libNamesHost

8.2.3.9 HSTR_OPENMP_POLICY HSTR_OPTIONS::openmp_policy

controls OpenMP startup

8.2.3.10 uint32_t HSTR_OPTIONS::phys_domains_limit

max # of phys domains

8.2.3.11 int HSTR_OPTIONS::time_out_ms_val

timeout for sync waits

The documentation for this struct was generated from the following file:

- include/hStreams_types.h

Chapter 9

File Documentation

9.1 include/hStreams_app_api.h File Reference

Functions

- HSTR_RESULT hStreams_app_init_in_version (uint32_t in_StreamsPerDomain, uint32_t in_LogStreamOversubscription, const char *interface_version)

  Initialize hStreams homogenously across all available Intel(R) Xeon Phi(TM) coprocessors.

- static HSTR_RESULT hStreams_app_init (uint32_t in_StreamsPerDomain, uint32_t in_LogStreamOversubscription)
- HSTR_RESULT hStreams_app_init_domains_in_version (uint32_t in_NumLogDomains, uint32_t *in_pStreamsPerDomain, uint32_t in_LogStreamOversubscription, const char *interface_version)

  Initialize hStreams state, allowing for non-heterogeneity and more control then `hStreams_app_init()`.

- static HSTR_RESULT hStreams_app_init_domains (uint32_t in_NumLogDomains, uint32_t *in_pStreamsPerDomain, uint32_t in_LogStreamOversubscription)
- HSTR_RESULT hStreams_app_fini ()

  Finalization of hStreams state.

- HSTR_RESULT hStreams_app_create_buf (void *in_BufAddr, const uint64_t in_NumBytes)

  Allocate 1-dimensional buffer on each currently existing logical domains.

- HSTR_RESULT hStreams_app_xfer_memory (HSTR_LOG_STR in_LogStreamID, void *in_pWriteAddr, void *in_pReadAddr, uint64_t in_NumBytes, HSTR_XFER_DIRECTION in_XferDirection, HSTR_EVENT *out_pEvent)

  Enqueue 1-dimensional data transfers in a logical stream.

- HSTR_RESULT hStreams_app_invoke (HSTR_LOG_STR in_LogStreamID, const char *in_pFuncName, uint32_t in_NumScalarArgs, uint32_t in_NumHeapArgs, uint64_t *in_pArgs, HSTR_EVENT *out_pEvent, void *out_pReturnValue, uint16_t in_ReturnValueSize)

  Enqueue an execution of a user-defined function in a stream.

- HSTR_RESULT hStreams_app_stream_sync (HSTR_LOG_STR in_LogStreamID)

  Block until all the operation enqueued in a stream have completed.

- HSTR_RESULT hStreams_app_thread_sync ()

  Block until all the operation enqueued in all the streams have completed.

- HSTR_RESULT hStreams_app_event_wait (uint32_t in_NumEvents, HSTR_EVENT *in_pEvents)

  Wait on a set of events.

- HSTR_RESULT hStreams_app_event_wait_in_stream (HSTR_LOG_STR in_LogStreamID, uint32_t in_NumEvents, HSTR_EVENT *in_pEvents, int32_t in_NumAddresses, void **in_pAddresses, HSTR_EVENT *out_pEvent)

  Aggregate multiple dependences into one event handle and optionally insert that event handle into a logical stream.

- HSTR_RESULT hStreams_app_memset (HSTR_LOG_STR in_LogStreamID, void *in_pWriteAddr, int in_Value, uint64_t in_NumBytes, HSTR_EVENT *out_pEvent)

  Set remote memory to a value, using a named stream.

- HSTR_RESULT hStreams_app_memcpy (HSTR_LOG_STR in_LogStreamID, void *in_pWriteAddr, void *in_pReadAddr, uint64_t in_NumBytes, HSTR_EVENT *out_pEvent)

  copy remote memory, using a named stream

- HSTR_RESULT hStreams_app_sgemm (HSTR_LOG_STR in_LogStreamID, const CBLAS_ORDER Order, const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int64_t M, const int64_t N, const int64_t K, const float alpha, const float *A, const int64_t ldA, const float *B, const int64_t ldB, const float beta, float *C, const int64_t ldC, HSTR_EVENT *out_pEvent)

  perform a remote cblas sgemm

- HSTR_RESULT hStreams_app_dgemm (HSTR_LOG_STR in_LogStreamID, const CBLAS_ORDER Order, const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int64_t M, const int64_t N, const int64_t K, const double alpha, const double *A, const int64_t ldA, const double *B, const int64_t ldB, const double beta, double *C, const int64_t ldC, HSTR_EVENT *out_pEvent)

  perform a remote cblas dgemm

- HSTR_RESULT hStreams_app_cgemm (HSTR_LOG_STR in_LogStreamID, const CBLAS_ORDER Order, const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int64_t M, const int64_t N, const int64_t K, const void *alpha, const void *A, const int64_t ldA, const void *B, const int64_t ldB, const void *beta, void *C, const int64_t ldC, HSTR_EVENT *out_pEvent)

  perform a remote cblas cgemm

- HSTR_RESULT hStreams_app_zgemm (HSTR_LOG_STR in_LogStreamID, const CBLAS_ORDER Order, const CBLAS_TRANSPOSE TransA, const CBLAS_TRANSPOSE TransB, const int64_t M, const int64_t N, const int64_t K, const void *alpha, const void *A, const int64_t ldA, const void *B, const int64_t ldB, const void *beta, void *C, const int64_t ldC, HSTR_EVENT *out_pEvent)

perform a remote cblas zgemm

9.1.1 Detailed Description

9.1.2 Function Documentation

9.1.2.1 static HSTR_RESULT hStreams_app_init (uint32_t *in_StreamsPerDomain*, uint32_t *in_LogStreamOversubscription*) [static]

9.1.2.2 static HSTR_RESULT hStreams_app_init_domains (uint32_t *in_NumLogDomains*, uint32_t * *in_pStreamsPerDomain*, uint32_t *in_LogStreamOversubscription*) [static]

9.2 include/hStreams_app_api_sink.h File Reference

Functions

- HSTREAMS_EXPORT void hStreams_memcpy_sink (uint64_t byte_len, uint64_t *src, uint64_t *dest)

  Calls memcpy from string.h from (remote) sink side.

- HSTREAMS_EXPORT void hStreams_memset_sink (uint64_t byte_len, uint64_t char_value, uint64_t *buf)

  Calls memset from string.h from (remote) sink side.

- HSTREAMS_EXPORT void hStreams_sgemm_sink (uint64_t arg0, uint64_t arg1, uint64_t arg2, uint64_t arg3, uint64_t arg4, uint64_t arg5, uint64_t arg6, uint64_t arg7, uint64_t arg8, uint64_t arg9, uint64_t arg10, uint64_t arg11, uint64_t arg12, uint64_t arg13)

  Calls sgemm from (remote) sink side.

- HSTREAMS_EXPORT void hStreams_dgemm_sink (uint64_t arg0, uint64_t arg1, uint64_t arg2, uint64_t arg3, uint64_t arg4, uint64_t arg5, uint64_t arg6, uint64_t arg7, uint64_t arg8, uint64_t arg9, uint64_t arg10, uint64_t arg11, uint64_t arg12, uint64_t arg13)

  Calls dgemm from (remote) sink side.

- HSTREAMS_EXPORT void hStreams_cgemm_sink (uint64_t arg0, uint64_t arg1, uint64_t arg2, uint64_t arg3, uint64_t arg4, uint64_t arg5, uint64_t arg6, uint64_t arg7, uint64_t arg8, uint64_t arg9, uint64_t arg10, uint64_t arg11, uint64_t arg12, uint64_t arg13)

  Calls cgemm from (remote) sink side.

- HSTREAMS_EXPORT void hStreams_zgemm_sink (uint64_t arg0, uint64_t arg1, uint64_t arg2, uint64_t arg3, uint64_t arg4, uint64_t arg5, uint64_t arg6, uint64_t arg7, uint64_t arg8, uint64_t arg9, uint64_t arg10, uint64_t arg11, uint64_t arg12, uint64_t arg13, uint64_t arg14, uint64_t arg15)

  Calls zgemm from (remote) sink side.

9.2.1 Detailed Description

9.3 include/hStreams_common.h File Reference

Defines

- #define HSTR_WAIT_CONTROL 0
- #define HSTR_WAIT_NONE -1
- #define HSTR_RETURN_SIZE_LIMIT 64
- #define HSTR_ARGS_IMPLEMENTED 19
- #define HSTR_SRC_PHYS_DOMAIN -1
- #define HSTR_SRC_LOG_DOMAIN 0
- #define HSTR_TIME_INFINITE -1
- #define HSTR_MAX_FUNC_NAME_SIZE 80
- #define HSTR_MISC_DATA_SIZE 4096
- #define HSTR_ARGS_SUPPORTED (HSTR_MISC_DATA_SIZE-HSTR_MAX_FUNC_NAME_-SIZE)/sizeof(uint64_t)
- #define DllAccess

9.3.1 Define Documentation

9.3.1.1 #define DllAccess

9.3.1.2 #define HSTR_ARGS_IMPLEMENTED 19

9.3.1.3 #define HSTR_ARGS_SUPPORTED (HSTR_MISC_DATA_SIZE-HSTR_MAX_FUNC_-NAME_SIZE)/sizeof(uint64_t)

9.3.1.4 #define HSTR_MAX_FUNC_NAME_SIZE 80

9.3.1.5 #define HSTR_MISC_DATA_SIZE 4096

9.3.1.6 #define HSTR_RETURN_SIZE_LIMIT 64

9.3.1.7 #define HSTR_SRC_LOG_DOMAIN 0

9.3.1.8 #define HSTR_SRC_PHYS_DOMAIN -1

9.3.1.9 #define HSTR_TIME_INFINITE -1

9.3.1.10 #define HSTR_WAIT_CONTROL 0

9.3.1.11 #define HSTR_WAIT_NONE -1

9.4 include/hStreams_sink.h File Reference

9.5 include/hStreams_source.h File Reference

Defines

- #define CHECK_HSTR_RESULT(func)

Functions

- HSTR_RESULT hStreams_InitInVersion (const char *interface_version)

  Initialize hStreams-related state.

- static HSTR_RESULT hStreams_Init ()
- HSTR_RESULT hStreams_IsInitialized ()

  Check if hStreams has been initialised properly.

- HSTR_RESULT hStreams_Fini ()

  Finalize hStreams-related state.

- HSTR_RESULT hStreams_GetNumPhysDomains (uint32_t *out_pNumPhysDomains, uint32_t *out_pNumActivePhysDomains, bool *out_pHomogeneous)

  Returns number of discovered and active physical domains.

- HSTR_RESULT hStreams_GetPhysDomainDetails (HSTR_PHYS_DOM in_PhysDomain, uint32_t *out_pNumThreads, HSTR_ISA_TYPE *out_pISA, uint32_t *out_pCoreMaxMHz, HSTR_CPU_MASK out_MaxCPUmask, HSTR_CPU_MASK out_AvoidCPUmask, uint64_t *out_pSupportedMemTypes, uint64_t out_pPhysicalBytesPerMemType[HSTR_MEM_TYPE_-SIZE])

  Returns information about specified physical domain.

- HSTR_RESULT hStreams_GetAvailable (HSTR_PHYS_DOM in_PhysDomainID, HSTR_-CPU_MASK out_AvailableCPUmask)

  Returns unused yet cpu threads.

- HSTR_RESULT hStreams_AddLogDomain (HSTR_PHYS_DOM in_PhysDomainID, HSTR_-CPU_MASK in_CPUmask, HSTR_LOG_DOM *out_pLogDomainID, HSTR_OVERLAP_TYPE *out_pOverlap)

  Create a new logical domain in a physical domain.

- HSTR_RESULT hStreams_RmLogDomains (uint32_t in_NumLogDomains, HSTR_LOG_DOM *in_pLogDomainIDs)

  Remove logical domains.

- HSTR_RESULT hStreams_GetNumLogDomains (HSTR_PHYS_DOM in_PhysDomainID, uint32_t *out_pNumLogDomains)

  Return number logical domains associated with a physical domain.

- HSTR_RESULT hStreams_GetLogDomainIDList (HSTR_PHYS_DOM in_PhysDomainID, uint32_t in_NumLogDomains, HSTR_LOG_DOM *out_pLogDomainIDs)

Returns list of logical domains attached to provided physical domain.

- HSTR_RESULT hStreams_GetLogDomainDetails (HSTR_LOG_DOM in_LogDomainID, HSTR_PHYS_DOM *out_pPhysDomainID, HSTR_CPU_MASK out_CPUmask)

  Returns associated cpu mask and physical domain to provided logical domain.

- HSTR_RESULT hStreams_StreamCreate (HSTR_LOG_STR in_LogStreamID, HSTR_LOG_DOM in_LogDomainID, const HSTR_CPU_MASK in_CPUmask)

  Register a logical stream and specify its domain and CPU mask.

- HSTR_RESULT hStreams_StreamDestroy (HSTR_LOG_STR in_LogStreamID)

  Destroy a logical stream.

- HSTR_RESULT hStreams_GetNumLogStreams (HSTR_LOG_DOM in_LogDomainID, uint32_t *out_pNumLogStreams)

  Return number of logical streams associated with a logical domain.

- HSTR_RESULT hStreams_GetLogStreamIDList (HSTR_LOG_DOM in_LogDomainID, uint32_t in_NumLogStreams, HSTR_LOG_STR *out_pLogStreamIDs)

  Returns list of logical streams attached to provided logical domain.

- HSTR_RESULT hStreams_GetLogStreamDetails (HSTR_LOG_STR in_LogStreamID, HSTR_LOG_DOM in_LogDomainID, HSTR_CPU_MASK out_CPUmask)

  Returns cpu mask assigned to provided logical stream.

- HSTR_RESULT hStreams_GetOversubscriptionLevel (HSTR_PHYS_DOM in_PhysDomainID, uint32_t in_NumThreads, uint32_t *out_pOversubscriptionArray)

  Query the number of streams overlapping for each HW thread.

- HSTR_RESULT hStreams_EnqueueCompute (HSTR_LOG_STR in_LogStreamID, const char *in_pFunctionName, uint32_t in_numScalarArgs, uint32_t in_numHeapArgs, uint64_t *in_pArgs, HSTR_EVENT *out_pEvent, void *out_ReturnValue, uint16_t in_ReturnValueSize)

  Enqueue an execution of a user-defined function in a stream.

- HSTR_RESULT hStreams_EnqueueData1D (HSTR_LOG_STR in_LogStreamID, void *in_pWriteAddr, void *in_pReadAddr, uint64_t in_size, HSTR_XFER_DIRECTION in_XferDirection, HSTR_EVENT *out_pEvent)

  Enqueue 1-dimensional data transfers in a logical stream.

- HSTR_RESULT hStreams_EnqueueDataXDomain1D (HSTR_LOG_STR in_LogStreamID, void *in_pWriteAddr, void *in_pReadAddr, uint64_t in_size, HSTR_LOG_DOM in_destLogDomain, HSTR_LOG_DOM in_srcLogDomain, HSTR_EVENT *out_pEvent)

  Enqueue 1-dimensional data between an arbitrary domain and one of the endpoint domains of this stream.

- HSTR_RESULT hStreams_StreamSynchronize (HSTR_LOG_STR in_LogStreamID)

  Block until all the operation enqueued in a stream have completed.

- HSTR_RESULT hStreams_ThreadSynchronize ()

Block until all the operation enqueued in all the streams have completed.

- HSTR_RESULT hStreams_EventWait (uint32_t in_NumEvents, HSTR_EVENT *in_pEvents, bool in_WaitForAll, int32_t in_TimeOutMilliSeconds, uint32_t *out_pNumSignaled, uint32_t *out_pSignaledIndices)

  Wait on a set of events.

- HSTR_RESULT hStreams_EventStreamWait (HSTR_LOG_STR in_LogStreamID, uint32_t in_NumEvents, HSTR_EVENT *in_pEvents, int32_t in_NumAddresses, void **in_pAddresses, HSTR_EVENT *out_pEvent)

  Aggregate multiple dependences into one event handle and optionally insert that event handle into a logical stream.

- HSTR_RESULT hStreams_Alloc1D (void *in_BaseAddress, uint64_t in_size)

  Allocate 1-dimensional buffer on each currently existing logical domains.

- HSTR_RESULT hStreams_Alloc1DEx (void *in_BaseAddress, uint64_t in_Size, HSTR_BUFFER_PROPS *in_pBufferProps, int64_t in_NumLogDomains, HSTR_LOG_DOM *in_pLogDomainIDs)

  Allocate 1-dimensional buffer with additional properties.

- HSTR_RESULT hStreams_AddBufferLogDomains (void *in_Address, uint64_t in_NumLogDomains, HSTR_LOG_DOM *in_pLogDomainIDs)

  Create instances of the buffer in the logical domains specified as parameters.

- HSTR_RESULT hStreams_RmBufferLogDomains (void *in_Address, int64_t in_NumLogDomains, HSTR_LOG_DOM *in_pLogDomainIDs)

  Deallocate buffer instantiations in the selected logical domains.

- HSTR_RESULT hStreams_DeAlloc (void *in_Address)

  Destroy the buffer and remove all its instantiations.

- HSTR_RESULT hStreams_GetBufferNumLogDomains (void *in_Address, uint64_t *out_pNumLogDomains)

  Return the number of logical domains or which the buffer has been instantiated.

- HSTR_RESULT hStreams_GetBufferLogDomains (void *in_Address, uint64_t in_NumLogDomains, HSTR_LOG_DOM *out_pLogDomains, uint64_t *out_pNumLogDomains)

  Return a list of logical domains for which the buffer is instantiated.

- HSTR_RESULT hStreams_GetBufferProps (void *in_Address, HSTR_BUFFER_PROPS *out_BufferProps)

  Returns buffer properties associated with a buffer.

- HSTR_RESULT hStreams_GetLastError ()

  Get the last error.

- void hStreams_ClearLastError ()

  Clear the last hStreams error across.

- HSTR_RESULT hStreams_Cfg_SetLogLevel (HSTR_LOG_LEVEL in_loglevel)
  Set a logging level for the hetero-streams library.
- HSTR_RESULT hStreams_Cfg_SetLogInfoType (uint64_t in_info_type_mask)
  Set a bitmask of message categories that the library should emit.
- HSTR_RESULT hStreams_Cfg_SetMKLInterface (HSTR_MKL_INTERFACE in_MKLInterface)
  Choose used MKL interface version.
- HSTR_RESULT hStreams_SetOptions (const HSTR_OPTIONS *in_options)
  Configure user parameters by setting hStreams Options.
- HSTR_RESULT hStreams_GetCurrentOptions (HSTR_OPTIONS *pCurrentOptions, uint64_t buffSize)
  Query user parameters by getting hStreams Options.
- HSTR_RESULT hStreams_GetVersionStringLen (uint32_t *out_pVersionStringLen)
  Report the length of the version string, including the null termination character.
- HSTR_RESULT hStreams_Version (char *buff, uint32_t buffLength)
  Report hStreams version info to buffer.
- HSTR_DEPRECATED ("hStreams_GetVerbose() has been deprecated. ""Please refer to hStreams_Cfg_SetLogLevel() and hStreams_Cfg_SetLogInfoType().") uint32_t hStreams_GetVerbose()
- HSTR_DEPRECATED ("hStreams_SetVerbose() has been deprecated. ""Please refer to hStreams_Cfg_SetLogLevel() and hStreams_Cfg_SetLogInfoType().") HSTR_RESULT hStreams_SetVerbose(int target_verbosity)
- const char * hStreams_ResultGetName (HSTR_RESULT in_olr)
  Get HSTR_RESULT name.
- static uint64_t HSTR_CPU_MASK_ISSET (int bitNumber, const HSTR_CPU_MASK cpu_mask)
  Roughly equivalent to CPU_ISSET().
- static void HSTR_CPU_MASK_SET (int bitNumber, HSTR_CPU_MASK cpu_mask)
  Roughly equivalent to CPU_SET().
- static void HSTR_CPU_MASK_ZERO (HSTR_CPU_MASK cpu_mask)
  Roughly equivalent to CPU_ZERO().
- static void HSTR_CPU_MASK_AND (HSTR_CPU_MASK dst, const HSTR_CPU_MASK src1, const HSTR_CPU_MASK src2)
  Roughly equivalent to CPU_AND().
- static void HSTR_CPU_MASK_XOR (HSTR_CPU_MASK dst, const HSTR_CPU_MASK src1, const HSTR_CPU_MASK src2)

Roughly equivalent to CPU_XOR().

- static void HSTR_CPU_MASK_OR (HSTR_CPU_MASK dst, const HSTR_CPU_MASK src1, const HSTR_CPU_MASK src2)

  Roughly equivalent to CPU_OR().

- static int HSTR_CPU_MASK_COUNT (const HSTR_CPU_MASK cpu_mask)

  Roughly equivalent to CPU_COUNT().

- static int HSTR_CPU_MASK_EQUAL (const HSTR_CPU_MASK cpu_mask1, const HSTR_CPU_MASK cpu_mask2)

  Roughly equivalent to CPU_EQUAL().

- static void HSTR_CPU_MASK_XLATE (HSTR_CPU_MASK dest, const cpu_set_t *src)

  Utility function to translate from cpu_set to COI_CPU_MASK.*

- static void HSTR_CPU_MASK_XLATE_EX (cpu_set_t *dest, const HSTR_CPU_MASK src)

  Utility function to translate from COI_CPU_MASK to cpu_set.*

9.5.1 Detailed Description

9.5.2 Define Documentation

9.5.2.1 #define CHECK_HSTR_RESULT(func)

Value:

```
{                                                                       \
        HSTR_RESULT hret = HSTR_RESULT_SUCCESS;                         \
        hret = func;                                                    \
        if (hret != HSTR_RESULT_SUCCESS) {                              \
            printf("%s returned %s.\n", #func, hStreams_ResultGetName(hret)); \
            return hret;                                                \
        }                                                               \
    }
```

9.5.3 Function Documentation

9.5.3.1 HSTR_DEPRECATED ("hStreams_SetVerbose() has been deprecated. ""Please refer to hStreams_Cfg_SetLogLevel() and hStreams_Cfg_SetLogInfoType().")

Deprecated

This function has been deprecated in favor of hStreams_Cfg_SetLogLevel() and hStreams_Cfg_SetLogInfoType().

9.5.3.2 HSTR_DEPRECATED ("hStreams_GetVerbose() has been deprecated. ""Please refer to hStreams_Cfg_SetLogLevel() and hStreams_Cfg_SetLogInfoType().")

Deprecated

This function has been deprecated in favor of hStreams_Cfg_SetLogLevel() and hStreams_Cfg_-SetLogInfoType().

9.5.3.3 static HSTR_RESULT hStreams_Init () [static]

9.6 include/hStreams_types.h File Reference

Data Structures

- struct HSTR_OPTIONS
- struct HSTR_BUFFER_PROPS

Defines

- #define HSTR_BUFFER_PROPS_INITIAL_VALUES
- #define HSTR_BUFFER_PROPS_INITIAL_VALUES_EX

Typedefs

- typedef int HSTR_OVERLAP_TYPE

  For managing overlap of cpu masks of partitions.

- typedef int HSTR_RESULT

  Type that is returned from hStream functions.

- typedef int64_t HSTR_INFO_TYPE

  Underlying type large enough to encompass any of HSTR_INFO_TYPE_VALUES.

- typedef enum HSTR_SEVERITY HSTR_SEVERITY
- typedef int32_t HSTR_LOG_LEVEL

  Underlying type large enough to encompass any of HSTR_LOG_LEVEL_VALUES.

- typedef int HSTR_DEP_POLICY

  This is the type associated with hStream dependence policies.

- typedef int HSTR_KMP_AFFINITY

  Type associated with hStream's KMP affinity policy.

- typedef int HSTR_OPENMP_POLICY

  Type associated with hStream OpenMP handling FIXME: This will be changing in the transition to support other threading runtimes.

- typedef int HSTR_MEM_TYPE

  Type associated with hStream physical memory types These are consecutive integers, NOT mask values.

- typedef int HSTR_XFER_DIRECTION

  Type associated with direction of data transfer.

- typedef int HSTR_ISA_TYPE

  This type encapsulates the COI_ISA_TYPE, to enable building hStreams on something other than COI in the future. It is used to indicate the ISA for enumerated domains.

- typedef int HSTR_BUFFER_PROP_FLAGS

  Type associated with mask of flags for buffers.

- typedef int HSTR_MEM_ALLOC_POLICY

  Type associated with hStream memory allocation policy regarding the behaviour when either: a. the requested memory type has been exhausted on some node b. the requested memory type does not even exist on some node.

- typedef int HSTR_MKL_INTERFACE
- typedef uint64_t HSTR_LOG_STR
- typedef uint32_t HSTR_LOG_DOM
- typedef int32_t HSTR_PHYS_DOM
- typedef void(* hStreams_FatalError_Prototype_Fptr)(int)
- typedef struct HSTR_OPTIONS HSTR_OPTIONS
- typedef struct HSTR_BUFFER_PROPS HSTR_BUFFER_PROPS

Enumerations

- enum HSTR_OVERLAP_TYPE_VALUES { HSTR_NO_OVERLAP = 0, HSTR_EXACT_OVERLAP, HSTR_PARTIAL_OVERLAP }

  Possible values of `HSTR_OVERLAP_TYPE`.

- enum HSTR_RESULT_VALUES {

  HSTR_RESULT_SUCCESS = 0, HSTR_RESULT_REMOTE_ERROR, HSTR_RESULT_NOT_INITIALIZED, HSTR_RESULT_NOT_FOUND,

  HSTR_RESULT_ALREADY_FOUND, HSTR_RESULT_OUT_OF_RANGE, HSTR_RESULT_DOMAIN_OUT_OF_RANGE, HSTR_RESULT_CPU_MASK_OUT_OF_RANGE,

  HSTR_RESULT_OUT_OF_MEMORY, HSTR_RESULT_INVALID_STREAM_TYPE, HSTR_RESULT_OVERLAPPING_RESOURCES, HSTR_RESULT_DEVICE_NOT_INITIALIZED,

  HSTR_RESULT_BAD_NAME, HSTR_RESULT_TOO_MANY_ARGS, HSTR_RESULT_TIME_OUT_REACHED, HSTR_RESULT_EVENT_CANCELED,

  HSTR_RESULT_INCONSISTENT_ARGS, HSTR_RESULT_BUFF_TOO_SMALL, HSTR_RESULT_MEMORY_OPERAND_INCONSISTENT, HSTR_RESULT_NULL_PTR,

  HSTR_RESULT_INTERNAL_ERROR, HSTR_RESULT_RESOURCE_EXHAUSTED, HSTR_RESULT_NOT_IMPLEMENTED, HSTR_RESULT_NOT_PERMITTED,

  HSTR_RESULT_SIZE }

  Possible values of `HSTR_RESULT`.

- enum HSTR_INFO_TYPE_VALUES {

  HSTR_INFO_TYPE_TRACE = (1ULL << 5), HSTR_INFO_TYPE_SINK_INVOKE = (1ULL << 6), HSTR_INFO_TYPE_MEM = (1ULL << 7), HSTR_INFO_TYPE_SYNC = (1ULL << 8),

  HSTR_INFO_TYPE_MISC = (1ULL << 9) }

  Message type categories.

- enum HSTR_SEVERITY

- enum HSTR_LOG_LEVEL_VALUES {
  HSTR_LOG_LEVEL_NO_LOGGING, HSTR_LOG_LEVEL_FATAL_ERROR, HSTR_LOG_-LEVEL_ERROR, HSTR_LOG_LEVEL_WARN,
  HSTR_LOG_LEVEL_LOG, HSTR_LOG_LEVEL_DEBUG1, HSTR_LOG_LEVEL_DEBUG2, HSTR_LOG_LEVEL_DEBUG3,
  HSTR_LOG_LEVEL_DEBUG4 }
  Message type categories.

- enum HSTR_DEP_POLICY_VALUES { HSTR_DEP_POLICY_CONSERVATIVE = 0, HSTR_-DEP_POLICY_BUFFERS, HSTR_DEP_POLICY_NONE, HSTR_DEP_POLICY_SIZE }
  Possible values of `HSTR_DEP_POLICY`.

- enum HSTR_KMP_AFFINITY_VALUES { HSTR_KMP_AFFINITY_BALANCED = 0, HSTR_-KMP_AFFINITY_COMPACT, HSTR_KMP_AFFINITY_SCATTER, HSTR_KMP_AFFINITY_-SIZE }
  Possible values of `HSTR_KMP_AFFINITY`.

- enum HSTR_OPENMP_POLICY_VALUES { HSTR_OPENMP_ON_DEMAND = 0, HSTR_-OPENMP_PRE_SETUP = 1, HSTR_OPENMP_POLICY_SIZE }
  Possible values of `HSTR_KMP_AFFINITY`.

- enum HSTR_MEM_TYPE_VALUES { HSTR_MEM_TYPE_ANY = -1, HSTR_MEM_TYPE_-NORMAL = 0, HSTR_MEM_TYPE_HBW, HSTR_MEM_TYPE_SIZE }
  Possible values of `HSTR_MEM_TYPE`.

- enum HSTR_XFER_DIRECTION_VALUES { HSTR_SINK_TO_SRC = 0, HSTR_SRC_TO_-SINK = 1 }
  Possible values of `HSTR_XFER_DIRECTION`.

- enum HSTR_ISA_TYPE_VALUES {
  HSTR_ISA_INVALID = 0, HSTR_ISA_x86_64, HSTR_ISA_MIC, HSTR_ISA_KNF,
  HSTR_ISA_KNC, HSTR_ISA_KNL }
  Possible values of `HSTR_ISA_TYPE`.

- enum HSTR_BUFFER_PROP_FLAGS_VALUES {
  HSTR_BUF_PROP_ALIASED = 1, HSTR_BUF_PROP_SRC_PINNED = 2, HSTR_BUF_-PROP_INCREMENTAL = 4, HSTR_BUF_PROP_AFFINITIZED = 8,
  HSTR_BUF_PROP_INVALID_VALUE = 16 }
  Possible values of `HSTR_BUFFER_PROP_FLAGS`.

- enum HSTR_MEM_ALLOC_POLICY_VALUES { HSTR_MEM_ALLOC_PREFERRED = 0, HSTR_MEM_ALLOC_STRICT, HSTR_MEM_ALLOC_POLICY_SIZE }
  Possible values of `HSTR_MEM_ALLOC_POLICY`.

- enum HSTR_MKL_INTERFACE_VALUES { HSTR_MKL_LP64 = 0, HSTR_MKL_ILP64, HSTR_MKL_NONE, HSTR_MKL_INTERFACE_SIZE }
  Possible values of `HSTR_MKL_INTERFACE`.

Variables

- const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_ALWAYSEMIT = DEPRECATED_HSTR_INFO_TYPE_ALWAYSEMIT
- const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_INVOKE = DEPRECATED_HSTR_INFO_TYPE_INVOKE
- const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_DEPS = DEPRECATED_HSTR_INFO_TYPE_DEPS
- const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_GENERAL = DEPRECATED_HSTR_INFO_TYPE_GENERAL
- const HSTR_INFO_TYPE_VALUES HSTR_INFO_TYPE_AND16 = DEPRECATED_HSTR_INFO_TYPE_AND16
- const HSTR_SEVERITY HSTR_SEVERITY_INFO = DEPRECATED_HSTR_SEVERITY_INFO
- const HSTR_SEVERITY HSTR_SEVERITY_WARNING = DEPRECATED_HSTR_SEVERITY_WARNING
- const HSTR_SEVERITY HSTR_SEVERITY_ERROR = DEPRECATED_HSTR_SEVERITY_ERROR
- const HSTR_SEVERITY HSTR_SEVERITY_FATAL_ERROR = DEPRECATED_HSTR_SEVERITY_FATAL_ERROR

9.6.1 Detailed Description

9.6.2 Define Documentation

9.6.2.1 #define HSTR_BUFFER_PROPS_INITIAL_VALUES

Value:

```
{         \
        HSTR_MEM_TYPE_NORMAL,                   \
        HSTR_MEM_ALLOC_PREFERRED,               \
        HSTR_BUF_PROP_SRC_PINNED}
```

9.6.2.2 #define HSTR_BUFFER_PROPS_INITIAL_VALUES_EX

Value:

```
{     \
        HSTR_MEM_TYPE_NORMAL,                   \
        HSTR_MEM_ALLOC_PREFERRED,               \
        0}
```

9.7 include/hStreams_version.h File Reference

Defines

- #define HSTR_VERSION_MAJOR 1
- #define HSTR_VERSION_MINOR 0
- #define HSTR_VERSION_MICRO 0
- #define HSTR_VERSION_STRING "1.0"

9.7.1 Define Documentation

9.7.1.1 #define HSTR_VERSION_MAJOR 1

9.7.1.2 #define HSTR_VERSION_MICRO 0

9.7.1.3 #define HSTR_VERSION_MINOR 0

9.7.1.4 #define HSTR_VERSION_STRING "1.0"

Index