

### Introduction

This guide assists users in porting their code from version 2 to version 3 of the ClearSpeed host programming interface (CSAPI). The general use of the CSAPI library has not changed, but a number of functions have had their parameters changed. These changes reflect the addition of new features and work to streamline the interface.

The first section provides an overview of the changes. The second section provides a step by step guide to modifying CSAPI function calls. The third section provides supplementary information.

### New Features

- Process handles are now fully incorporated into the CSAPI.
- Semaphores are now dynamically allocated using semaphore handles.
- The Global Semaphore Unit (GSU) semaphores are now accessible using the same semaphore functions as the TSC semaphores.
- The GSU semaphores can be linked with data transfers to reduce latency.
- New functions allow access to 'Get' and 'Put' the semaphore value.
- Functions that block the calling thread now include a timeout parameter.
- A list of sections to be loaded to ESRAM can now be specified at load time.
- Runtime arguments can now be passed to the **C<sup>n</sup>** program using the `argc` and `argv` parameters of the main function.
- The memory transfer functions now provide control over the following:
  - Whether the board side address is checked and what action is taken.
  - Whether the alignment of the buffers is checked and what action is taken.
  - Whether the processor should be halted during the transfer.
  - Whether the data cache on the processor should be flushed.
- New functions provide information on the following:
  - The current connection and the current process.
  - Processor temperatures and clock speeds.
  - PCI link status and error conditions.

## Table of contents

<b>1</b>	<b>Overview of changes</b>	<b>5</b>
1.1	Process handles	5
1.2	Semaphore handles	5
1.3	GSU semaphore interface	5
1.4	Timeout parameters	6
1.5	CSAPI_connect	6
1.6	CSAPI_load	6
1.7	CSAPI_run	6
1.8	CSAPI_read_mono_memory and CSAPI_write_mono_memory	6
1.9	Type definitions	7
1.10	Removed unnecessary functions	7
1.11	Removed unnecessary parameters	7
1.12	New functions	7
<b>2</b>	<b>Step by step guide to migration</b>	<b>8</b>
2.1	Replace statically linked stub library	8
2.2	Replace CSAPI header file path	8
2.3	Remove error codes	8
2.4	Remove calls to old functions	9
2.5	Replace DRV with CSAPI	9
2.6	Replace call back function declarations	9
2.7	Remove use of old declarations	9
2.8	Replace enumerated type definitions	10
2.9	CSAPI_num_cards	10
2.10	CSAPI_new	10
2.11	CSAPI_connect	10
2.12	CSAPI_reset	11
2.13	CSAPI_load	11
2.14	CSAPI_get_last_loaded_handle	12
2.15	CSAPI_run	12
2.16	CSAPI_run_process	12

---

2.17	CSAPI_wait_on_terminate	12
2.18	CSAPI_get_return_value	12
2.19	CSAPI_unload	12
2.20	CSAPI_get_symbol_value	13
2.21	CSAPI_get_symbol_value_loaded	13
2.22	CSAPI_get_free_mem	13
2.23	CSAPI_allocate_shared_memory	13
2.24	CSAPI_allocate_static_shared_memory	13
2.25	CSAPI_free	14
2.26	CSAPI_write_mono_memory	14
2.27	CSAPI_read_mono_memory	14
2.28	CSAPI_write_mono_memory_raw	14
2.29	CSAPI_read_mono_memory_raw	15
2.30	CSAPI_write_mono_memory_async	15
2.31	CSAPI_read_mono_memory_async	15
2.32	CSAPI_write_mono_memory_async_wait	16
2.33	CSAPI_read_mono_memory_async_wait	16
2.34	CSAPI_write_mono_memory_async_poll	16
2.35	CSAPI_read_mono_memory_async_poll	16
2.36	CSAPI_num_semaphores	17
2.37	CSAPI_semaphore_signal	17
2.38	CSAPI_semaphore_wait	17
2.39	CSAPI_get_callback	18
2.40	CSAPI_register_callback	18
2.41	CSAPI_endianness	18
<b>3</b>	<b>Supplementary information</b>	<b>19</b>
3.1	Connection	19
3.2	Process handles	19
3.3	Runtime arguments	19
3.4	Memory allocation functions	20
3.5	Memory transfer functions	20
3.6	Asynchronous memory transfer functions	21

---

3.7	Semaphores .....	21
3.8	Memory and Register parameter types .....	22
3.9	Header files .....	22
3.9.1	Header files for enumerated types .....	22
3.9.2	Reorganization of header files .....	22
3.10	Version identifiers .....	22
	<b>Revision history .....</b>	<b>24</b>

# 1 Overview of changes

## 1.1 Header files

### 1.1.1 Header files for enumerated types

The enumerated types have been moved to their own header file called `csapi_types.h`. This new file is included by the `csapi.h` header file, along with `csapi_errno.h`. You can replace the inclusion of `csapi.h` by the inclusion of `csapi_types.h` in source files or header files that require only the type declarations.

### 1.1.2 Reorganization of header files

The directory structure for C<sup>n</sup> and assembler code header files has been rationalized to separate those used on the host and the card. If you just use the setup scripts and shortcuts installed with the software, you should not need to do anything: the search paths will be updated and your code should just compile as before.

If you use your own initialization environment setup via scripts or within an IDE for example, you will need to update the paths.

New `card` and `host` subdirectories have been created under the `include` directory.

The `card` directory has a `cn` subdirectory for all the standard C<sup>n</sup> header files and an `asm` subdirectory for all the assembler include files (including those for the card-side FFT library).

The `host` directory contains the header files needed by host-side code, for example, the CSAPI header files (`csaspi.h` and the others) and the host-side FFT headers.

## 1.2 Process handles

The process handles used with dynamically linked `csx` files in version 2 of the CSAPI are now fully incorporated into the interface. `CSAPI_load` will provide a process handle in a thread safe way, and the process handle is now required by the following functions, which use it instead of a processor index or file name:

- `CSAPI_run`
- `CSAPI_wait_on_terminate`
- `CSAPI_get_symbol_value`
- `CSAPI_allocate_shared_memory`
- `CSAPI_allocate_static_shared_memory`
- `CSAPI_allocate_shared_semaphore`
- `CSAPI_allocate_static_shared_semaphore`

The process handle is used in the same way for statically linked `csx` files. This allows memory to be dynamically allocated after a statically linked `csx` file has been loaded. It is also possible to load statically linked and dynamically linked `csx` files at the same time, as long as the memory required by the statically linked `csx` program is available.

## 1.3 Semaphore handles

The semaphore functions now take a semaphore handle instead of a processor index and semaphore number. The semaphore handle is provided by the new semaphore allocation functions, which are called at runtime. This allows independent pieces of host side and board side **C<sup>n</sup>** code to be used at the same time, without the need to reserve statically allocated semaphores. It also prevents the user from operating on semaphores used by the SDK library.

The semaphore allocation functions are similar to the memory allocation functions, and include a static allocation function for backwards compatibility with csx programs that use a fixed semaphore.

## 1.4 GSU semaphore interface

The modified semaphore functions can be used on GSU semaphores in exactly the same way as with TSC semaphores. The only difference is the type of semaphore requested during allocation. This provides host side access to the GSU semaphores, which can be waited on by multiple processors and can be signalled with data.

## 1.5 Timeout parameters

The following CSAPI functions now take a timeout parameter (in micro-seconds), which can be used to prevent the functions from blocking indefinitely:

- `CSAPI_reset`
- `CSAPI_load`.
- `CSAPI_wait_on_terminate`
- `CSAPI_read_mono_memory_wait`
- `CSAPI_write_mono_memory_wait`

There is a `CSAPI_NO_TIMEOUT` definition, which can be used to block indefinitely.

## 1.6 CSAPI\_connect

The `CSAPI_connect` function now takes a timeout parameter. This allows the connection to block until the board is available. The function also has parameters to set the connection type, rather than relying on the special case where the host name parameter is `NULL` for a direct connection to hardware.

## 1.7 CSAPI\_load

The `CSAPI_load` function now takes a list of section names. This allows the user to specify sections to be loaded to the ESRAM memory. The keywords "ALL" and "ANY" can also be used to load all sections or as many sections as possible to ESRAM.

## 1.8 CSAPI\_run

The `CSAPI_run` function now takes an argument string for the `csx` program. The arguments passed to `CSAPI_run` will be available using the `argc` and `argv` parameters of the main function in the `Cn` program.

## 1.9 CSAPI\_read\_mono\_memory and CSAPI\_write\_mono\_memory

The `CSAPI_read_mono_memory` and `CSAPI_write_mono_memory` functions now take a structure of transfer parameters. These parameters allow control over the following:

- Whether validity of the card side address is checked and what action is taken.
- Whether the necessary alignment of the buffers is checked and what action is taken.
- Whether the processor should be halted during the transfer.
- Whether the data cache on the processor should be flushed.

The parameters also allow GSU semaphores to be linked to the DMA transfer. One semaphore can be signalled to start the transfer and another semaphore can be signalled when the transfer completes. The GSU semaphores can be signalled and waited on by the host or any CSX600 processor. This can be used to eliminate the delay between executing code on the CSX600 processor and DMA transfers.

## 1.10 Type definitions

Parameters for card side memory addresses, register addresses and register values are now defined as a unique CSAPI type. This provides forward compatibility if the size of the parameters is changed.

## 1.11 Removed unnecessary functions

The `CSAPI_register_application` and `CSAPI_register_semaphore` functions have been removed, and the host can wait on semaphores at the same time as the processor.

The `CSAPI_get_last_loaded_handle`, `CSAPI_run_process` and `CSAPI_get_symbol_value_loaded` functions have been removed now that the `CSAPI_load`, `CSAPI_run` and `CSAPI_get_symbol_value` functions use process handles.

The `CSAPI_read_mono_memory_raw` and `CSAPI_write_mono_memory_raw` functions have been removed now that the `CSAPI_read_mono_memory` and `CSAPI_write_mono_memory` functions take transfer parameters that provide full control over halting the processor and flushing the cache.

The `CSAPI_read_mono_memory_async_poll` and `CSAPI_write_mono_memory_async_poll` functions have been removed now that the `CSAPI_read_mono_memory_async_wait` and `CSAPI_write_mono_memory_async_wait` functions take a timeout parameter that can have a value of 0.

## 1.12 Removed unnecessary parameters

The `CSAPI_new` function no longer takes a `mode` parameter.

The `CSAPI_num_cards` function no longer takes a `CSAPIState` parameter.

Functions which take processor handles or semaphore handles no longer require a processor index, since the handle is associated with a processor when it is created.

The `CSAPI_read_mono_memory` and `CSAPI_write_mono_memory` functions no longer required a processor index. It is now determined from the card side address.

## 1.13 New functions

The `CSAPI_semaphore_get` and `CSAPI_semaphore_put` functions have been added to expand the interface for accessing both TSC and GSU semaphores.

The `CSAPI_get_connection_info` and `CSAPI_get_process_info` functions have been added to provide access to the details of a connection or process.

The `CSAPI_status` function has been added to provide information about the current state of the hardware, including temperatures and error conditions.

## 2 Step by step guide to migration

### 2.1 Replace statically linked stub library

You must replace the static link to `libclear_stub_lib`.

On Windows:

<b>Replace</b>	<code>libclear_stub_lib.lib</code>
<b>With</b>	<code>csapi.lib</code>

On Linux:

<b>Replace</b>	<code>libclear_stub_lib.a</code>
<b>With</b>	<code>libcsapi.a</code>

### 2.2 Replace CSAPI header file path

If you have specified the path to the `csapi.h` header file explicitly, you must update this as follows:

<b>Replace</b>	<code>&lt;package installation directory&gt;/include/cs_api/csapi.h</code>
<b>With</b>	<code>&lt;package installation directory&gt;/include/host/csapi.h</code>

The `CSHOSTINC` environment variable has already been updated with this change.

### 2.3 Remove error codes

You must remove any reference to the following error codes because they are no longer returned by any CSAPI function and their declaration has been removed:

- `DRVerrno_connection_broken`
- `DRVerrno_cannot_connect`
- `DRVerrno_bad_inet_address`
- `DRVerrno_socket`
- `DRVerrno_kernel_max_app_count_exceeded`
- `DRVerrno_kernel_que`
- `DRVerrno_failed_to_load_function`
- `DRVerrno_cannot_allocate_event`
- `DRVerrno_semaphore_number_out_of_bounds`

## 2.4 Remove calls to old functions

You must remove calls to the following functions, which have been removed:

- CSAPI\_register\_application
- CSAPI\_register\_semaphore

## 2.5 Replace DRV with CSAPI

You must replace all instances of DRVErrno with CSAPIErrno, even where they form part of a word.

<b>Replace</b>	DRVErrno
<b>With</b>	CSAPIErrno
<b>Replace</b>	DRVEvent
<b>With</b>	CSAPIEvent
<b>Replace</b>	DRVProcess
<b>With</b>	CSAPIProcess

## 2.6 Replace call back function declarations

You must replace all instances of CSAPI\_EventFnPtr with CSAPIEventFnPtr.

<b>Replace</b>	CSAPI_EventFnPtr
<b>With</b>	CSAPIEventFnPtr

## 2.7 Remove use of old declarations

You must replace the following definitions, which have been removed:

- Replace CS\_PROCESSOR0 with the integer value 0.
- Replace CS\_PROCESSOR1 with the integer value 1.

## 2.8 Replace enumerated type definitions

You must replace all instances of the following types, which have changed name.

<b>Replace</b>	CSV_Fpga
<b>With</b>	CSV_FpgaVersion
<b>Replace</b>	CSV_Mtap
<b>With</b>	CSV_MtapVersion
<b>Replace</b>	CSP_NoZeroBss
<b>With</b>	CSP_ZeroBss and invert the associated value
<b>Replace</b>	CSP_CCBRClockSlow
<b>With</b>	CSP_CcbrClockSlow
<b>Replace</b>	CSP_CCBRClockFast
<b>With</b>	CSP_CcbrClockFast
<b>Replace</b>	CSP_DDRClock
<b>With</b>	CSP_DdrClock

You must not call CSAPI\_set\_system\_param with the following parameter types, because these types have been removed:

- CSP\_ResetFlags
- CSP\_NoFPGAModel

## 2.9 CSAPI\_num\_cards

You must remove the CSAPIState parameter from all calls to CSAPI\_num\_cards.

<b>Replace</b>	CSAPI_num_cards(s, &num_of_cards);
<b>With</b>	CSAPI_num_cards(&num_of_cards);

## 2.10 CSAPI\_new

You must remove the mode parameter from all calls to CSAPI\_new.

<b>Replace</b>	CSAPI_new(CM_Direct);
<b>With</b>	CSAPI_new();

## 2.11 CSAPI\_connect

You must use a value of CSH\_Private for the new sharing\_type parameter and a value of CSC\_Direct or CSC\_Socket for the new connection\_type parameter, see

*Section 3.1: Connection.* You must also use a timeout value of 0 to continue the previous behavior. When connecting directly to hardware make the following replacement.

<b>Replace</b>	<code>CSAPI_connect(s, NULL, instance);</code>
<b>With</b>	<code>CSAPI_connect(s, CSH_Private, CSC_Direct, "localhost", instance, 0);</code>

When connecting to a simulator running locally make the following replacement.

<b>Replace</b>	<code>CSAPI_connect(s, "localhost", instance);</code>
<b>With</b>	<code>CSAPI_connect(s, CSH_Private, CSC_Socket, "localhost", instance, 0);</code>

## 2.12 CSAPI\_reset

You must replace `CSAPIFlags_FULL_SYSTEM_RESET` with `CSR_FullSystemReset` and use a timeout value of `CSAPI_NO_TIMEOUT` to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_reset(s, proc_inx, CSAPIFlags_FULL_SYSTEM_RESET);</code>
<b>With</b>	<code>CSAPI_reset(s, proc_inx, CSR_FullSystemReset, CSAPI_NO_TIMEOUT);</code>

## 2.13 CSAPI\_load

If you call the `CSAPI_get_last_loaded_handle` function after `CSAPI_load` then you must replace these two calls with a single call to the `CSAPI_load` function.

<b>Replace</b>	<code>CSAPI_load(s, proc_inx, "file_name.csx");</code> <code>CSAPI_get_last_loaded_handle(s, proc_inx, &amp;process);</code>
<b>With</b>	<code>CSAPI_load(s, proc_inx, "file_name.csx", NULL, &amp;process, CSAPI_NO_TIMEOUT);</code>

If you did not call the `CSAPI_get_last_loaded_handle` function then you must declare a process handle of type `struct CSAPIProcess*` and pass the address of this as the process parameter of the `CSAPI_load` function.

<b>Replace</b>	<code>CSAPI_load(s, proc_inx, "file_name.csx");</code>
<b>With</b>	<code>struct CSAPIProcess* process;</code> <code>CSAPI_load(s, proc_inx, "file_name.csx", NULL, &amp;process, CSAPI_NO_TIMEOUT);</code>

If multiple boards or processors are being used then a process handle will be required for each processor on each board:

Use `CSAPI_load(s[board], proc_inx, "file_name.csx", NULL, &process[board][proc_inx], CSAPI_NO_TIMEOUT)`.

This array of processes should then be used instead of a single process handle for the following changes.

You must call `CSAPI_unload` at the end of your program to unload the process.

### 2.14 CSAPI\_get\_last\_loaded\_handle

You must remove calls to the `CSAPI_get_last_loaded_handle` function, which has been removed. (See [2.13: CSAPI\\_load](#)).

### 2.15 CSAPI\_run

You must replace the processor index with a process handle and use a value of `NULL` for the `csx_args` parameter to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_run(s, proc_inx);</code>
<b>With</b>	<code>CSAPI_run(s, process, NULL);</code>

### 2.16 CSAPI\_run\_process

You must replace each call to `CSAPI_run_process` with a call to `CSAPI_run`.

<b>Replace</b>	<code>CSAPI_run_process(s, proc_inx, process);</code>
<b>With</b>	<code>CSAPI_run(s, process, NULL);</code>

### 2.17 CSAPI\_wait\_on\_terminate

You must replace the processor index with a process handle and use a timeout value of `CSAPI_NO_TIMEOUT` to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_wait_on_terminate(s, proc_inx);</code>
<b>With</b>	<code>CSAPI_wait_on_terminate(s, process, CSAPI_NO_TIMEOUT);</code>

### 2.18 CSAPI\_get\_return\_value

You must replace the processor index with a process handle.

<b>Replace</b>	<code>CSAPI_get_return_value(s, proc_inx, &amp;return_value);</code>
<b>With</b>	<code>CSAPI_get_return_value(s, process, &amp;return_value);</code>

### 2.19 CSAPI\_unload

You must remove the `proc_inx` parameter from all calls to `CSAPI_unload`.

<b>Replace</b>	<code>CSAPI_unload(s, proc_inx, process);</code>
<b>With</b>	<code>CSAPI_unload(s, process);</code>

## 2.20 CSAPI\_get\_symbol\_value

You must replace the csx file name with a process handle.

<b>Replace</b>	<code>CSAPI_get_symbol_value(s, "file_name.csx", symbol_name, &amp;symbol_value);</code>
<b>With</b>	<code>CSAPI_get_symbol_value(s, process, symbol_name, &amp;symbol_value);</code>

## 2.21 CSAPI\_get\_symbol\_value\_loaded

You must replace calls to the `CSAPI_get_symbol_value_loaded` function with a call to the `CSAPI_get_symbol_value` function, which now takes the same parameters.

## 2.22 CSAPI\_get\_free\_mem

You must replace calls to the `CSAPI_get_free_mem` function with a call to the `CSAPI_get_free_memory` function and use an enumerated value for the memory type parameter.

<b>Replace</b>	<code>CSAPI_get_free_mem(s, proc_inx, 0, &amp;size);</code>
<b>With</b>	<code>CSAPI_get_free_memory(s, proc_inx, CSM_Dram, &amp;size);</code>

## 2.23 CSAPI\_allocate\_shared\_memory

You must use an enumerated value for the memory type parameter and provide a process handle if you are using the `symbol_name` parameter.

<b>Replace</b>	<code>CSAPI_allocate_shared_memory(s, proc_inx, 0, size, 8, symbol_name, &amp;address);</code>
<b>With</b>	<code>CSAPI_allocate_shared_memory(s, proc_inx, CSM_Dram, size, 8, process, symbol_name, &amp;address);</code>

If you are not using the `symbol_name` parameter (the parameter is `NULL`) then you must also use `NULL` for the process handle parameter.

## 2.24 CSAPI\_allocate\_static\_shared\_memory

You must use an enumerated value for the memory type parameter and provide a process handle if you are using the `symbol_name` parameter.

<b>Replace</b>	<code>CSAPI_allocate_static_shared_memory(s, proc_inx, 0, address, size, symbol_name);</code>
<b>With</b>	<code>CSAPI_allocate_static_shared_memory(s, proc_inx, CSM_Dram, address, size, process, symbol_name);</code>

If you are not using the `symbol_name` parameter (the parameter is `NULL`) then you must also use `NULL` for the process handle parameter.

## 2.25 CSAPI\_free

You must replace calls to the `CSAPI_free` function with a call to the `CSAPI_free_memory` function. You must also remove the `proc_inx` parameter.

<b>Replace</b>	<code>CSAPI_free(s, proc_inx, address);</code>
<b>With</b>	<code>CSAPI_free_memory(s, address);</code>

## 2.26 CSAPI\_write\_mono\_memory

You must replace the processor index with a transfer parameters structure. Use the `CSAPI_TRANSFER_PARAMS_SAFE` structure to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_write_mono_memory(s, proc_inx, address, size, &amp;buffer);</code>
<b>With</b>	<code>CSAPI_write_mono_memory(s, CSAPI_TRANSFER_PARAMS_SAFE, address, size, &amp;buffer);</code>

## 2.27 CSAPI\_read\_mono\_memory

You must replace the processor index with a transfer parameters structure. Use the `CSAPI_TRANSFER_PARAMS_SAFE` structure to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_read_mono_memory(s, proc_inx, address, size, &amp;buffer);</code>
<b>With</b>	<code>CSAPI_read_mono_memory(s, CSAPI_TRANSFER_PARAMS_SAFE, address, size, &amp;buffer);</code>

## 2.28 CSAPI\_write\_mono\_memory\_raw

You must replace the call to `CSAPI_write_mono_memory_raw` with a call to `CSAPI_write_mono_memory` using the `CSAPI_TRANSFER_PARAMS_FAST` structure to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_write_mono_memory_raw(s, address, size, &amp;buffer);</code>
<b>With</b>	<code>CSAPI_write_mono_memory(s, CSAPI_TRANSFER_PARAMS_FAST, address, size, &amp;buffer);</code>

### 2.29 CSAPI\_read\_mono\_memory\_raw

You must replace the call to `CSAPI_read_mono_memory_raw` with a call to `CSAPI_read_mono_memory` using the `CSAPI_TRANSFER_PARAMS_FAST` structure to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_read_mono_memory_raw(s, address, size, &amp;buffer);</code>
<b>With</b>	<code>CSAPI_read_mono_memory(s, CSAPI_TRANSFER_PARAMS_FAST, address, size, &amp;buffer);</code>

### 2.30 CSAPI\_write\_mono\_memory\_async

You must replace the processor index with a transfer parameters structure. Use the `CSAPI_TRANSFER_PARAMS_SAFE` structure to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_write_mono_memory_async(s, proc_inx, address, size, &amp;buffer);</code>
<b>With</b>	<code>CSAPI_write_mono_memory_async(s, CSAPI_TRANSFER_PARAMS_SAFE, address, size, &amp;buffer);</code>

### 2.31 CSAPI\_read\_mono\_memory\_async

You must replace the processor index with a transfer parameters structure. Use the `CSAPI_TRANSFER_PARAMS_SAFE` structure to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_read_mono_memory_async(s, proc_inx, address, size, &amp;buffer);</code>
<b>With</b>	<code>CSAPI_read_mono_memory_async(s, CSAPI_TRANSFER_PARAMS_SAFE, address, size, &amp;buffer);</code>

### 2.32 CSAPI\_write\_mono\_memory\_async\_wait

You must use a value of CSAPI\_NO\_TIMEOUT for the new timeout\_ms parameter to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_write_mono_memory_async_wait(s);</code>
<b>With</b>	<code>CSAPI_write_mono_memory_async_wait(s, CSAPI_NO_TIMEOUT);</code>

### 2.33 CSAPI\_read\_mono\_memory\_async\_wait

You must use a value of CSAPI\_NO\_TIMEOUT for the new timeout\_ms parameter to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_read_mono_memory_async_wait(s);</code>
<b>With</b>	<code>CSAPI_read_mono_memory_async_wait(s, CSAPI_NO_TIMEOUT);</code>

### 2.34 CSAPI\_write\_mono\_memory\_async\_poll

You must replace the call to CSAPI\_write\_mono\_memory\_async\_poll with a call to CSAPI\_write\_mono\_memory\_async\_wait using a value of 0 for the new timeout\_ms parameter. The value of the old completed parameter must be determined from the return status.

<b>Replace</b>	<code>CSAPI_write_mono_memory_async_poll(s, &amp;completed);</code>
<b>With</b>	<code>status = CSAPI_write_mono_memory_async_wait(s, 0); completed = (status == CSAPIErrno_success);</code>

### 2.35 CSAPI\_read\_mono\_memory\_async\_poll

You must replace the call to CSAPI\_read\_mono\_memory\_async\_poll with a call to CSAPI\_read\_mono\_memory\_async\_wait using a value of 0 for the new timeout\_ms parameter. The value of the old completed parameter must be determined from the return status.

<b>Replace</b>	<code>CSAPI_read_mono_memory_async_poll(s, &amp;completed);</code>
<b>With</b>	<code>status = CSAPI_read_mono_memory_async_wait(s, 0); completed = (status == CSAPIErrno_success);</code>

## 2.36 CSAPI\_num\_semaphores

You must replace calls to the `CSAPI_num_semaphores` function with a call to the `CSAPI_get_free_semaphores` function and use a semaphore type of `CST_Tsc`.

<b>Replace</b>	<code>CSAPI_num_semaphores(s, proc_inx, &amp;num_of_semaphores);</code>
<b>With</b>	<code>CSAPI_get_free_semaphores(s, proc_inx, CST_Tsc, &amp;num_of_semaphores);</code>

## 2.37 CSAPI\_semaphore\_signal

You must call `CSAPI_allocate_static_shared_semaphore` to allocate and obtain a semaphore handle. You must then replace the `proc_inx` and `sem_number` parameters with this semaphore handle.

<b>Replace</b>	<code>CSAPI_semaphore_signal(s, proc_inx, sem_number);</code>
<b>With</b>	<code>struct CSAPISemaphore* semaphore; CSAPI_allocate_static_shared_semaphore(s, proc_inx, CST_Tsc, sem_number, NULL, NULL, &amp;semaphore); CSAPI_semaphore_signal(s, semaphore, 0);</code>

If multiple boards or processors are being used then a semaphore handle will be required for each processor on each board.

Use `CSAPI_allocate_static_shared_semaphore(s[board], proc_inx, CST_Tsc, sem_number, NULL, NULL, &semaphore[board][proc_inx])`.

This array of semaphores should then be used instead of a single semaphore handle when calling the `CSAPI_semaphore_signal`, `CSAPI_semaphore_wait` and `CSAPI_free_semaphore` functions.

You must call `CSAPI_free_semaphore` with the semaphore handle when the semaphore is no longer needed.

## 2.38 CSAPI\_semaphore\_wait

You must replace the `proc_inx` and `sem_number` parameters with a semaphore handle. (See [2.37: CSAPI\\_semaphore\\_signal](#)). You must also use a timeout value of `CSAPI_NO_TIMEOUT` to continue the previous behavior.

<b>Replace</b>	<code>CSAPI_semaphore_wait(s, proc_inx, sem_number);</code>
<b>With</b>	<code>CSAPI_semaphore_wait(s, semaphore, NULL, CSAPI_NO_TIMEOUT);</code>

### 2.39 CSAPI\_get\_callback

You must use an enumerated value for the event type parameter.

<b>Replace</b>	<code>CSAPI_get_callback(s, 4, &amp;event_cb);</code>
<b>With</b>	<code>CSAPI_get_callback(s, CSE_Print, &amp;event_cb);</code>

### 2.40 CSAPI\_register\_callback

You must use an enumerated value for the event type parameter.

<b>Replace</b>	<code>CSAPI_register_callback(s, 4, &amp;event_cb, NULL);</code>
<b>With</b>	<code>CSAPI_register_callback(s, CSE_Print, &amp;event_cb, NULL);</code>

### 2.41 CSAPI\_endianness

The function `CSAPI_endianness` now takes a parameter of enumerated type `CSAPIEndianType` instead of an integer and the definitions for `CS_BIG_ENDIAN` and `CS_LITTLE_ENDIAN` have been replaced by enumerated values `CSN_BigEndian` and `CSN_LittleEndian`.

<b>Replace</b>	<code>CS_BIG_ENDIAN</code>
<b>With</b>	<code>CSN_BigEndian</code>
<b>Replace</b>	<code>CS_LITTLE_ENDIAN</code>
<b>With</b>	<code>CSN_LittleEndian</code>

## 3 Supplementary information

### 3.1 Connection

The `CSAPI_connect` function now takes `sharing_type` and `connection_type` parameters. You must use a value of `CSH_Private` for the `sharing_type` parameter and a value of `CSC_Direct` or `CSC_Socket` for the `connection_type` parameter. A value of `CSC_Direct` must be used when connecting to a card and a value of `CSC_Socket` must be used when connecting to a simulator. The `host_name` parameter is ignored when `CSC_Direct` is used.

The `CSAPI_connect` function also takes a `timeout_ms` parameter. You must use a value of zero for the `timeout_ms` parameter. This indicates that the function should not wait for the card or simulator to become available.

### 3.2 Process handles

The `CSAPI_load` function now takes the address of a process handle as a parameter. You must declare a process handle of type `struct CSAPIProcess*` and pass the address of this to the process parameter of the `CSAPI_load` function. The `CSAPI_load` function now returns the process handle directly, so the `CSAPI_get_last_loaded_handle` function has been removed.

The `CSAPI_load` function now takes a `section_names` parameter, which can be used to declare sections to be loaded to ESRAM. You must pass in a value of `NULL` for the `section_names` parameter for default behavior (`.text` section loaded to ESRAM). You can use the word "ALL" to require all sections to be loaded to ESRAM, or the word "ANY" to request as many sections as possible be loaded to ESRAM. A comma separated list of section names may also be used.

The `CSAPI_load` function also takes a `timeout_ms` parameter. You can use a value of `CSAPI_NO_TIMEOUT` for the `timeout_ms` parameter to keep the original behavior. If any stage of the load exceeds the `timeout_ms` value then the function will return `CSAPIErrno_timeout`.

The `CSAPI_run` function no longer takes a `proc_inx` parameter. You must replace the `proc_inx` parameter with the process handle provided by `CSAPI_load`. The process will be run on the processor on which it was loaded.

### 3.3 Runtime arguments

The `CSAPI_run` function takes a `csx_args` parameter. You must pass in a value of `NULL` for the `csx_args` parameter for default behavior. If a string is passed in for the `csx_args` parameter then it will be split where there are spaces and provided to the `csx` program using the `argc` and `argv` parameters of the main function. The `Cn` program will need to interpret the arguments in the normal way.

### 3.4 Memory allocation functions

The `CSAPI_get_free_mem` function has been renamed to `CSAPI_get_free_memory`, and now takes a `mem_type` parameter of enumerated type `CSAPIMemoryType` instead of an integer `mem_inx` parameter. You must replace all instances of `CSAPI_get_free_mem` with `CSAPI_get_free_memory` and replace integer values for `mem_inx` with an enumerated value of `CSM_Dram` or `CSM_Esram` for `mem_type`.

The `CSAPI_allocate_shared_memory` and `CSAPI_allocate_static_shared_memory` functions now take a `mem_type` parameter of enumerated type `CSAPIMemoryType` instead of an integer `mem_inx` parameter. You must replace integer values for `mem_inx` with an enumerated value of `CSM_Dram` or `CSM_Esram` for `mem_type`.

The `CSAPI_allocate_shared_memory` and `CSAPI_allocate_static_shared_memory` functions now require a process handle if the `symbol_name` parameter is being used. You must pass in the process handle provided by `CSAPI_load` if the `symbol_name` parameter is not `NULL`. If the `symbol_name` parameter is `NULL`, then `NULL` must also be used for the `process` parameter.

### 3.5 Memory transfer functions

The `CSAPI_read_mono_memory` and `CSAPI_write_mono_memory` functions no longer take a `proc_inx` parameter. The processor index is only required when halting the processor or flushing the cache. If it is required then it is determined from the start address of the transfer. You must replace the `proc_inx` parameter with a transfer parameters structure as follows:

There are two predefined transfer parameters structures named `CSAPI_TRANSFER_PARAMS_SAFE` and `CSAPI_TRANSFER_PARAMS_FAST`. These can be used directly, or a copy can be taken from one and modified. You must replace the `proc_inx` parameter with `CSAPI_TRANSFER_PARAMS_SAFE` for calls to `CSAPI_read_mono_memory` or `CSAPI_write_mono_memory`. You must replace the `proc_inx` parameter with `CSAPI_TRANSFER_PARAMS_FAST` for calls to `CSAPI_read_mono_memory_raw` or `CSAPI_write_mono_memory_raw`, and use the `CSAPI_read_mono_memory` or `CSAPI_write_mono_memory` function instead. The `CSAPI_read_mono_memory_raw` and `CSAPI_write_mono_memory_raw` functions have been removed.

The validity of the card side address will now be checked when `CSAPI_TRANSFER_PARAMS_SAFE` is used. This means that the mono memory transfer functions will now return `CSAPIErrno_invalid_address` for an invalid card side address, where as they would previously have returned `CSAPIErrno_success`.

The alignment of the host buffer and card side address will now be checked, and when `CSAPI_TRANSFER_PARAMS_SAFE` is used then the host buffer will be bounced via an aligned address if necessary. If `CSAPI_TRANSFER_PARAMS_FAST` is used then the host buffer will not be bounced and `CSAPIErrno_address_not_aligned` will be returned instead. Previously a transfer with badly aligned buffers would be performed slowly, without DMA, and this behavior can be achieved by setting the `alignment_fix` member of the transfer parameters structure to `CSL_NoCheck`.

## 3.6 Asynchronous memory transfer functions

The `CSAPI_read_mono_memory_async` and `CSAPI_write_mono_memory_async` functions have the same parameter changes as their blocking counter parts `CSAPI_read_mono_memory` and `CSAPI_write_mono_memory`. You must replace the `proc_inx` parameter with `CSAPI_TRANSFER_PARAMS_SAFE` for calls to `CSAPI_read_mono_memory_async` or `CSAPI_write_mono_memory_async`.

The `CSAPI_read_mono_memory_async_wait` and `CSAPI_write_mono_memory_async_wait` functions now take a `timeout_ms` parameter. You can use a value of `CSAPI_NO_TIMEOUT` for the `timeout_ms` parameter to keep the original behavior. If the transfer has not completed within the `timeout_ms` period then the function will return `CSAPIErrno_timeout`.

The `CSAPI_read_mono_memory_async_poll` and `CSAPI_write_mono_memory_async_poll` functions have been removed. You must replace calls to these functions with a call to `CSAPI_read_mono_memory_async_wait` or `CSAPI_write_mono_memory_async_wait`, and use a value of 0 for the `timeout_ms` parameter. The functions will return `CSAPIErrno_success` if the transfer has completed, and `CSAPIErrno_timeout` if the function succeeded but the transfer has not completed.

## 3.7 Semaphores

The `CSAPI_semaphore_signal` and `CSAPI_semaphore_wait` functions now take a semaphore handle instead of `proc_inx` and `sem_number` parameters. You must call `CSAPI_allocate_shared_semaphore` or `CSAPI_allocate_static_shared_semaphore` to obtain a semaphore handle. You must also replace the `proc_inx` and `sem_number` parameters for the `CSAPI_semaphore_signal` and `CSAPI_semaphore_wait` functions with the semaphore handle provided by `CSAPI_allocate_shared_semaphore` or `CSAPI_allocate_static_shared_semaphore`.

The `proc_inx` and `sem_number` integers can be passed to `CSAPI_allocate_static_shared_semaphore` to obtain the required semaphore to maintain compatibility with existing csx programs.

The `CSAPI_semaphore_signal` function now takes a data parameter of type `CSAPISemaphoreData`. You can use a positive value with semaphores of type `CST_GsuWithData`, and the value will be passed to the thread that receives the signal.

The `CSAPI_semaphore_wait` function now takes the address of a `CSAPISemaphoreData` variable as a data parameter. You can use the address of a `CSAPISemaphoreData` variable with semaphores of type `CST_GsuWithData`, and the variable will be set to the value that the semaphore was signalled with.

The `CSAPI_semaphore_wait` function now takes a `timeout_ms` parameter. You can use a value of `CSAPI_NO_TIMEOUT` for the `timeout_ms` parameter to keep the original behavior. If the semaphore does not get signalled and the signal received by the host within the `timeout_ms` period then the function will return `CSAPIErrno_timeout`.

You must call `CSAPI_free_semaphore` with the semaphore handle when the semaphore is no longer needed.

### 3.8 Memory and Register parameter types

There are now explicit types for the address, size and register data parameters: `CSAPIMemoryAddress`, `CSAPIMemorySize`, `CSAPIRegisterAddress` and `CSAPIRegisterValue`. You must use these types when declaring variables for use with the memory transfer and register access functions. This will provide forward compatibility if the underlying type of these parameters is changed. The underlying type continues to be `unsigned int`, so this change is recommended but not required.

### 3.9 Version identifiers

It is worth adding a compile time check for the version numbers declared in `csapi.h`. For example:

```
#if CSAPI_HEADER_VERSION_MAJOR != 2 || CSAPI_HEADER_VERSION_MINOR < 1
#error Incompatible CSAPI version
#endif
```

The version numbers are used to indicate the following:

- `CSAPI_HEADER_VERSION_MAJOR`: When this version number is changed then one or more changes have been made to the parameters of an existing CSAPI function. If you have used an affected function then your source code will not compile with the modified header file. You must modify your use of the affected functions to ensure correct behavior.
- `CSAPI_HEADER_VERSION_MINOR`: When only this version number is changed then your code will compile and run as with the original version. You will be able to compile with a minor version equal to or greater than the original value. Changes to the CSAPI will be limited to:
  - Some enumerations may have additional values.
  - Some structures may have additional member variables.
  - There may be additional CSAPI functions.

## Revision history

Date	Revision	Changes
October 2007	1.0	New document.
December 2007	1.1	Additions following Beta review.
December 2007	1.2	Made updates from development team.
January 2008	1.3	Clarification after further review.
January 2008	1.D	Changes to document number system.
September 2008	1.E	Update to copyright.
September 2010	1.F	Updated company information

**ClearSpeed Technology Ltd**

130 Aztec West  
Park Avenue  
Bristol BS32 4UB  
United Kingdom

Tel: +44 (0)1454 629 623

Fax: +44 (0)1454 629 624

**Email:** [info@clearspeed.com](mailto:info@clearspeed.com)

**Web:** <http://www.clearspeed.com>

**Support:** <http://support.clearspeed.com>

1. Information and data contained in this document, together with the information contained in any and all associated ClearSpeed documents including without limitation, data sheets, application notes and the like ('Information') is provided in connection with ClearSpeed products and is provided for information only. Quoted figures in the Information, which may be performance, size, cost, power and the like are estimates based upon analysis and simulations of current designs and are liable to change.
2. Such Information does not constitute an offer of, or an invitation by or on behalf of ClearSpeed, or any ClearSpeed affiliate to supply any product or provide any service to any party having access to this Information. Except as provided in ClearSpeed Terms and Conditions of Sale for ClearSpeed products, ClearSpeed assumes no liability whatsoever.
3. ClearSpeed products are not intended for use, whether directly or indirectly, in any medical, life saving and/ or life sustaining systems or applications.
4. The worldwide intellectual property rights in the Information and data contained therein is owned by ClearSpeed. No license whether express or implied either by estoppel or otherwise to any intellectual property rights is granted by this document or otherwise. You may not download, copy, adapt or distribute this Information except with the consent in writing of ClearSpeed.
5. The system vendor remains solely responsible for any and all design, functionality and terms of sale of any product which incorporates a ClearSpeed product including without limitation, product liability, intellectual property infringement, warranty including conformance to specification and or performance.
6. Any condition, warranty or other term which might but for this paragraph have effect between ClearSpeed and you or which would otherwise be implied into or incorporated into the Information (including without limitation, the implied terms of satisfactory quality, merchantability or fitness for purpose), whether by statute, common law or otherwise are hereby excluded.
7. ClearSpeed reserves the right to make changes to the Information or the data contained therein at any time without notice.

© Copyright ClearSpeed Technology Ltd 2010. All rights reserved.

Advance is a registered trademark of ClearSpeed Technology Ltd

ClearSpeed, ClearConnect, Advance and the ClearSpeed logo are trade marks or registered trade marks of ClearSpeed Technology Ltd. All other brands and names are the property of their respective owners.