CORSAIR

iCUE SDK

Overview and Reference

Protocol version 18

Table of Contents

Overview

The iCUE SDK gives ability for third-party applications to control lightings on Corsair RGB devices. iCUE SDK interacts with hardware through iCUE so it should be running in order for SDK to work properly.

SDK features are supported in iCUE version 4.31 or higher.

To use this SDK you should have basic knowledge in C and library linking.

SDK Package

The following folders are included:

- **include** contains C/C++ header files with function prototypes and enum declarations;
- **redist** contains both 32 and 64 bit .dll files;
- **lib** contains companion .lib files to access exported functions (32 and 64 bit);
- **examples** contains sample project that shows how to use SDK;
- **doc** contains SDK documentation (this document).

Requirements

This SDK can be used on the same platforms that iCUE does:

- **Windows 7** (32-bit and 64-bit);
- **Windows 8** (32-bit and 64-bit);
- **Windows 10** (32-bit and 64-bit).
- **Windows 11** (32-bit and 64-bit).

Other considerations

LED identification

All supported devices have one or more LED groups. An LED can only belong to one LED group. Each LED within an LED group has its own unique identifier which in this document is defined as the CorsairLedLuid type. The CorsairLedLuid value consists of a group and an LED index/id, structured as follows.

bits 31 .. 16	bits 15 .. 0
LED GROUP	LED INDEX / ID

- LED group - CorsairLedGroup value
- LED index/id - either a number starting from 1, or a value of specific enumeration (e.g., CorsairLedId_Keyboard). 0 indicates an invalid value.

Examples:

- 0x00000005 - group is CLG_Keyboard (0), LED is F4 (index is 5, which is CLK_F4 in CorsairLedId_Keyboard)
- 0x00010001 - group is CLG_KeyboardGKeys (1), LED is G1 key (index is 1)
- 0x00060001 - group is CLG_Headset (6), LED is the first one on the headset (typically, left)

Single-color devices

If a connected device only has LEDs of one color instead of all three (RGB) then when RGB color is set to such leds SDK chooses a maximum of three (RGB) values and uses it as brightness for LED.

On/off leds

If a connected device has some LEDs that support only on/off control (as opposed to 8bit) then if supplied brightness value is >= 128 such LED will be switched on, otherwise it will be switched off.

LEDs that are not controlled by SDK

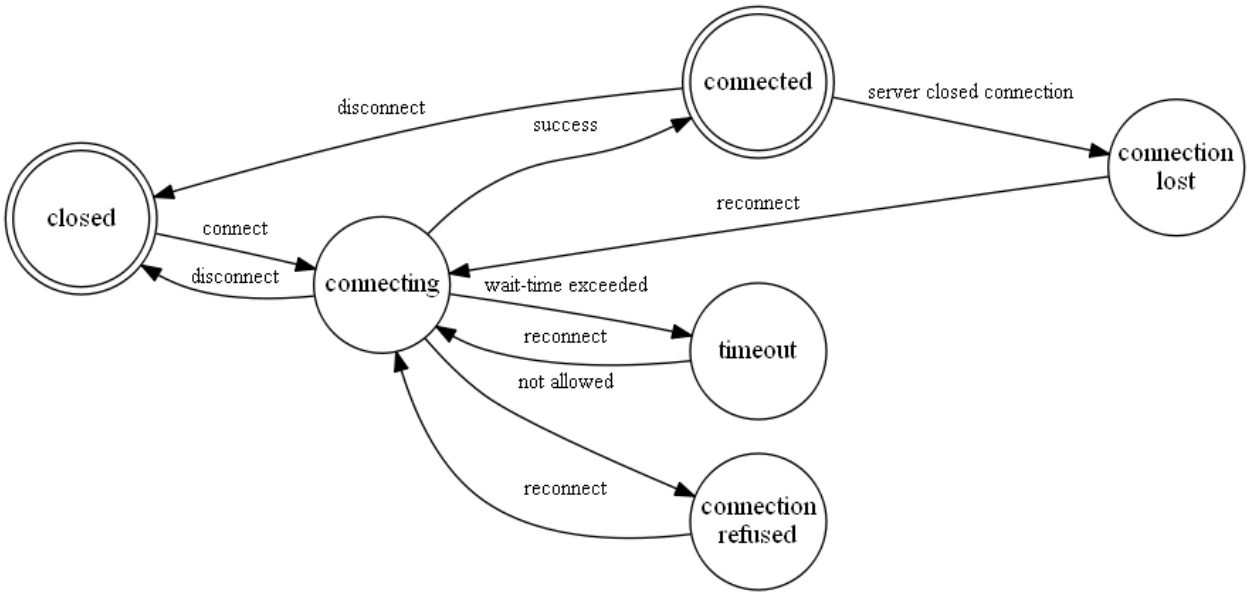
Side LEDs on CORSAIR STRAFE keyboards cannot be controlled by SDK. These LEDs remain controlled by iCUE regardless connected SDK clients.

Some devices have a certain number of service LED indicators (for example, WinLock or Profile LEDs on keyboards) and depending on device model iCUE may, optionally or unconditionally, prohibit controlling these LEDs by SDK. If, however, despite the

prohibition on controlling the service LEDs, SDK tries to set a color for them, the new color settings will be ignored for these service LEDs without an error.

Transition between session states

Session states are represented by CorsairSessionState enumeration. SDK client starts with CSS_Closed state. When CorsairConnect function is called, SDK sends notification that it switches to CSS_Connecting state. From CSS_Connecting it may end up with CSS_Connected (if everything is ok), CSS_Timeout (when iCUE was not found) or CSS_ConnectionRefused (if third-party control is disabled in iCUE settings). SDK performs automatic reconnects to go from failing state back to CSS_Connecting.



After transitioning to CSS_Connected state SDK will notify client application about all connected devices by sending corresponding event for each device. After transitioning to CSS_Closed or CSS_ConnectionLost state SDK will notify client application that all devices have been disconnected by sending corresponding event for each device, also for all G/M/S keys which are in a pressed state at the time one of these states occurred SDK should notify client application that these keys have been released.

Win+L

iCUE will preserve communication channels with SDK clients when user locks screen, so when user session is restored and set of connected devices is unchanged the client can continue using SDK as if the session was never locked.

When user locks screen SDK will perform transition to CSS_ConnectionLost state.

When user unlocks screen SDK will perform transition to CSS_Connected state.

RDP limitations

Both iCUE and SDK are designed to work on a local computer and will not work if accessed remotely through Microsoft RDP.

Access rights

In order to be able to communicate with each other, iCUE and SDK client must run with identical access rights: either with regular access or "As Administrator"

Reference

CorsairSetLedColors

```
CorsairError CorsairSetLedColors(
  const CorsairDeviceId deviceId,
  int size,
  const CorsairLedColor* ledColors);
```

Description: sets specified LEDs to some colors. The color is retained until changed by successive calls. This function does not take logical layout into account. This function executes synchronously, if you are concerned about delays consider using *CorsairSetLedColorsBuffer* together with *CorsairSetLedColorsFlushBufferAsync*.

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *int size* - number of leds in ledColors array
- *const CorsairLedColor* ledColors* - array containing colors for each LED

Returns: error code. *CE_Success* if successful. If there is no such ledId present in currently connected hardware (missing key in physical keyboard layout) then functions completes successfully and returns *CE_Success*.

Possible errors:

- *CE_NotConnected*, *CE_DeviceNotFound*
- *CE_InvalidArguments* - if deviceId is NULL

> ***Note***
>
> if ledColors array contains duplicates of some led ids, then color from the last item should be applied (ie "last win" strategy).

CorsairSetLedColorsBuffer

```
CorsairError CorsairSetLedColorsBuffer(
  const CorsairDeviceId deviceId,
  int size,
  const CorsairLedColor* ledColors);
```

Description: sets specified LEDs to some colors. This function sets LEDs colors in the buffer which is written to the devices via *CorsairSetLedColorsFlushBufferAsync*. Typical usecase is next: *CorsairSetLedColorsFlushBufferAsync* is called to write LEDs colors to the device and follows after one or more calls of *CorsairSetLedColorsBuffer* to set the LEDs buffer. This function does not take logical layout into account.

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *int size* - number of leds in ledColors array
- *const CorsairLedColor* ledColors* - array containing colors for each LED

Returns: error code. *CE_Success* if successful. If there is no such ledId present in currently connected hardware (missing key in physical keyboard layout) then function completes successfully and returns *CE_Success*.

Possible errors:

- *CE_NotConnected*, *CE_DeviceNotFound*
- *CE_InvalidArguments* - if deviceId is NULL

> ***Note***
>
> if ledColors array contains duplicates of some led ids, then color from the last item should be applied (ie "last win" strategy).

CorsairSetLedColorsFlushBufferAsync

```
typedef void(*CorsairAsyncCallback)(
  void *context,
  CorsairError error);

CorsairError CorsairSetLedColorsFlushBufferAsync(
  CorsairAsyncCallback callback,
  void *context);
```

Description: writes to the devices LEDs colors buffer which is previously filled by the *CorsairSetLedColorsBuffer* function. This function executes asynchronously and returns control to the caller immediately.

Input arguments:

- *CorsairAsyncCallback callback* - callback that is called by SDK when colors are set. Can be NULL if client is not interested in result. Callback parameters:
    - *context* - contains value that was supplied by user in *CorsairSetLedColorsFlushBufferAsync* call.
    - *error* - error code. *CE_Success* if the call was successful. **Possible errors:** *CE_NotConnected*, *CE_NoControl*
- *void* context* - arbitrary context that will be returned in callback call. Can be NULL

Returns: error code. *CE_Success* if successful. If there is no such ledId present in currently connected hardware (missing key in physical keyboard layout, or trying to control mouse while it's disconnected) then function completes successfully and returns *CE_Success*.

Possible errors:

- *CE_NotConnected*

CorsairGetDevices

```
CorsairError CorsairGetDevices(
  const CorsairDeviceFilter *filter,
  int sizeMax,
  CorsairDeviceInfo *devices,
  int *size);
```

Description: populates the buffer with filtered collection of devices

Input arguments:

- *const CorsairDeviceFilter *filter* - pointer to filter, that describes which of devices SDK should put to the devices array
- *int sizeMax* - size of devices array (should typically be equal to *CORSAIR_DEVICE_COUNT_MAX*)
- *CorsairDeviceInfo *devices* - preallocated buffer that SDK should populate after successful call with information about connected devices that satisfy search criteria specified by filter parameter.
- *int *size* - pointer to memory where SDK should store the actual number of items written to devices array. The value should always be less than or equal to sizeMax

Returns: error code. *CE_Success* if successful. Also devices array will contain information about devices and size - number of devices in array on return.

Possible errors:

- *CE_NotConnected*
- *CE_InvalidArguments* - if filter is invalid or NULL, if devices array is NULL or too small

CorsairGetDeviceInfo

```
CorsairError CorsairGetDeviceInfo(
  const CorsairDeviceId deviceId,
  CorsairDeviceInfo *deviceInfo);
```

Description: gets information about device specified by deviceId

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *CorsairDeviceInfo *deviceInfo* - preallocated buffer that SDK should populate after successful call with information about connected device

Returns: error code. CE_Success if successful. Also deviceInfo will contain valid values read from the device on return.

Possible errors:

- *CE_NotConnected*, *CE_DeviceNotFound*
- *CE_InvalidArguments* - if deviceInfo is NULL

CorsairGetDevicePropertyInfo

```
CorsairError CorsairGetDevicePropertyInfo(
  const CorsairDeviceId deviceId,
  CorsairDevicePropertyId propertyId,
  unsigned int index,
  CorsairDataType *dataType,
  unsigned int *flags);
```

Description: gets information about device property for the device specified by deviceId

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *CorsairDevicePropertyId propertyId* - device property identifier
- *unsigned int index* - zero based property index; used when property has *CPF_Indexed* flag set, otherwise should be 0
- *CorsairDataType *dataType* - a pointer to the variable to store the returned value of the *CorsairDataType* enumeration that specifies the type of property data
- *unsigned int *flags* - a pointer to the variable to store the returned flag values that describe operations that can be applied to the property. The value is formed as a bitwise "or" of *CorsairPropertyFlag* enum values

Returns: error code. *CE_Success* if successful. Also dataType and flags will contain valid values read from device on return.

Possible errors:

- *CE_NotConnected*, *CE_DeviceNotFound*
- *CE_InvalidArguments* - if deviceId is NULL, if dataType is NULL or flags is NULL
- *CE_InvalidOperation* - if property does not exist for specified device

CorsairReadDeviceProperty

```
CorsairError CorsairReadDeviceProperty(
  const CorsairDeviceId deviceId,
  CorsairDevicePropertyId propertyId,
  unsigned int index,
  CorsairProperty *property);
```

Description: gets the data of device property by device identifier, property identifier and property index

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *CorsairDevicePropertyId propertyId* - device property identifier
- *unsigned int index* - zero based property index; used when property has *CPF_Indexed* flag set, otherwise should be 0
- *CorsairProperty *property* - a variable to store a pointer to the property data. When SDK client is finished using *CorsairProperty* instance it should call *CorsairFreeProperty* function to free memory occupied by struct and its members

Returns: error code. *CE_Success* if successful. Also property will contain a pointer to valid property data read from device on return.

Possible errors:

- *CE_NotConnected*, *CE_DeviceNotFound*
- *CE_InvalidArguments* - if deviceId is NULL or property is NULL
- *CE_InvalidOperation* - if property does not exist for specified device

CorsairWriteDeviceProperty

```
CorsairError CorsairWriteDeviceProperty(
  const CorsairDeviceId deviceId,
  CorsairDevicePropertyId propertyId,
  unsigned int index,
  const CorsairProperty *property);
```

Description: sets the data of device property by device identifier, property identifier and property index. Can be called only with writable properties

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *CorsairDevicePropertyId propertyId* - device property identifier
- *unsigned int index* - zero based property index; used when property has *CPF_Indexed* flag set, otherwise should be 0
- *const CorsairProperty *property* - a pointer to the property data

Returns: error code. *CE_Success* if successful.

Possible errors:

- *CE_NotConnected*, *CE_DeviceNotFound*
- *CE_InvalidArguments* - if deviceId is NULL or property is NULL
- *CE_InvalidOperation* - if property does not exist for specified device

CorsairFreeProperty

```
CorsairError CorsairFreeProperty(CorsairProperty *property);
```

Description: frees memory occupied by *CorsairProperty* instance

Input arguments:

- *CorsairProperty *property* - a pointer to instance of *CorsairProperty* for which memory was allocated as a result of a call to *CorsairReadDeviceProperty*

Returns: error code. CE_Success if successful.

Possible errors: - *CE_InvalidArguments* - if property is NULL - *CE_InvalidOperation* - if SDK failed to free memory for some reason

CorsairGetLedPositions

```
CorsairError CorsairGetLedPositions(
  const CorsairDeviceId deviceId,
  int sizeMax,
  CorsairLedPosition *ledPositions,
  int *size);
```

Description: provides a list of supported device LEDs by its id with their positions. Position could be either physical (only device-dependent) or logical (depend on device as well as iCUE settings).

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *int sizeMax* - size of ledPositions array (should typically be equal to *CORSAIR_DEVICE_LEDCOUNT_MAX*)
- *CorsairLedPosition *ledPositions* - preallocated buffer that SDK should populate after successful call with the list of LEDs positions

- *int *size* - pointer to memory where SDK should store the actual number of items written to ledPositions array. The value should always be less than or equal to sizeMax

Returns: error code. *CE_Success* if successful. Also ledPositions array will contain information about LEDs positions and size - number of items in this array on return.

Possible errors:

- *CE_NotConnected*, *CE_DeviceNotFound*
- *CE_InvalidArguments* - if deviceId is NULL, if ledPositions array is NULL or too small

CorsairGetLedLuidForKeyName

```
CorsairError CorsairGetLedLuidForKeyName(
  const CorsairDeviceId deviceId,
  char keyName,
  CorsairLedLuid *luid);
```

Description: retrieves LED luid for key name taking logical layout into account. So on AZERTY keyboards if user calls *CorsairGetLedLuidForKeyName(deviceId, 'A', &luid)* he gets luid with *CLK_Q* code. This luid can be used in *CorsairSetLedColors* or *CorsairSetLedColorsBuffer* function

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *char keyName* - key name. ['A'..'Z'] (26 values) are valid values
- *CorsairLedLuid *luid* - pointer to memory where SDK should store proper *CorsairLedLuid* or *CLI_Invalid* if error occurred

Returns: error code. *CE_Success* if successful. Also ledId will contain proper *CorsairLedId* on return

Possible errors:

- *CE_NotConnected*, *CE_DeviceNotFound*
- *CE_InvalidArguments* - if deviceId is NULL, if keyName is invalid

CorsairRequestControl

```
CorsairError CorsairRequestControl(
  const CorsairDeviceId deviceId,
  CorsairAccessLevel accessLevel);
```

Description: requests control using specified access level. By default client has shared control over lighting and events so there is no need to call *CorsairRequestControl()* unless client requires exclusive control

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier. Can be NULL to request control for all devices
- *CorsairAccessLevel accessLevel* - requested access level

Returns: error code. *CE_Success* if SDK received requested control

Possible errors:

- *CE_NotConnected*, *CE_DeviceNotFound*, *CE_NoControl*
- *CE_InvalidArguments* - if provided accessLevel is not supported by this version of SDK
- *CE_NotAllowed* - if exclusive control is disabled in iCUE settings

CorsairReleaseControl

```
CorsairError CorsairReleaseControl(const CorsairDeviceId deviceId);
```

Description: releases previously requested control for specified device. This action resets access level to default (shared)

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier. Can be NULL to release control for all devices

Returns: error code. *CE_Success* if SDK released control

Possible errors:

- *CE_NotConnected*, *CE_DeviceNotFound*
- *CE_InvalidArguments* - if deviceId is NULL

Behavior: when this function is called iCUE should clear all colors set by previous calls to *CorsairSetLedsColors*`*` functions if client had exclusive lighting access level and devices should show lightings that come from iCUE. Client may request exclusive control again after calling *CorsairReleaseControl*

CorsairGetLedColors

```
CorsairError CorsairGetLedColors(
  const CorsairDeviceId deviceId,
  int size,
  CorsairLedColor* ledColors);
```

Description: get current color for the list of requested LEDs of supported device. The color should represent the actual state of the hardware LED, which could be a combination of SDK and/or iCUE input.

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *int size* - number of leds in ledColors array
- *CorsairLedColor* ledColors* - array containing colors for each LED. Caller should only fill ledId field, and then SDK will fill R, G, B and A values on return.

Returns: error code. *CE_Success* if successful. If there is no such ledId present in currently connected hardware (missing key in physical keyboard layout, or trying to control mouse while it's disconnected) then function completes successfully and returns *CE_Success*. Also ledColors array will contain R, G, B and A values of colors on return

Possible errors:

- *CE_NotConnected*, *CE_DeviceNotFound*
- *CE_InvalidArguments* - if deviceId is NULL; if array contains duplicates of some led ids

CorsairSetLayerPriority

```
CorsairError CorsairSetLayerPriority(unsigned int priority);
```

Description: set layer priority for this shared client. By default iCUE has priority of 127 and all shared clients have priority of 128 if they don't call this function. Layers with higher priority value are shown on top of layers with lower priority.

Input arguments:

- *unsigned int priority* - priority of a layer [0..`CORSAIR_LAYER_PRIORITY_MAX`]

Returns: error code. *CE_Success* if successful. If this function is called in exclusive mode then it will return *CE_Success*

Possible errors:

- *CE_NotConnected*
- *CE_InvalidArguments* - if priority value is beyond [0..255] interval

CorsairSubscribeForEvents

```
typedef void(*CorsairEventHandler)(
  void *context,
  const CorsairEvent *event);

CorsairError CorsairSubscribeForEvents(
  CorsairEventHandler onEvent,
  void *context);
```

Description: registers a callback that will be called by SDK when some event happened. If client is already subscribed but calls this function again SDK should use only last callback registered for sending notifications

Input arguments:

- *CorsairEventHandler onEvent* - callback that is called by SDK when some event happened. Callback parameters:
    - *context* - contains value that was supplied by user in *CorsairSubscribeForEvents* call.
    - *event* - information about event, user can distinguish between events by reading *event->id* field.
- *void* context* - arbitrary context that will be returned in callback call. Can be NULL

Returns: error code. *CE_Success* if successful.

Possible errors:

- *CE_NotConnected*

- *CE_InvalidArguments* - if onEvent is NULL

CorsairUnsubscribeFromEvents

```
CorsairError CorsairUnsubscribeFromEvents();
```

Description: unregisters callback previously registered by *CorsairSubscribeForEvents* call

Input arguments: no

Returns: error code. *CE_Success* if successful.

Possible errors:

- *CE_NotConnected*

CorsairConfigureKeyEvent

```
CorsairError CorsairConfigureKeyEvent(
  const CorsairDeviceId deviceId,
  const CorsairKeyEventConfiguration *config);
```

Description: asks iCUE to send a key event from the device specified by deviceId to an exclusive SDK client only (if *config->isIntercepted* == *true*) or to all SDK clients including active exclusive client (if *config->isIntercepted* == *false*). Effective only for SDK clients with *ExclusiveKeyEventsListening* or *ExclusiveLightingControlAndKeyEventsListening* access level (see *CorsairRequestControl*). This function gives possibility for a client with *ExclusiveKeyEventsListening* or *ExclusiveLightingControlAndKeyEventsListening* access level to control selected set of macro keys only and let iCUE to pass some key events to other shared clients

Input arguments:

- *const CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *CorsairKeyEventConfiguration* config* - key event configuration. If *config->isIntercepted* equals true then iCUE should pass the event to an active exclusive SDK client only. If false then iCUE should send it to all the clients.

Returns: error code. *CE_Success* if configuration was successfully applied or it was not changed (but no error occurred).

Possible errors:

- *CE_NotConnected*
- *CE_NotAllowed* - if exclusive control is disabled in iCUE settings
- *CE_NoControl* - if some other client has or took over exclusive control
- *CE_InvalidOperation* - if client has insufficient access level (must have *ExclusiveKeyEventsListening* or *ExclusiveLightingControlAndKeyEventsListening* access level)
- *CE_InvalidArguments* - if deviceId or config is NULL

CorsairConnect

```
typedef void(*CorsairSessionStateChangedHandler)(
  void *context,
  const CorsairSessionStateChanged *eventData);

CorsairError CorsairConnect(
  CorsairSessionStateChangedHandler onStateChanged,
  void *context);
```

Description: sets handler for session state changes, checks versions of SDK client, server and host (iCUE) to understand which of SDK functions can be used with this version of iCUE

Input arguments:

CorsairSessionStateChangedHandler onStateChanged - callback that is called by SDK when session state changed. Callback parameters:

  - *context* - contains value that was supplied by user in *CorsairConnect* call.
  - *eventData* - information about new session state and client/server versions.
- *void* context* - arbitrary context that will be returned in callback call. Can be NULL

Returns: error code. *CE_Success* if successful.

Possible errors:

- *CE_InvalidArguments* - if onStateChanged is NULL
- *CE_InvalidOperation* - if SDK client is already connected or is connecting

CorsairGetSessionDetails

```
CorsairError CorsairGetSessionDetails(
  CorsairSessionDetails* details);
```

Description: checks versions of SDK client, server and host (iCUE) to understand which of SDK functions can be used with this version of iCUE. If there is no active session or client is not connected to the server, then only client version will be filled

Input arguments:

- *CorsairSessionDetails* details* - pointer to preallocated memory where to store information about SDK and iCUE versions.

Returns: error code. *CE_Success* if successful.

Possible errors:

CE_InvalidArguments - if details is NULL

CorsairDisconnect

```
CorsairError CorsairDisconnect();
```

Description: removes handler for session state changes previously set by *CorsairConnect*

Input arguments: no

Returns: error code. *CE_Success* if successful.

Possible errors:

- *CE_NotConnected*

Behavior: before function returns session state event will be triggered with state changed to *CSS_Closed*

CorsairLedGroup

```
enum CorsairLedGroup
```

Description: contains a list of led groups. Led group is used as a part of led identifier

Items:

- *CLG_Keyboard* - for keyboard leds
- *CLG_KeyboardGKeys* - for keyboard leds on G keys
- *CLG_KeyboardEdge* - for keyboard lighting pipe leds
- *CLG_KeyboardOem* - for vendor specific keyboard leds (ProfileSwitch, DialRing, etc.)
- *CLG_Mouse* - for mouse leds
- *CLG_Mousemat* - for mousemat leds
- *CLG_Headset* - for headset leds
- *CLG_HeadsetStand* - for headset stand leds
- *CLG_MemoryModule* - for memory module leds
- *CLG_Motherboard* - for motherboard leds
- *CLG_GraphicsCard* - for graphics card leds
- *CLG_DIY_Channel1* - for leds on the first channel of DIY devices and coolers
- *CLG_DIY_Channel2* - for leds on the second channel of DIY devices and coolers
- *CLG_DIY_Channel3* - for leds on the third channel of DIY devices and coolers
- *CLG_Touchbar* - for touchbar leds
- *CLG_GameController* - for game controller leds

CorsairLedId_Keyboard

```
enum CorsairLedId_Keyboard
```

Description: contains a list of keyboard leds that belong to *CLG_Keyboard* group

Item samples:

- *CLI_Invalid* - dummy value
- *CLK_F1*, *CLK_Esc*, *CLK_Q*, *CLK_ 1*, *CLK_ArrowUp*, ... - for keyboard leds

CorsairMacroKeyId

```
enum CorsairMacroKeyId
```

Description: contains a shared list of G, M and S keys (not all available keys!)

Item samples:

- *CMKI_Invalid* - dummy value
- *CMKI_1*, ... *CMKI_20* - for keyboard G keys, mouse M keys or touchbar S keys

CorsairDeviceType

```
enum CorsairDeviceType
```

Description: contains list of available device types

Items:

- *CDT_Unknown = 0x0000* - for unknown/invalid devices
- *CDT_Keyboard = 0x0001* - for keyboards
- *CDT_Mouse = 0x0002* - for mice
- *CDT_Mousemat = 0x0004* - for mousemats
- *CDT_Headset = 0x0008* - for headsets
- *CDT_HeadsetStand = 0x0010* - for headset stands
- *CDT_FanLedController = 0x0020* - for DIY-devices like Commander PRO
- *CDT_LedController = 0x0040* - for DIY-devices like Lighting Node PRO
- *CDT_MemoryModule = 0x0080* - for memory modules
- *CDT_Cooler = 0x0100* - for coolers
- *CDT_Motherboard = 0x0200* - for motherboards
- *CDT_GraphicsCard = 0x0400* - for graphics cards
- *CDT_Touchbar = 0x0800* - for touchbars
- *CDT_GameController = 0x1000* - for game controllers
- *CDT_All = 0xFFFFFFFF* - for all devices

CorsairChannelDeviceType

```
enum CorsairChannelDeviceType
```

Description: contains list of the LED-devices which can be connected to the DIY-device, memory module or cooler.

Items:

- *CCDT_Invalid* - dummy value
- *CCDT_HD_Fan*, *CCDT_SP_Fan*, *CCDT_LL_Fan*, *CCDT_ML_Fan*, *CCDT_QL_Fan*, *CCDT_8LedSeriesFan*, *CCDT_Strip*, *CCDT_DAP*, *CCDT_Pump*, *CCDT_DRAM*, *CCDT_WaterBlock*, *CCDT_QX_Fan* - valid values

CorsairPhysicalLayout

```
enum CorsairPhysicalLayout
```

Description: contains list of available physical layouts for keyboards

Items:

- *CPL_Invalid* - dummy value
- *CPL_US*, *CPL_UK*, *CPL_JP*, *CPL_KR*, *CPL_BR* - valid values

CorsairLogicalLayout

```
enum CorsairLogicalLayout
```

Description: contains list of available logical layouts for keyboards

Items:

- *CLL_Invalid* - dummy value
- *CLL_US_Int*, *CLL_NA*, *CLL_EU*, *CLL_UK*, *CLL_BE*, *CLL_BR*, *CLL_CH*, *CLL_CN*, *CLL_DE*, *CLL_ES*, *CLL_FR*, *CLL_IT*, *CLL_ND*, *CLL_RU*, *CLL_JP*, *CLL_KR*, *CLL_TW*, *CLL_MEX* - valid values

CorsairSessionState

```
enum CorsairSessionState
```

Description: contains a list of all possible session states

Items:

- *CSS_Invalid = 0* - dummy value
- *CSS_Closed = 1* - client not initialized or client closed connection (initial state)

- *CSS_Connecting = 2* - client initiated connection but not connected yet
- *CSS_Timeout = 3* - server did not respond, sdk will try again
- *CSS_ConnectionRefused = 4* - server did not allow connection
- *CSS_ConnectionLost = 5* - server closed connection
- *CSS_Connected = 6* - successfully connected

CorsairError

```
enum CorsairError
```

Description: contains shared list of all errors which could happen during calling of *Corsair** functions

Items:

- *CE_Success* - if previously called function completed successfully
- *CE_NotConnected* - if iCUE is not running or was shut down or third-party control is disabled in iCUE settings (runtime error), or if developer did not call *CorsairConnect* after calling *CorsairDisconnect* or on app start (developer error)
- *CE_NoControl* - if some other client has or took over exclusive control (runtime error)
- *CE_IncompatibleProtocol* - if developer is calling the function that is not supported by the server (either because protocol has broken by server or client or because the function is new and server is too old. Check *CorsairSessionDetails* for details) (developer error)
- *CE_InvalidArguments* - if developer supplied invalid arguments to the function (for specifics look at function descriptions) (developer error)
- *CE_InvalidOperation* - if developer is calling the function that is not allowed due to current state (reading improper properties from device, or setting callback when it has already been set) (developer error)
- *CE_DeviceNotFound* - if invalid device id has been supplied as an argument to the function (when device id refers to disconnected device) (runtime error)
- *CE_NotAllowed* - if specific functionality (key interception) is disabled in iCUE settings (runtime error)

CorsairAccessLevel

```
enum CorsairAccessLevel
```

Description: contains list of available SDK access levels

Items:

- *CAL_Shared = 0* - shared mode (default)
- *CAL_ExclusiveLightingControl = 1* - exclusive lightings, but shared events
- *CAL_ExclusiveKeyEventsListening = 2* - exclusive key events, but shared lightings
- *CAL_ExclusiveLightingControlAndKeyEventsListening = 3* - exclusive mode

CorsairEventId

```
enum CorsairEventId
```

Description: contains list of event identifiers

Items:

- *CEI_Invalid* - dummy value,
- *CEI_DeviceConnectionStatusChangedEvent*, *CEI_KeyEvent* - valid values

CorsairDevicePropertyId

```
enum CorsairDevicePropertyId
```

Description: contains list of properties identifiers which can be read from device.

Items:

- *CDPI_Invalid = 0* - dummy value
- *CDPI_PropertyArray = 1* - array of *CorsairDevicePropertyId* members supported by device
- *CDPI_MicEnabled = 2* - indicates Mic state (On or Off); used for headset, headset stand
- *CDPI_SurroundSoundEnabled = 3* - indicates Surround Sound state (On or Off); used for headset, headset stand

- *CDPI_SidetoneEnabled = 4* - indicates Sidetone state (On or Off); used for headset (where applicable)
- *CDPI_EqualizerPreset = 5* - the number of active equalizer preset (integer, 1 - 5); used for headset, headset stand
- *CDPI_PhysicalLayout = 6* - keyboard physical layout (see *CorsairPhysicalLayout* for valid values); used for keyboard
- *CDPI_LogicalLayout = 7* - keyboard logical layout (see *CorsairLogicalLayout* for valid values); used for keyboard
- *CDPI_MacroKeyArray = 8* - array of programmable G, M or S keys on device
- *CDPI_BatteryLevel = 9* - battery level (0 - 100); used for wireless devices
- *CDPI_ChannelLedCount = 10* - total number of LEDs connected to the channel
- *CDPI_ChannelDeviceCount = 11* - number of LED-devices (fans, strips, etc.) connected to the channel which is controlled by the DIY device
- *CDPI_ChannelDeviceLedCountArray = 12* - array of integers, each element describes the number of LEDs controlled by the channel device
- *CDPI_ChannelDeviceTypeArray = 13* - array of *CorsairChannelDeviceType* members, each element describes the type of the channel device

CorsairDataType

```
enum CorsairDataType
```

Description: contains list of available property types

Items:

- *CT_Boolean = 0* - for property of type Boolean
- *CT_Int32 = 1* - for property of type Int32 or Enumeration
- *CT_Float64 = 2* - for property of type Float64
- *CT_String = 3* - for property of type String
- *CT_Boolean_Array = 16* - for array of Boolean
- *CT_Int32_Array = 17* - for array of Int32
- *CT_Float64_Array = 18* - for array of Float64
- *CT_String_Array = 19* - for array of String

CorsairPropertyFlag

```
enum CorsairPropertyFlag
```

Description: contains list of operations that can be applied to the property

Items:

- *CPF_None = 0x00*
- *CPF_CanRead = 0x01* - describes readable property
- *CPF_CanWrite = 0x02* - describes writable property
- *CPF_Indexed = 0x04* - if flag is set, then index should be used to read/write multiple properties that share the same property identifier

CorsairLedColor

```
struct CorsairLedColor
```

Description: contains information about LED and its color

Fields:

- *CorsairLedLuid id* - unique identifier of led
- *unsigned char r* - red brightness [0..255]
- *unsigned char g* - green brightness [0..255]
- *unsigned char b* - blue brightness [0..255]
- *unsigned char a* - alpha channel [0..255]. The opacity of the color from 0 for completely translucent to 255 for opaque

CorsairLedPosition

```
struct CorsairLedPosition
```

Description: contains led id and position of led.

Fields:

- *CorsairLedLuid id* - unique identifier of led

- *double cx*, *double cy* - for keyboards, mice, mousemats, headset stands and memory modules values are in mm, for DIY-devices, headsets and coolers values are in logical units

CorsairDeviceInfo

struct CorsairDeviceInfo

Description: contains information about device

Fields:

- *CorsairDeviceType type* - enum describing device type
- *CorsairDeviceId id* - null terminated Unicode string that contains unique device identifier
- *char serial[CORSAIR_STRING_SIZE_M]* - null terminated Unicode string that contains device serial number. Can be empty, if serial number is not available for the device
- *char model[CORSAIR_STRING_SIZE_M]* - null terminated Unicode string that contains device model (like "K95RGB")
- *int ledCount* - number of controllable LEDs on the device
- *int channelCount* - number of channels controlled by the device

CorsairDeviceFilter

struct CorsairDeviceFilter

Description: contains device search filter.

Fields:

- *int deviceTypeMask* - mask that describes device types, formed as logical "or" of *CorsairDeviceType* enum values

CorsairVersion

struct CorsairVersion

Description: contains information about version that consists of three components

Fields:

- *int major*
- *int minor*
- *int patch*

CorsairSessionDetails

struct CorsairSessionDetails

Description: contains information about SDK and iCUE versions

Fields:

- *CorsairVersion clientVersion* - version of SDK client (like {4,0,1}). Always contains valid value even if there was no iCUE found. Must comply with the semantic versioning rules.
- *CorsairVersion serverVersion* - version of SDK server (like {4,0,1}) or empty struct ({0,0,0}) if the iCUE was not found. Must comply with the semantic versioning rules.
- *CorsairVersion serverHostVersion* - version of iCUE (like {4,30,58}) or empty struct ({0,0,0}) if the iCUE was not found.

CorsairSessionStateChanged

struct CorsairSessionStateChanged

Description: contains information about session state and client/server versions

Fields:

- *CorsairSessionState state* - new session state which SDK client has been transitioned to
- *CorsairSessionDetails details* - information about client/server versions

CorsairEvent

struct CorsairEvent

Description: contains information about event id and event data

Fields:

- *CorsairEventId id* - event identifier
- **Anonymous union with fields:**
    - *const CorsairDeviceConnectionStatusChangedEvent *deviceConnectionStatusChangedEvent* - when *id* == *CEI_DeviceConnectionStatusChangedEvent* contains valid pointer to structure with information about connected or disconnected device
    - *const CorsairKeyEvent *keyEvent* - when *id* == *CEI_KeyEvent* contains valid pointer to structure with information about pressed or released G, M or S button and device where this event happened

CorsairDeviceConnectionStatusChangedEvent

struct CorsairDeviceConnectionStatusChangedEvent

Description: contains information about device that was connected or disconnected

Fields:

- *CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *bool isConnected* - true if connected, false if disconnected

CorsairKeyEvent

struct CorsairKeyEvent

Description: contains information about device where G, M or S key was pressed/released and the key itself

Fields:

- *CorsairDeviceId deviceId* - null terminated Unicode string that contains unique device identifier
- *CorsairMacroKeyId keyId* - G, M or S key that was pressed/released
- *bool isPressed* - true if pressed, false if released

CorsairKeyEventConfiguration

struct CorsairKeyEventConfiguration

Description: contains information about key event configuration

Fields:

- *CorsairMacroKeyId keyId* - G, M or S key
- *bool isIntercepted* - flag that defines how key event should behave. If true then iCUE will pass the event to an active exclusive SDK client and stop passing it to the rest SDK clients. If false then iCUE will resume sending it to all SDK clients

CorsairDataType_BooleanArray

struct CorsairDataType_BooleanArray

Description: represents an array of boolean values

Fields:

- *bool* items* - an array of boolean values
- *unsigned int count* - number of items array elements

CorsairDataType_Int32Array

struct CorsairDataType_Int32Array

Description: represents an array of integer values

Fields:

- *int* items* - an array of integer values
- *unsigned int count* - number of items array elements

CorsairDataType_Float64Array

struct CorsairDataType_Float64Array

Description: represents an array of double values

Fields:

- *double* items* - an array of double values
- *unsigned int count* - number of items array elements

CorsairDataType_StringArray

struct CorsairDataType_StringArray

Description: represents an array of pointers to null terminated strings

Fields:

- *char** items* - an array of pointers to null terminated strings
- *unsigned int count* - number of items array elements

CorsairDataValue

union CorsairDataValue

Description: a union of all property data types

Fields:

- *bool boolean* - actual property value if it's type is *CPDT_Boolean*
- *int int32* - actual property value if it's type is *CPDT_Int32*
- *double float64* - actual property value if it's type is *CPDT_Float64*
- *char* string* - actual property value if it's type is *CPDT_String*
- *CorsairDataType_BooleanArray boolean_array* - actual property value if it's type is *CPDT_Boolean_Array*
- *CorsairDataType_Int32Array int32_array* - actual property value if it's type is *CPDT_Int32_Array*
- *CorsairDataType_Float64Array float64_array* - actual property value if it's type is *CPDT_Float64_Array*
- *CorsairDataType_StringArray string_array* - actual property value if it's type is *CPDT_String_Array*

CorsairProperty

struct CorsairProperty

Description: contains information about device property type and value

Fields:

- *CorsairDataType type* - type of property
- *CorsairDataValue value* - property value

End User License Agreement

On-Line End User License Agreement

IMPORTANT: This End-User License Agreement ("EULA") is a legal Agreement between you and Corsair Components Inc., and any of its affiliates and/or subsidiaries ("CORSAIR") with respect to the software, SDKs and source code provided by CORSAIR, any associated media, printed materials, "online" documentation and electronic documentation (collectively referred to as "Software"). By installing, copying, or otherwise using the Software, you agree to be bound by the terms of this EULA. "You" and "Your" may refer to a natural person or to a legal entity including, but not limited to, a corporation, partnership or a limited liability company. If you do not agree to the terms of this EULA, you are not authorized to install or use the Software.

1. Ownership of Software.

CORSAIR owns certain rights in the Software. THE SOFTWARE IS A PROPRIETARY PRODUCT OF CORSAIR OR THIRD PARTIES FROM WHOM CORSAIR HAS OBTAINED LICENSING RIGHTS. THE SOFTWARE IS PROTECTED BY COPYRIGHT LAWS AND OTHER INTELLECTUAL PROPERTY LAWS. TITLE TO THIS SOFTWARE, ANY COPY OF THIS SOFTWARE, AND ANY INTELLECTUAL PROPERTY RIGHTS IN THE SOFTWARE WILL AT ALL TIMES REMAIN WITH CORSAIR AND SUCH THIRD PARTIES. Your rights are defined by this Agreement which You agree creates a legally binding and valid contract. CORSAIR retains the right to utilize its affiliated companies, authorized distributors, authorized resellers and other third parties in pursuing any of its rights and fulfilling any of its obligations under this Agreement.

2. License Grant.

CORSAIR grants to You a nonexclusive, nontransferable (except as may be required by applicable law) royalty-free license to allow You to use the Software.

3. Your Responsibilities and Prohibited Actions.

(a) Transfer of Rights. You may not transfer or assign all or any portion of the Software, or any rights granted in this Agreement, to any other person.

(b) Reverse Engineering or Modifying the Software. You will not reverse engineer, decompile, translate, disassemble, or otherwise attempt to discover the source code of the Software. The prohibition against modifying or reverse engineering the Software does not apply to the extent that You are allowed to do so by applicable law including, but not limited to, the European Union Directive on the Interoperability of Software or its implementing legislation in the member countries. You may not otherwise modify, alter, adapt, or merge the Software.

(c) Third Party Supplier. You agree that CORSAIR's third party suppliers may enforce this Agreement as it relates to their Software directly against You.

(d) Export. CORSAIR shall not be required to undertake any action pursuant to this Agreement that is prevented by any impediments arising out of national or international foreign trade or customs requirements, including embargoes or any other sanctions. This Agreement is subject to all United States government laws and regulations as may be enacted, amended or modified from time to time regarding the export from the United States of CORSAIR software, services, technology, or any derivatives thereof. You will not export or re-export any CORSAIR software, services, technology, or any derivatives thereof, or permit the shipment of same. This section will survive the expiration or termination of this Agreement for any reason.

4. Term and Termination.

CORSAIR reserves the right to terminate this Agreement if You fail to comply with any of the terms described herein. All license rights granted will cease upon any termination of this Agreement.

5. Disclaimer of Warranty.

CORSAIR MAKES NO WARRANTIES OF ANY KIND, AND NO WARRANTY IS GIVEN THAT THE SOFTWARE IS ERROR-FREE OR THAT ITS USE WILL BE UNINTERRUPTED OR THAT IT WILL WORK IN CONNECTION WITH ANY OTHER SOFTWARE. ALL WARRANTIES, CONDITIONS, REPRESENTATIONS, INDEMNITIES AND GUARANTEES, WHETHER EXPRESS OR IMPLIED, ARISING BY LAW, CUSTOM, PRIOR ORAL OR WRITTEN STATEMENTS (INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR PARTICULAR PURPOSE OR OF ERROR-FREE AND UNINTERRUPTED USE OR ANY WARRANTY AGAINST INFRINGEMENT) ARE HEREBY OVERRIDDEN, EXCLUDED AND DISCLAIMED, EXCEPT AS OTHERWISE EXPRESSLY STATED IN THIS LICENSE AGREEMENT.

6. Limitation of Liability.

CORSAIR's entire liability for all claims or damages arising out of or related to this Agreement, regardless of the form of action, whether in contract, equity, negligence, intended conduct, tort or otherwise, will be limited to and will not exceed, in the aggregate for all claims, actions and causes of action of every kind and nature; the amount paid to CORSAIR for the specific item that caused the damage or that is the subject matter of the cause of action. In no event will the measure of damages payable by CORSAIR include, nor will CORSAIR be liable for, any amounts for loss of income, profit or savings or indirect, incidental, consequential, exemplary, punitive or special damages of any party, including third parties, even if CORSAIR has been advised of the possibility of such damages in advance, and all such damages are expressly disclaimed. This section shall not be interpreted to exclude any liability that is

prohibited from being excluded by applicable law. Except as otherwise provided by applicable law, no claim, regardless of form, arising out of or in connection with this Agreement may be brought by You unless such claim is brought either (i) within two years after the cause of action has accrued or (ii) within the shortest period of time after the cause of action has accrued that may be legally contracted for in the applicable jurisdiction if a two year limitation would be legally unenforceable.

7. Software Support Services.

CORSAIR offers technical support services. See www.corsair.com. Such technical support shall be provided in CORSAIR's sole discretion without any guarantee or warranty of any kind. It is your responsibility to back up of all your existing data, software and programs before receiving any technical support from CORSAIR. CORSAIR reserves the right to refuse, suspend or terminate any technical support, in its sole discretion.

8. Choice of Law and Jurisdiction.

This Agreement will be governed by and construed in accordance with the substantive laws of the State of California in the United States, without giving effect to any choice-of-law rules that may require the application of the laws of another jurisdiction. This Agreement will not be governed by the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded. Any disputes arising under this Agreement shall be settled exclusively in the California state courts or United States federal courts located in California. The parties hereby submit to the personal jurisdiction of such courts for the purpose of resolving any dispute under this Agreement.

9. Severability/Reformation.

If any provision of this Agreement is found to be void or unenforceable, it will not affect the validity of any other provision of this Agreement and those provisions will remain valid and enforceable according to their terms. To the extent that an unenforceable provision may be reformed to be enforceable by a court of law, such provision will be deemed to be so reformed in this Agreement.

10. Other Rights Reserved.

All rights not specifically granted in this Agreement are reserved by CORSAIR.

11. Entire Agreement.

You acknowledge that You have read this Agreement, understand it and agree to be bound by its terms and conditions. You further agree that it is the complete and exclusive statement of the agreement between us which supersedes any proposal or

prior agreement, oral or written, and any other communications between us relating to the subject matter of this Agreement.