



Intel® Converged Security Engine Firmware Integrated Clock Controller (ICC) Tool

Tools User Guide

October 2019

Revision 1.1

Intel Confidential



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.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at Intel.com, or from the OEM or retailer.

No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.

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.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

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

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

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

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

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

Copyright © 2019, Intel Corporation. All rights reserved.



Contents

1	Introduction	5
1.1	Terminology	5
1.2	Reference Documents	5
2	Intel® ICCS SDK	7
2.1	Intel® ICCS SDK.....	7
2.2	Intel® ICCS Control Library SDK Features	7
2.3	Intel® Integrated Clock Controller Service Data Structures	7
2.3.1	ICC_HECI_CLOCK_ID Type	8
2.3.2	ErrorCodes Type	8
2.3.3	ICC_GET_CLOCK_SETTINGSEx Type	8
2.3.4	ICC_SET_CLOCK_SETTINGSEx Type	9
2.4	Intel® Integrated Clock Controller Service API – Clock Manipulation Interface	10
2.4.1	Get Current Clock Settings Wrapper	10
2.4.2	Set Current Clock Settings Wrapper	11
3	Intel® Watchdog Timer Driver (Intel® WDT Driver).....	12
3.1	Communication with the Driver	12
3.2	Device Functionally - IOCTLs.....	12
3.2.1	Acquire Intel® Watchdog Timer Driver Control	13
3.2.2	Intel® Watchdog Timer Driver Control	13
3.2.3	Reload and Start Intel® Watchdog Timer Driver	14
3.2.4	Stop Intel Watchdog Timer Driver.....	15
3.2.5	Get Supported Countdown Data	15
3.2.6	Get Intel® Watchdog Timer Driver Status.....	16
3.2.7	Start Intel® Watchdog Timer Driver on Next OS Load	18
3.3	Driver Access.....	18
3.4	Usage Flows Example.....	19
3.4.1	Normal Flow	19
3.4.2	Start on Next Boot Flow	20



Revision History

Revision Number	Description	Revision Date
0.5	• Initial release.	November 2018
1.0	• Fixed typos	May 2019
1.1	• Removed CCT tool	October 2019

§ §



1 Introduction

The purpose of the document is to provide guidance on the usage of the tools provided for Intel® Converged Security Engine (Intel® CSE) Firmware Integrated Clock Controller (ICC) included within the Intel firmware kit.

This document covers the usage of the ICC SDK available in the **Tools\System_Tools\ICC Tools\ICC SDK** directory within the Intel® CSE FW kit.

1.1 Terminology

Acronym or Term	Definition
API	Application Programming Interface
BIOS	Basic Input Output System
CPU	Central Processing Unit
DLL	Dynamic Link Library
Intel® FIT	Intel® Flash Image Tool
FW	Firmware
Intel® ICCS	Intel® Integrated Clock Controller Services
Intel® CSE	Intel® Converged Security Engine
Intel® MEI	Intel® Management Engine Interface (formerly HECI)
PCH	Platform Controller Hub
Permanent UOB	UOB that is applied on every boot.
UOB	Update on Boot. A record of ICC registers setting that is applied on the next platform boot.

1.2 Reference Documents

Document	Document No./Location
Comet Lake Platform Controller Hub (CBL PCH) SPI Programming Guide	FW release kit



Document	Document No./Location
Comet Lake Platform Controller Hub (CML PCH) Intel® Management Engine Firmware Bring Up Guide	FW release kit
Comet Lake Platform Controller Hub (CML PCH) External Design Specification (EDS) Vol 1 & Vol 2	CDI#: Vol 1 → 574211 Vol 2 → 575200



2 Intel® ICCS SDK

2.1 Intel® ICCS SDK

Intel® Integrated Clock Controller Services (Intel® ICCS) provides a lot of flexibility for OS applications. To ease OS application development and to avoid erratic programming of ICC, Intel provides an ICC Control Library and abstracts ICC hardware from clock tuning applications such as BIOS.

2.2 Intel® ICCS Control Library SDK Features

ICC HW is only accessible to Intel® Management Engine (Intel® ME) and is accessible indirectly to the host software through a set of Intel® Management Engine Interface (Intel® MEI) APIs that is known to the Intel® Integrated Clock Controller Service. Intel does not expose this Intel MEI APIs and does not recommend OS applications to use them to keep platform stability.

Example of application that may call Intel® Integrated Clock Controller Service:

- Intel® Extreme Tuning Utility (Intel® XTU). This application can overclock or underclock platform BCLK (Processor clock).

2.3 Intel® Integrated Clock Controller Service Data Structures

Intel® Integrated Clock Controller Service provides a simplified ICC data structures and APIs for clock manipulation. The new data structure has significantly been reduced and simplified compared to previous generation of ICC control library. The ICC data structure is described in this section.

Table 2. Intel® Integrated Clock Controller Service API Data Structures

Name	Type	Description
ICC_HECI_CLOCK_ID	Enum	Defines the clock id for applicable clocks
ErrorCodes	UINT32	Returns error codes from Intel® Integrated Clock Controller Service API calls
ICC_GET_CLOCK_SETTI NGSEx	Struct	Contains the current clock setting
ICC_SET_CLOCK_SETTI NGSEx	Struct	Contains updatable clock setting



2.3.1 ICC_HECI_CLOCK_ID Type

The ICC_HECI_CLOCK_ID data structure provides the applicable clock to be selected with the following structure.

```
typedef enum
{
    ICC_HECI_PCIE_CLOCK_ID = 0,
    ICC_HECI_BCLK_CLOCK_ID = 1,
    ICC_HECI_WMPHY_CLOCK_ID = 2
}ICC_HECI_CLOCK_ID;
```

Table 3. Intel ICC_HECI_CLOCK_ID type

Name	Description
ICC_HECI_PCIE_CLOCK_ID	PCIe Clock (CPUBCLK Signal to CPU)
ICC_HECI_BCLK_CLOCK_ID	BCLK Clock (CPUBCLK Signal to CPU)
ICC_HECI_WMPHY_CLOCK_ID	White Mountain PLL

2.3.2 ErrorCodes Type

The ErrorCodes data structure provides description of the return error code from the Intel® Integrated Clock Controller Service.

The returned values are represented in UINT32 the following function is required to parse the values into char type:

```
const char* GetErrorStringByCode(const UINT32 errorCode);
```

2.3.3 ICC_GET_CLOCK_SETTINGSEx Type

The ICC_GET_CLOCK_SETTINGSEx structure provides all the clock settings details as the following:

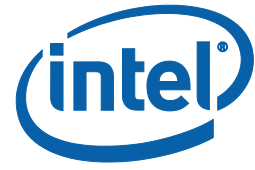
```
typedef struct _ICC_GET_CLOCK_SETTINGSEx
{
    UINT32 Frequency;
    UINT32 UserFrequency;
    UINT32 MaxFrequency;
    UINT32 MinFrequency;
    UINT8 SscMode;
    UINT8 SscPercent;
    UINT8 MaxSscPercent;
    UINT16 CurrentFlags;
    UINT16 SupportFlags;
} ICC_GET_CLOCK_SETTINGSEx;
```




2.3.4 ICC_SET_CLOCK_SETTINGSEx Type

The ICC_SET_CLOCK_SETTINGSEx structure provides all the updatable clock settings as the following:

```
typedef struct _ICC_SET_CLOCK_SETTINGSEx
{
    UINT32 UserFrequency;
    UINT8  SscPercent;    // encoding example: 1.28% -> SSC_SPREAD value is
128
    BOOL   SetToDefault;
    BOOL   ForcePowerFlow;
} ICC_SET_CLOCK_SETTINGSEx;
```



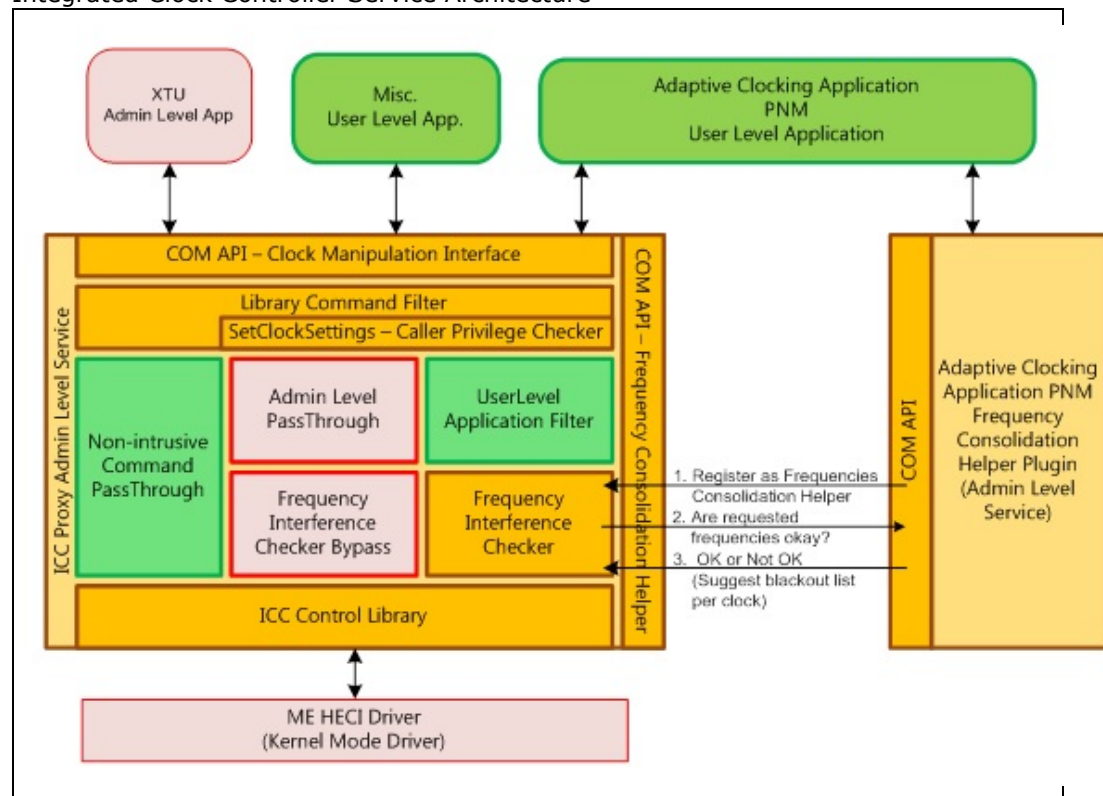
2.4 Intel® Integrated Clock Controller Service API – Clock Manipulation Interface

The Intel® Integrated Clock Controller Service provides a new set of simplified APIs in form of COM. The API exposed by the Intel® Integrated Clock Controller Service is used for communication between Intel® Integrated Clock Controller Service and client applications.

It is assumed that an application would only configure ICC clocks that “belong” to it. There is no provision in the library to lock a specific clock from a specific application.

Each API in the following section provides detail about Input and Output parameters for each API, as well as function prototype. For this guide, example in the following programming language will be provided: IDL and C++. Other programming languages are supported by the API, but an example will not be provided by this SDK user guide.

Note: Please note that Intel MEI Driver installation is required. Figure 1. Intel® Integrated Clock Controller Service Architecture



2.4.1 Get Current Clock Settings Wrapper

Through this function, the host application can get the runtime settings of the ICC clock identified by `ICC_HECI_PCIE_CLOCK_ID` parameter. The request returns the runtime settings of the clock.

**Definition:**

```
UINT32 IccLibGetCurrentClockSettingsWrapper(const ICC_HECI_CLOCK_ID
clockId, ICC_GET_CLOCK_SETTINGSEX * const clockSettings);
```

Table 4. Get Clock Runtime Settings Parameters

Type	Field	Description
Input	clockId	Clock identifier
Output	clockSettings	Runtime settings of the clock. Meaningful only if successful status is received.

2.4.2 Set Current Clock Settings Wrapper

Host application calls this function to change the settings of one of the ICC clocks.

For host application (with Admin level) call, the request will be verified against the ICC Clock Range Definition Record.

For host application (with user level) call, the request will be verified against the ICC enhanced SKU Clock Range definition.

Note: If the requested frequency is not supported by the HW, it will be automatically rounded by Intel® Integrated Clock Controller Service to the nearest valid frequency. If there are two nearest valid frequencies (up and down), the lower value will be chosen.

Definition:

```
UINT32 IccLibSetCurrentClockSettingsWrapper(const ICC_HECI_CLOCK_ID
clockId, ICC_SET_CLOCK_SETTINGSEX * clockSettings);
```

Table 5. Set Clock Runtime Settings Parameters

Type	Field	Description
Input	clockId	Clock identifier
Input	clockSettings	Runtime configuration for the clock.



3 Intel® Watchdog Timer Driver (Intel® WDT Driver)

Intel® Watchdog Timer Driver is a kernel driver (KMDF) that responsible for claiming the ACPI Intel WDT Device which is created by the BIOS and attaching to it. It is responsible for directly communicating with the hardware via I/O and should be aware of the hardware interface details necessary to communicate with the PCH. The Driver supports the following OS:

- Windows* 7 (32-bit and 64-bit Editions)
- Windows* 8 (32-bit and 64-bit Editions)
- Windows* 8.1 (32-bit and 64-bit Editions)
- Windows* Threshold (32-bit and 64-bit Editions)
- Windows* Threshold2 (32-bit and 64-bit Editions)
- Windows* 10 RS1(32-bit and 64-bit Editions)

3.1 Communication with the Driver

Communication with the Intel WDT driver is like any other kernel driver: When it's needed - use the standard Device/Files API (CreateFile, ReadFile, WriteFile, DeviceIoControl).

The device is identified by the following GUID (that is defined in ICCWDT_Interface.h):

```
// {C2E625A9-8693-4dea-BAC4-B15CA98F9EE9}  
  
DEFINE_GUID(GUID_DEVINTERFACE_ICCWDT,  
0xc2e625a9, 0x8693, 0x4dea, 0xba, 0xc4, 0xb1, 0x5c, 0xa9, 0x8f, 0x9e, 0xe9);
```

Connection example code can be found in Microsoft* Windows* Driver Kit
<http://www.microsoft.com/whdc/devtools/wdk/default.aspx>.

3.2 Device Functionally - IOCTLs

The following list describes the device IOCTLs:

Table 6. Device Functionality - IOCTLs

IOCTL	Description
ICCWDT_AQUIRE_WDT	Acquire Intel Watchdog Timer Driver Control
ICCWDT_RELEASE_WDT	Release Intel Watchdog Timer Driver Control



IOCTL	Description
ICCWDT_RELOADANDSTART_WDT	Reload and Start Intel Watchdog Timer Driver
ICCWDT_STOP_WDT	Stop Intel Watchdog Timer Driver
ICCWDT_GET_SUPPORTED_CD_DATA	Get Supported Countdown Data
ICCWDT_GET_WDT_STATUS	Get Intel Watchdog Timer Driver Status
ICCWDT_START_WDT_ON_NEXT_OSBOOT	Start Intel Watchdog Timer Driver on Next OS Load

3.2.1 Acquire Intel® Watchdog Timer Driver Control

The Acquire Intel Watchdog Timer Driver (Intel® WDT Driver) Control method allows a single application to be able to gain control of the Intel WDT Driver. Due to the timing sensitive nature of the Intel WDT Driver, associated timeouts and reloads, only a single application can be allowed to control the hardware at a time.

Table 7. Acquire Intel® Watchdog Timer Driver Control

IOCTL Name	ICCWDT_AQUIRE_WDT
IOCTL Definition	CTL_CODE(FILE_DEVICE_ACPI, 0x800, METHOD_BUFFERED, FILE_READ_ACCESS FILE_WRITE_ACCESS)
Input	None
Output	ULONG Key – Watchdog Acquiring key
Error Code	SUCCESS - No Error – Successful
	ACCESS_DENIED_ERROR – Intel Watchdog Timer Driver already been acquired
	HARDWARE_LOCKED_ERROR – Intel Watchdog Timer Driver hardware registers are lock, can't be modified.
	FATAL_ERROR - Unexpected error – Fatal Error.

3.2.2 Intel® Watchdog Timer Driver Control

The Release Intel Watchdog Timer Driver Control IOCTL is used to release control of the Intel WDT Driver from the application that currently controls it. This communicates to the driver that the current application is done with its needs for the Intel WDT Driver and that it can be allocated to another application if necessary.

**Table 8. Release Intel® Watchdog Timer Driver Control**

IOCTL Name	ICCWDT_RELEASE_WDT
IOCTL Definition	CTL_CODE(FILE_DEVICE_ACPI, 0x802, METHOD_BUFFERED, FILE_READ_ACCESS FILE_WRITE_ACCESS)
Input	ULONG Key – Intel Watchdog Timer Driver Acquiring key, retrieved from ICCWDT_AQUIRE_WDT
Output	None
Error Code	SUCCESS - No Error – Successful
	ACCESS_DENIED_ERROR – Intel Watchdog Timer Driver already been released
	FATAL_ERROR - Unexpected error – Fatal Error.

3.2.3 Reload and Start Intel® Watchdog Timer Driver

The Reload and Start Intel Watchdog Timer Driver IOCTL is responsible for loading the appropriate countdown timer into the Intel WDT Driver, reloading the countdown timer, and enabling the Intel WDT Driver countdown. When the Intel WDT Driver counter reaches 0 then the Intel WDT Driver will signal a Global Reset. If this happens it is assumed that the platform is no longer responsive, and the reset action is required to return the platform to a usable state. The countdown value can vary between 1 sec and 1024 sec (~17Min).

Table 9. Reload and Start Intel® Watchdog Timer Driver

IOCTL Name	ICCWDT_RELOADANDSTART_WDT
IOCTL Definition	CTL_CODE(FILE_DEVICE_ACPI, 0x803, METHOD_BUFFERED, FILE_READ_ACCESS FILE_WRITE_ACCESS)
Input	<pre>typedef struct _ICCWDT_RELOADANDSTART_DATA{ ULONG Key; UINT16 CountdownVal; }ICCWDT_RELOADANDSTART_DATA, *PICCWDT_RELOADANDSTART_DATA;</pre> <p>ULONG Key – Intel Watchdog Timer Driver Acquiring key, retrieved from ICCWDT_AQUIRE_WDT</p> <p>UINT16 CountdownVal – Intel Watchdog Timer Driver Countdown Value. Can be set between 1d and 1024d</p>
Output	None
Error Code	SUCCESS - No Error – Successful
	ACCESS_DENIED_ERROR - Intel Watchdog Timer Driver was not been acquired or wrong acquiring key.
	INVALID_PARAMETER_ERROR - Bad Countdown value.
	FATAL_ERROR - Unexpected error – Fatal Error.



3.2.4 Stop Intel Watchdog Timer Driver

The Stop Intel® Watchdog Timer Driver IOCTL is responsible for stopping the Intel WDT Driver countdown. When the Intel WDT Driver counter has been stopped it will never signal a Global Reset. This method does not infer that a reload of the countdown timer has occurred. It merely disabled the Global Reset output. To restart the Intel WDT Driver, the user can just call the Reload and Start Intel Watchdog Timer Driver IOCTL.

Table 10. Stop Intel® Watchdog Timer Driver

IOCTL Name	ICCWDT_STOP_WDT
IOCTL Definition	CTL_CODE(FILE_DEVICE_ACPI, 0x804, METHOD_BUFFERED, FILE_READ_ACCESS FILE_WRITE_ACCESS)
Input	ULONG Key – Intel Watchdog Timer Driver Acquiring key, retrieved from ICCWDT_AQUIRE_WDT
Output	None
Error Code	SUCCESS - No Error – Successful
	ACCESS_DENIED_ERROR – Intel Watchdog Timer Driver was not being acquired or wrong acquiring key.
	FATAL_ERROR - Unexpected error – Fatal Error.

3.2.5 Get Supported Countdown Data

The Get Supported Countdown Data IOCTL is responsible for providing the calling application with the information necessary to understand what timeout settings are available for the Intel WDT Driver. This IOCTL can be called at any point while the caller has control over the Intel WDT Driver.



Table 11. Get Supported Countdown Data

IOCTL Name	ICCWDT_GET_SUPPORTED_CD_DATA
IOCTL Definition	CTL_CODE(FILE_DEVICE_ACPI, 0x805, METHOD_BUFFERED, FILE_READ_ACCESS FILE_WRITE_ACCESS)
Input	ULONG Key – Intel Watchdog Timer Driver Acquiring key, retrieved from ICCWDT_AQUIRE_WDT
Output	<pre>typedef struct _ICCWDT_SUPPORTED_CD_DATA{ UINT16 MinimumTimeoutPeriod; UINT16 MaximumTimeoutPeriod; UINT16 TimeoutResolution; }ICCWDT_SUPPORTED_CD_DATA, *PICCWDT_SUPPORTED_CD_DATA;</pre> <p>UINT16 MinimumTimeoutPeriod - Minimum Intel Watchdog Timer Driver Timeout Period UINT16 MaximumTimeoutPeriod - Maximum Intel Watchdog Timer Driver Timeout Period UINT16 TimeoutResolution – Intel Watchdog Timer Driver Timeout Resolution</p> <p>Note: all values are in seconds.</p>
Error Code	SUCCESS - No Error – Successful
	ACCESS_DENIED_ERROR – Intel Watchdog Timer Driver was not acquired or wrong acquiring key.
	FATAL_ERROR - Unexpected error – Fatal Error.

3.2.6 Get Intel® Watchdog Timer Driver Status

This IOCTL is used to report the status of the previous boot to the application calling the Intel WDT Driver. The intent is to let the controlling application know whether the previous boot had resulted in a failed POST. This includes either an Intel WDT Driver Timeout or an unexpected reboot while the Intel WDT Driver was running. Either situation will result in a timeout and communication of that failure through this method.

Table 12. Get Intel® Watchdog Timer Driver Status

IOCTL Name	ICCWDT_GET_WDT_STATUS
-------------------	-----------------------



IOCTL Definition	CTL_CODE(FILE_DEVICE_ACPI, 0x806, METHOD_BUFFERED, FILE_READ_ACCESS FILE_WRITE_ACCESS)												
Input	ULONG Key – Watchdog Acquiring key, retrieved from ICCWDT_AQUIRE_WDT												
Output	<pre>typedef enum _ICCWDT_TIMER_STATUS_TYPE { FAIL, PASS }ICCWDT_TIMER_STATUS_TYPE;</pre> <pre>typedef enum _ICCWDT_TIMER_STATE_TYPE { RUNNING, STOPPED }ICCWDT_TIMER_STATE_TYPE;</pre> <pre>typedef struct _ICCWDT_GET_WDT_STATUS_DATA{ ICCWDT_TIMER_STATUS_ENUM WDTTimerStatus; ICCWDT_TIMER_STATE_TYPE WDTTimerState; UINT16 WDTCountdownPeriod; }ICCWDT_GET_WDT_STATUS_DATA, *PICCWDT_GET_WDT_STATUS_DATA;</pre> <p>ICCWDT_TIMER_STATUS_TYPE WDTTimerStatus - Watchdog Timer Status ICCWDT_TIMER_STATE_TYPE WDTTimerState - Watchdog Timer State UINT16 WDTCountdownPeriod - Watchdog Timer Countdown Period</p> <table border="1"> <thead> <tr> <th>ICC WDT STATUS</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Fail</td><td> <ul style="list-style-type: none"> Previous boot occurred during Intel WDT Driver was running Intel WDT Driver expired </td></tr> <tr> <td>Pass</td><td> <ul style="list-style-type: none"> Previous boot occurred when Intel WDT Driver was stopped </td></tr> <tr> <th>ICC WDT STATE</th><th>Description</th></tr> <tr> <td>Running</td><td> <ul style="list-style-type: none"> Intel WDT Driver is running </td></tr> <tr> <td>Stopped</td><td> <ul style="list-style-type: none"> Intel WDT Driver is stopped </td></tr> </tbody> </table>	ICC WDT STATUS	Description	Fail	<ul style="list-style-type: none"> Previous boot occurred during Intel WDT Driver was running Intel WDT Driver expired 	Pass	<ul style="list-style-type: none"> Previous boot occurred when Intel WDT Driver was stopped 	ICC WDT STATE	Description	Running	<ul style="list-style-type: none"> Intel WDT Driver is running 	Stopped	<ul style="list-style-type: none"> Intel WDT Driver is stopped
ICC WDT STATUS	Description												
Fail	<ul style="list-style-type: none"> Previous boot occurred during Intel WDT Driver was running Intel WDT Driver expired 												
Pass	<ul style="list-style-type: none"> Previous boot occurred when Intel WDT Driver was stopped 												
ICC WDT STATE	Description												
Running	<ul style="list-style-type: none"> Intel WDT Driver is running 												
Stopped	<ul style="list-style-type: none"> Intel WDT Driver is stopped 												
Error Code	<p>SUCCESS - No Error – Successful</p> <p>ACCESS_DENIED_ERROR - Watchdog Timer was not acquired or wrong acquiring key.</p> <p>FATAL_ERROR - Unexpected error – Fatal Error.</p>												



3.2.7 Start Intel® Watchdog Timer Driver on Next OS Load

This IOCTL is used to communicate to the BIOS that on the next boot attempt, the Intel WDT Driver should be turned on after the POST process has been completed. This allows for instability coverage after POST has completed, yet before drivers can be loaded by the OS. It is a mechanism that allows for automatically recovering from system instability issues while applying a setting that requires a reboot. The given value is between 0 and 1008. 0 is for disabling Starting Intel Watchdog Timer Driver on Next OS Load. The values will be rounded to 16sec intervals (e.g. 1-16 will be set to 16 secs; 17-32 will be set to 32sec etc....)

It is required that any application which requests the Intel WDT Driver be started during the next OS load must automatically load during the next OS boot and call the Check Intel Watchdog Timer Driver Status IOCTL and handle the result appropriately.

Table 13. Start Intel® Watchdog Timer Driver on Next OS Load

IOCTL Name	ICCWDT_START_WDT_ON_NEXT_OSBOOT
IOCTL Definition	<u>CTL_CODE</u> (FILE_DEVICE_ACPI, 0x807, METHOD_BUFFERED, FILE_READ_ACCESS FILE_WRITE_ACCESS)
Input	<pre>typedef struct _ICCWDT_START_WDT_ON_NEXT_OSBOOT_DATA{ ULONG Key; UINT16 TimeoutValue; }ICCWDT_START_WDT_ON_NEXT_OSBOOT_DATA, PICCWD T_START_WDT_ON_NEXT_OSBOOT_DATA; ULONG Key - Watchdog Acquiring key, retrieved from ICCWDT_AQUIRE_WDT UINT16 TimeoutValue - Watchdog Timer Timeout Value</pre>
Output	None
Error Code	SUCCESS - No Error – Successful
	ACCESS_DENIED_ERROR – Watchdog Timer was not acquired or wrong acquiring key.
	INVALID_PARAMETER_ERROR - Invalid Countdown Value.
	COUNTDOWN_ERROR - Enable to Start Intel WDT Driver ON Next Load.
	FATAL_ERROR - Unexpected error – Fatal Error.

3.3 Driver Access

By default, the access to the driver is limited for “Local System” accounts; this means that only Services will be able to access the driver. For development purposes there is a version of the driver called “Debug/Development Mode” this driver has all user access to the driver after installation.

Note: Intel Watchdog Timer Driver Debug Driver will reload and start the timer every 0.5 seconds after the first ReloadAndStart command from the User.



Note: Development' mode driver should only be used in development, and not in production.

3.4 Usage Flows Example

Examples below use Pseudo Code.

3.4.1 Normal Flow

```
#include <windows.h>
#include "ICCWDT_Interface.h"
...
ULONG key = 0;

void MyIccWdtThread()
{
    // Boilerplate code from MSFT to getting the Device Path from Device Interface.
    string devicePath = GetDevicePath(GUID_DEVINTERFACE_ICCWDT);

    HANDLE deviceHandle = CreateFile(devicePath, ...);

    // Acquire Timer
    DeviceControl(deviceHandle, ICCWDT_AQUIRE_WDT, NULL, key);

    while(!MyApp_Stop)
    {
        ICCWDT_RELOADANDSTART_DATA data;
        data.key = key;
        data.CountdownVal = MY_WDT_VALUE;

        // Start the Timer
        DeviceControl(deviceHandle, ICCWDT_RELOADANDSTART_WDT, data, NULL);

        // Sleep for Time < ICC WDT Timer Value (MY_WDT_VALUE)
        Sleep(MY_WDT_SLEEP)
    }

    // Stop Timer
    DeviceControl(deviceHandle, ICCWDT_STOP_WDT, key, NULL);

    // Release Timer
    DeviceControl(deviceHandle, ICCWDT_RELEASE_WDT, key, NULL);

    CloseHandle(deviceHandle);
}
```



3.4.2 Start on Next Boot Flow

```
#include <windows.h>
#include "ICCWDT_Interface.h"
...

ULONG key = 0;

void StartOnNextBoot()
{
    // Boilerplate code from MSFT to getting the Device Path from Device Interface.
    string devicePath = GetDevicePath(GUID_DEVINTERFACE_ICCWDT);

    HANDLE deviceHandle = CreateFile(devicePath, ...);

    ULONG key;

    // Acquire Timer (if not already acquired)
    DeviceControl(deviceHandle, ICCWDT_AQUIRE_WDT, NULL, key);

    ICCWDT_START_WDT_ON_NEXT_OSBOOT_DATA data;
    data.key = key;
    data.TimeoutValue = MY_WDT_VALUE;

    // Start the Timer
    DeviceControl(deviceHandle, ICCWDT_START_WDT_ON_NEXT_OSBOOT, data, NULL);

    // Release Timer (if not releasing in another place)
    DeviceControl(deviceHandle, ICCWDT_RELEASE_WDT, key, NULL);

    CloseHandle(deviceHandle);

    // Reboot System or prompt the user to reboot.
    RebootSystem();
}
```