Intel® Extreme Tuning Utility Software Development Kit (SDK)

Programmer’s Guide

September 2018

Revision 0.91

Intel Confidential

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document 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.

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

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by going to: http://www.intel.com/design/literature.htm.

Code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Chief River and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.

Intel® Centrino®, Intel® Smart Response Technology, Intel® Rapid Start Technology, Intel® Core™, Intel® Management Engine (Intel® ME), Intel® Smart Connect Technology, 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 © 2014, 2016, Intel Corporation. All rights reserved.

## Contents

[Contents 3](#_Toc1047021)

[Revision History 7](#_Toc1047022)

[Terminology 9](#_Toc1047023)

[1 Introduction 10](#_Toc1047024)

[1.1 Purpose of this Document 10](#_Toc1047025)

[1.2 XTU Overview 10](#_Toc1047026)

[2 XTU SDK API 12](#_Toc1047027)

[2.1 Overview 12](#_Toc1047028)

[2.2 Interface IIntelOverclockingLibrary 12](#_Toc1047029)

[2.2.1 API Definition 12](#_Toc1047030)

[2.2.2 Operation Theory 13](#_Toc1047031)

[2.3 Interface ITuningLibrary 16](#_Toc1047032)

[2.3.1 API Definition 16](#_Toc1047033)

[2.3.2 Tuning ID Definition 24](#_Toc1047034)

[2.3.3 Operation Theory 27](#_Toc1047035)

[2.4 Interface IMonitoringLibrary 29](#_Toc1047036)

[2.4.1 API Definitions 29](#_Toc1047037)

[2.4.2 Monitor ID Definitions 30](#_Toc1047038)

[2.4.3 Operation Therory 34](#_Toc1047039)

[2.5 Interface IServiceInfoLibrary 35](#_Toc1047040)

[2.5.1 API Definitions 35](#_Toc1047041)

[2.5.2 Operation Theory 40](#_Toc1047042)

[2.6 Interface IWatchdogTimerInfo 41](#_Toc1047043)

[2.6.1 API Definitions 41](#_Toc1047044)

[2.7 Interface IProcessorInfo 42](#_Toc1047045)

[2.7.1 API Definitions 42](#_Toc1047046)

[2.8 Interface IOverclockingServiceInfo 43](#_Toc1047047)

[2.8.1 API Definitions 43](#_Toc1047048)

[2.9 Interface IProfileLibrary 44](#_Toc1047049)

[2.9.1 Data Structures 44](#_Toc1047050)

[2.9.2 Enumerations 50](#_Toc1047051)

[2.9.3 API Definitions 51](#_Toc1047052)

[2.9.4 Operation Theory 54](#_Toc1047053)

[2.10 Interface IAppProfileLibrary 56](#_Toc1047054)

[2.10.1 API Definitions 56](#_Toc1047055)

[2.11 Interface IProcessLibrary 59](#_Toc1047056)

[2.11.1 API Definitions 59](#_Toc1047057)

[2.12 Interface IBenchmarkLibrary 60](#_Toc1047058)

[2.12.1 API Definitions 60](#_Toc1047059)

[2.12.2 Operation Theory 62](#_Toc1047060)

[2.13 Interface IEventBroadcastLibrary 66](#_Toc1047061)

[2.13.1 API Definition 66](#_Toc1047062)

[3 XTU SDK Installation 68](#_Toc1047063)

[4 XTU SDK Integration 70](#_Toc1047064)

[4.1 Integration With a .NET Application 70](#_Toc1047065)

[4.2 Integration With a Native Application 73](#_Toc1047066)

[4.2.1 SDK Wrapper DLL for Native Application 73](#_Toc1047067)

[4.2.2 Native Application Development 79](#_Toc1047068)

[4.2.3 Debugging SdkWrapperForNativeCode.DLL 83](#_Toc1047069)

[5 OC Application Deployment 85](#_Toc1047070)

[5.1 Merge Modules 85](#_Toc1047071)

[5.2 Runtime Libraries 85](#_Toc1047072)

[5.3 Benchmark Executables 86](#_Toc1047073)

[5.4 Coexistence with other overclocking applications 86](#_Toc1047074)

[5.5 Overclocking Warning 87](#_Toc1047075)

[5.6 64bit Support 87](#_Toc1047076)

[6 XTU SDK Debugging 89](#_Toc1047077)

[6.1 Local Debug 89](#_Toc1047078)

[6.2 Remote Debug 89](#_Toc1047079)

[6.3 SDK Log 89](#_Toc1047080)

[7 XTU SDK Sample Code 90](#_Toc1047081)

[7.1 Sample Application Package 90](#_Toc1047082)

[7.2 XtuCLI Demo Application 90](#_Toc1047083)

[7.3 NativeConsoleApp Application 93](#_Toc1047084)

[8 XTU SDK Redistributables Customization Guide 94](#_Toc1047085)

[8.1 Merge Module Installer 94](#_Toc1047086)

[8.2 Install XTU Service 94](#_Toc1047087)

[8.3 Device Driver Requirement 95](#_Toc1047088)

[8.4 Install XTU Drivers 95](#_Toc1047089)

[8.4.1 Folder AcpiDriver 95](#_Toc1047090)

[8.4.2 Folder IccWdtDriver 96](#_Toc1047091)

[8.4.3 Folder IocDriver 96](#_Toc1047092)

[8.4.4 Folder IocExtensionDriver 96](#_Toc1047093)

[8.4.5 Folder IocComponentDriver 96](#_Toc1047094)

Figures

Figure 1‑1 XTU SDK Software Components 11

Figure 1‑2 Add Reference to Intel XTU SDK 70

Figure 1‑3 Browse to Intel XTU SDK Assembly 72

Figure 1‑4 IntelOverclockingSdk Reference Added to the Solution 73

Figure 1‑5 Create a SDK Wrapper DLL for Native Application 73

Figure 1‑6 SDK Wrapper DLL Project Property 74

Figure 1‑7 Add Reference to IntelOverclockingSDK DLL 75

Figure 1‑8 Reference is added to SDK Wrapper Project 76

Figure 1‑9 SDK Wrapper For Native Application Project 77

Figure 1‑10 Global Handle to IntelOverclockingSDK 78

Figure 1‑11 Add Include Path to Native Application 80

Figure 1‑12 Add Additional Dependencies to Native Application 81

Figure 1‑13 NativeConsoleApp Sample Code 82

Figure 1‑14 SdkWrapperForNativeCode.dll is listed as an External Dependency 83

Figure 1‑15 Configure Debugging Command for SdkWrapperForNativeCode.DLL to NativeConsoleApp.exe 84

Figure 1‑16 64bit Configuration for Sample Applications 88

Figure 4‑1 XTU SDK Demo Application UI 91

Tables

Table 2‑1. Intel® Smart Connect Technology Agent Registry Configuration Settings 12

## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Document Number | Revision Number | Description | Revision Date |
|  | 0.5 | * Initial release | December 2014 |
|  | 0.6 | * Added ProfileLibrary, ProcessLibrary, AppProfileLibrary, BenchmarkControl, CompareOnlineControl, and associated data structures and APIs. * Changed the library initialization process. Most of libraries are implemented as singleton classes. The only way to get a reference to their instances is to go through IntelOverclockingLibrary class. | April 2015 |
|  | 0.7 | * Added two more APIs in TuningLibrary: SetTuningDelay and SetSuspendRestoreOptions * Added procedures for native application integration. * Removed some tuning controls that no longer applies. | June 11, 2015 |
|  | 0.71 | * Update for redistribution and debugging for SDK * 64bit native application support |  |
|  | 0.72 | * Visual Studio 2010 C++ Redistributable is required for benchmark to run. | July 30, 2015 |
|  | 0.73 | * Added a section for redistritables customization | August 27, 2015 |
|  | 0.74 | * Modified section 1.3.1 to add IntelBenchmarkSDK.dll and CommonDriverEssentials.msm. * Modified section 1.3.2 to provide more guidance about the SDK integration. * Removed section 1.4 * Added GetControl() API in section 2.3.1 |  |
|  | 0.75 | * Add SetSuspendRestoreOptions() | October 27, 2015 |
|  | 0.76 | * Re-arranged the document layout. * Add Microsoft SQL Compact Edition runtime support in Capture 3 and 5. * Renamed some of chapter names. | December 1, 2015 |
|  | 0.77 | * Update the copyright to 2016 | March 10, 2016 |
|  | 0.78 | * Update Benchmark SDK. Added Benchmark APIs and Removed Benchmark User Controls. * Also updated Tuning and Monitor Control IDs for BDX and KBL. | April 19, 2016 |
|  | 0.79 | * Updated information about SQL Server CE version requirements | August 16, 2018 |
|  | 0.90 | * Updated information about driver installation for DCHU compliance | September 10, 2018 |
|  | 0.91 | * Remove Microsoft SQL Compact Edition requirements | February 14, 2019 |

Reference Documents

|  |  |
| --- | --- |
| Document | Document Number |
| *Xxxx Performance Tuning Guide.pdf* |  |
|  |  |
|  |  |

## Terminology

|  |  |
| --- | --- |
| Term | Description |
| XTU | Intel® Extreme Tuning Utility designed for performance tuning, verification, and monitoring. |
| SDK | Software Development Kit designed to enable OEM/ODM overclocking applications |
| HWBOT | HWBOT.org is an overclocking competition organization. The the HWBOT overclocking record submission process is integrated with XTU SDK. |
| OC | Overclocking to boost system performance |
| Merge Module | An installation library that is responsible for a module installation. It has to be integrated into an installer in order to execute the merge module. |
| PTG | Performance Tuning Guide. This is the overclocking whitepaper for a specific platform. |
| API | Application Programming Interface |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

§

# Introduction

## Purpose of this Document

This document describes Intel® Extreme Tuning Utility software development kit (SDK) and how the SDK can be used by a third party software.

This document does not cover either the performance tuning mechanism nor the overclocking best practice. Please refer to the relevant platform Performance Tuning Guide (PTG) for performance tuning methodologies.

The audience of this document include the performance tuning software application developer, the validation engineer, and the application enabling engineer.

## XTU Overview

Intel® Extreme Tuning Utility (XTU) is a system performance tuning tool provided by Intel. The focus of this tool is to provide runtime performance tuning capability to the end user. However, with BIOS support, it can be used to support BOOT time performance tuning as well. This tool has been available for many years and it can be run on many generations of Intel® CoreTM processors. By default, the tool supports backward compatibility up to 3 generations prior to the latest Intel® CoreTM processor that is publically available. The tool can be downloaded from [Intel Download Center](https://downloadcenter.intel.com/Detail_Desc.aspx?agr=Y&DwnldID=24075&lang=eng&wapkw=intel+extreme+tuning+utility+%28intel+xtu%29).

In general prior to XTU 6.0, XTU consists of the following major components:

* XTU GUI application
* XTU Core Service
* XTU Common Drivers

The XTU GUI application presents the information about the platform to the user and also provides options for the user to tune, stress, and benchmark the system performance. For security reason, the administrator privilege is required for the tuning operation. The XTU Core Service provides the support to the GUI application. It is running as a Windows service with the local system privilege. The XTU Common Drivers layer provides the software and hardware interfaces. The XTU Core Service relies on those drivers to make changes to the hardware.

Starting from XTU 6.0, an XTU SDK will be provided to external software providers including OEMs/ODMs. The Figure 1‑1 shows the relation of SDK to other software components. The SDK provides an abstraction layer between the XTU Core Service and the end user application.

The Intel Extreme Tuning Utility SDK provides a mechanism for a third party (ISV) application to detect system tuning capability, monitor system performance and thermal conditions, or even tune the platform performance on an IA-CoreTM based platform. As indicated in the previous paragraph, it relies on the Intel XTU core service and common device drivers to be running.

The XTU SDK is developed based on .NET framework. However, it can be easily extended to support the native application (refer to sample code for more information). The XTU SDK can be integrated into a Windows service or an user interactive application. Multiple clients can be supported by the SDK. The communication from the multiple clients and the hardware is serialized, however, please note that changes from any client could affect the hardware. If multiple clients are trying to change the system performance, the last change request wins.



Figure 1‑1 XTU SDK Software Components

Techinically there are two SDKs in XTU SDK package:

* Overclocking SDK: this SDK is basis for all other overclocking activities. Its primary focus is to provide APIs for 3rd party application to tune and monitor the system. It also provides APIs to manage the XTU overclocking profiles and application-to-profile pairing service.
* Benchmark SDK: this SDK is a value add-on for application developers to run XTU benchmark and HWBOT online competition submission. It provides two user controls that can be easily dropped to any C# UI application.

# XTU SDK API

## Overview

There are several interfaces exposed by the SDK. All interfaces are accessible within a namespace (Table 2‑1). Interfaces are implemented in hierarchy. The interface in the higher hierarchy contains the interfaces below. For example, the interface IIntelOverclockingLibrary contains 3 interfaces: ITuningLibrary, IMonitorLibrary, and IServiceInfoLibrary. The interface IServiceInfoLibrary contains 3 interfaces: IWatchdogTimerInfo, IProcessorInfo, and IOverclockingServiceInfo.

|  |  |  |
| --- | --- | --- |
| Namespace | Interface Hierarchy | Description |
| Intel.Overclocking.SDK | * IIntelOverclockingLibrary | Top level interface |
| Intel.Overclocking.SDK.Tuning | * ITuningLibrary | Provide tuning APIs |
| Intel.Overclocking.SDK.Monitoring | * IMonitoringLibrary | Provide monitoring APIs. |
| Intel.Overclocking.SDK.ServiceInfo | * IServiceInfoLibrary   + IWatchdogTimerInfo   + IProcessorInfo   + IOverclockingServiceInfo | Provide system and service information including SDK version. |
| Intel.Overclocking.SDK.Profile | * IProfileLibrary * IAppProfileLibrary * IBenchmarkLibrary | Provide APIs for profile management as well as benchmark and HWBOT submission. |
| Intel.Overclocking.SDK.ProcessLauncher | * IProcessLibrary | Provide APIs for user process activation service. |
| Intel.Overclocking.SDK.Event | * IEventBroadcastLibrary | Provide APIs for event notification registration |

Table 2‑1 SDK Interfaces and Namespaces

**Please note there are more APIs and data structures available through the interfaces than what we have published here. The APIs not published here are treated as undocumented APIs. Intel will not guarantee that those APIs and data structures will be supported in the current format in the future release. Intel reserves all rights to make any changes to those undocumented APIs.**

**If you think some undocumented APIs are very useful for your application, please contact the local Intel AE to make a request.**

## Interface IIntelOverclockingLibrary

### API Definition

namespace Intel.Overclocking.SDK

{

using Intel.Overclocking.SDK.Monitoring;

using Intel.Overclocking.SDK.ProcessLauncher;

using Intel.Overclocking.SDK.ServiceInfo;

using Intel.Overclocking.SDK.Tuning;

using Intel.Overclocking.SDK.Profile;

public interface IIntelOverclockingLibrary

{

/// <summary>

/// Returns the object for ServiceInfo library

/// </summary>

IServiceInfoLibrary SystemInfoLib { get; }

/// <summary>

/// Returns the object for Monitoring Library

/// </summary>

IMonitoringLibrary MonitoringLib { get; }

/// <summary>

/// Returns the object for Tuning Library

/// </summary>

ITuningLibrary TuningLib { get; }

/// <summary>

/// Returns the object for Profile Library

/// </summary>

IProfileLibrary ProfileLib { get; }

/// <summary>

/// Returns the object for Process Library

/// </summary>

IProcessLibrary ProcessLib { get; }

/// <summary>

/// Initialize the ServiceInfo, Monitoring, and Tuning connections.

/// </summary>

void Initialize();

}

}

### Operation Theory

namespace OverclockingSdkSampleApp

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using Intel.Overclocking.SDK;

class Program

{

static void Main(string[] args)

{

List<ClientTuningControl> theTuningControlList = new

List<ClientTuningControl>();

IIntelOverclockingLibrary theOcLib = new IntelOverclockingLibrary();

// Initialize the library

theOcLib.Initialize();

System.Console.WriteLine("Intel(R) Overclocking SDK version " +

theOcLib.SystemInfoLib.ServiceInfo.GetOverclockingSdkVersion());

System.Console.WriteLine("Intel(R) Overclocking Service version " +

theOcLib.SystemInfoLib.ServiceInfo.GetOverclockingServiceVersion());

System.Console.WriteLine("Target CPU: " +

theOcLib.SystemInfoLib.Processor.GetBrandString());

System.Console.WriteLine("CPU Core Count: " +

theOcLib.SystemInfoLib.Processor.GetPhysicalCpuCoreCount());

System.Console.WriteLine("CPU uCode Version: " +

theOcLib.SystemInfoLib.Processor.GetMicrocodeUpdateVersion());

if (!theOcLib.SystemInfoLib.Processor.IsSystemUnlocked())

{

System.Console.WriteLine("System has locked part - no overclocking is

allowed!");

return;

}

if (!theOcLib.SystemInfoLib.Processor.IsOverclockSupported())

{

System.Console.WriteLine("System does not support overclocking!");

return;

}

if (!theOcLib.SystemInfoLib.Processor.IsTurboBoostTechnologyEnabled())

{

System.Console.WriteLine("Intel(R) Turbo Boost Technology is required for

overclocking!");

return;

}

if (theOcLib.SystemInfoLib.WatchdogTimer.IsWatchdogTimerRunning())

{

System.Console.WriteLine("Overclocking Watchdog Timer is running!");

}

// Monitoring

List<ClientMonitor> theMonitorList = new List<ClientMonitor>();

theMonitorList = theOcLib.MonitoringLib.GetAvailableMonitors(false);

if (theMonitorList.Count > 0)

{

System.Console.WriteLine("There are total of " + theMonitorList.Count +

"monitors available.");

}

if (theOcLib.MonitoringLib.MonitorIsReady(5))

{

System.Console.WriteLine("The processor frequency is " +

theOcLib.MonitoringLib.GetValue(5) + "MHz");

}

// Tuning

theTuningControlList = theOcLib.TuningLib.GetAvailableControls();

if (theTuningControlList.Count > 0)

{

System.Console.WriteLine("There are total of " +

theTuningControlList.Count + "tuning knobs available.");

}

List<ClientTuningProposal> theTuningProposalList = new

List<ClientTuningProposal>();

theTuningProposalList.Add(new ClientTuningProposal() { Id = 0x1D, Value = 20

}); // set 1-Active Core Ratio limit to 2GHz

theTuningProposalList.Add(new ClientTuningProposal() { Id = 0x1E, Value = 19

}); // set 2-Active Core Ratio limit to 1.9GHz

List<ClientTuningProposalResult> theTuningProposalResultList = new

List<ClientTuningProposalResult>();

bool bRestartRequired = false;

if (!theOcLib.TuningLib.ProposeChange(theTuningProposalList, out

theTuningProposalResultList, out bRestartRequired))

{

System.Console.WriteLine("Unable to propose the change to Processor Core

Ratio Limits");

return;

}

// The watchdog timer will be started before the changes are made to the

// hardware.

if (!theOcLib.TuningLib.ApplyChanges(bRestartRequired))

{

System.Console.WriteLine("Unable to apply changes to the hardware!");

}

if (bRestartRequired)

{

System.Console.WriteLine("System will reboot in 10 seconds");

Thread.Sleep(10 \* 1000);

theOcLib.TuningLib.Restart();

}

}

}

}

## Interface ITuningLibrary

### API Definition

namespace Intel.Overclocking.SDK.Tuning

{

/// <summary>

/// This class defines all properties associated with a tuning contorl. A tuning

/// control can be identified by the unique ID or the name.

/// </summary>

public class ClientTuningControl

{

public uint Id { get; set; }

public decimal DefaultValue { get; set; }

public decimal ActiveValue { get; set; }

public decimal ProposedValue { get; set; }

public decimal BootValue { get; set; }

public bool RequiresReboot { get; set; }

public List<decimal> SupportedValues { get; set; }

public bool ReadOnly { get; set; }

public bool Enabled { get; set; }

public string Name { get; set; }

public string Category { get; set; }

public string ShowDisabled { get; set; }

public string Description { get; set; }

public string Units { get; set; }

public string ControlType { get; set; }

public string Enumeration { get; set; }

public int CategoryOrder { get; set; }

public decimal Ratio { get; set; }

public uint TrailLength { get; set; }

public string GetMinPossibleValue();

public string GetMaxPossibleValue();

}

/// <summary>

/// This structure represents a single change proposal that user has made.

/// This change does not actually apply to the hardware until user has committed the

/// change.

/// </summary>

public struct ClientTuningProposal

{

public uint Id { get; set; }

public decimal Value { get; set; }

}

/// <summary>

/// This structure represents the feedback from the overclocking service to a change

/// proposal. When a change proposal was made to the overclocking service,the service

/// will examine its legitimacy and dependencies. For example, the value proposal

/// may not be valid for this control and other controls may also need to be updated

/// when this control changes. In this case, the overclocking service will provide

/// a list of objects in the structure like this to let the client know that

/// the following change should be allowed.

/// </summary>

public struct ClientTuningProposalResult

{

public uint Id { get; set; }

public decimal Value { get; set; }

public bool Enabled { get; set; }

public bool RebootRequired { get; set; }

}

/// <summary>

/// The custom delegate definition for tuning proposal change notification. When a

/// proposed change is made to the overclocking service, three things could happen on

/// the service side:

/// 1. The proposed change is acurate. The service will provide the next available

/// value to the caller. The caller is expected to use this value for display. This

/// is especially important for descrete tuning knobs, such as BCLK frequency.

/// 2. There might have some dependent tuning knobs that also need update.

/// 3. The proposed change is not valid. The service just simply reject the proposal.

/// If the proposed change requires reboot, the service will notify the caller via

/// this event too.

/// </summary>

/// <param name="proposalDelta">It contains the feedback from the service.

/// If the proposed value does not meet the monitor control's requirement, the new

/// value will be put into this parameter. If any dependency is detected, the

/// dependent monitor item(s) will also be added to this parameter as a list.

/// </param>

/// <param name="requiresReboot">If the proposed change requires reboot, this

/// parameter will be set to true by the overclocking service. The caller should

/// react to this request properly.

/// </param>

public delegate void ProposeChangesDelegate(List<ClientTuningProposalResult>

proposalDelta, bool requiresReboot);

/// <summary>

/// This interface defines all available tuning APIs. The caller of this interface

/// should call Initialize() API before trying to access any other property or API.

/// The client should also register a proper event if it is interested in the

/// activity.

/// </summary>

public interface ITuningLibrary

{

/// <summary>

/// The client should call this API before any tuning APIs.

/// This API connects the client to the overclocking service. It also

/// updates all available tuning controls including the XMP profiles

/// </summary>

void Initialize();

/// <summary>

/// Get all available tuning controls as a list of ClientTuningControl objects.

/// </summary>

/// <returns></returns>

List<ClientTuningControl> GetAvailableControls();

/// <summary>

/// Returns the tuning control settings for the control identified by the control

/// id.

/// </summary>

/// <param name="controlId">controlId as parameter</param>

/// <returns>Null or an object for ClientTuningControl</returns>

ClientTuningControl GetControl(uint controlId);

/// <summary>

/// Check to see if the tool supports BCLK (CPU reference clock) overclocking.

/// BCLK overclocking requires support from ME/ICC proxy service.

/// </summary>

/// <returns></returns>

bool IsBclkTunable();

/// <summary>

/// Returns true if there is any tunable controls in this category

/// </summary>

/// <returns></returns>

bool IsProcessorCoreTunable();

/// <summary>

/// Returns true if there is any tunable controls in this category

/// </summary>

/// <returns></returns>

bool IsProcessorGraphicTunable();

/// <summary>

/// Returns true if there is any tunable controls in this category

/// </summary>

/// <returns></returns>

bool IsProcessorRingCacheTunable();

/// <summary>

/// Returns true if there is any tunable controls in this category

/// </summary>

/// <returns></returns>

bool IsProcessorEdramTunable();

/// <summary>

/// Returns true if there is any tunable controls in this category

/// </summary>

/// <returns></returns>

bool IsMemoryTunable();

/// <summary>

/// The event that caller can register for notification when the change has

/// applied to the hardware. When changes have been applied to the hardware, the

/// caller should update all tuning knobs to reflect the current active value.

/// </summary>

event Action OnApplyChanges;

/// <summary>

/// The event that caller can register for notification when a change proposal

/// has been processed by the overclocking service.

/// </summary>

event ProposeChangesDelegate OnProposeChanges;

/// <summary>

/// This is the first step that user should do in order to overclocking the

/// system. When user changes a tuning knob, the change is sent to the

/// overclocking service via the API. The overclocking service verifies the

/// proposed change. It could either deny, adjust, or add additional changes

/// to the proposal. If the change requires reboot to take effect, the service

/// also notifies the caller.

/// The caller application should register the OnProposeChanges event from

/// the ITuningLibrary interface. The change proposal result will be passed

/// to the caller via this event. The caller application can examine the

/// result. If it satisfies the change, it can call ApplyChanges() to

/// apply the proposal to the hardware.

/// </summary>

/// <param name="proposals">This is what user has proposed to change. The

/// change proposal could have one or more items.

/// </param>

/// <param name="proposalDelta">This is what the service thinks about the

/// change. The feedback could have one or more change items.

/// </param>

/// <param name="requiresReboot">If any one of proposed changes requires

/// reboot, this parameter will be set to true by the service.

/// </param>

/// <returns></returns>

bool ProposeChange(List<ClientTuningProposal> proposals, out

List<ClientTuningProposalResult> proposalDelta,

out bool requiresReboot);

/// <summary>

/// In general, the overclocking process works like this. The user makes some

/// changes on the tuning control. The changes are sent to the overclocking

/// service for verification. The service may veto or provide more information

/// for the proposed change. Up to now, none of changes have applied to the

/// hardware. User is required to explicitly commit the change. When this

/// occurs, the ApplyChanges() API should be called. The fundamental actions

/// that the service will do are following:

/// 1. Setup the watchdog timer to ensure system can be recovered automatically.

/// 2. Apply all proposed changes to the hardware.

/// 3. Reboot system if it is required.

/// When the change is applied to the hardware without issues, the SDK will

/// fire the event OnApplyChanges. The caller application can register the event

/// and do more processing.

/// </summary>

/// <param name="forceRestart"></param>

/// <returns></returns>

bool ApplyChanges(bool forceRestart);

/// <summary>

/// User would like to discard the proposed change. This essentially causes

/// the overclocking service to reset the proposed change list.

/// </summary>

/// <returns></returns>

List<ClientTuningProposalResult> DiscardChanges();

/// <summary>

/// Get a list of SSD hard drives on the system.

/// </summary>

/// <returns></returns>

List<CoreDrive> GetDrives();

/// <summary>

/// Get a two dementional lists of XMP profiles. The inner list describes

/// one XMP profile. The outer list contains the number of XMP profiles.

/// </summary>

/// <returns></returns>

List<List<ClientTuningProposal>> GetXmpProfiles();

/// <summary>

/// The overclocking UI uses this API to notify the overclocking service

/// to reboot the system.

/// </summary>

void Restart();

/// <summary>

/// Returns the tunable status for the control identified by the control id.

/// </summary>

/// <param name="nControlId"></param>

/// <returns></returns>

bool IsControlTunable(uint nControlId);

/// <summary>

/// Returns the realtime tunable status for the control identified by the control id.

/// </summary>

/// <param name="nControlId"></param>

/// <returns></returns>

bool IsControlTunableRealTime(uint nControlId);

/// <summary>

/// Make a complete tuning operation for one or more tuning knobs. This is the

/// combination of ProposeChange() and ApplyChanges() operation. After this call,

/// the proposed changes including the derived changes will be made to hardware.

/// There is no need to call ApplyChanges() after this API call.

/// </summary>

/// <param name="proposals"></param>

/// <param name="bRequiresReboot"></param>

/// <returns>true if the operation succeeds</returns>

bool Tune(List<ClientTuningProposal> proposals, out bool bRequiresReboot);

/// <summary>

/// Make a complete tuning operation for exactly one tuning knob. This is the

/// combination of ProposeChange() and ApplyChanges() operation. After this call,

/// the proposed changes including the derived changes will be made to hardware.

/// There is no need to call ApplyChanges() after this API call.

/// </summary>

/// <param name="nControlId"></param>

/// <param name="dValue"></param>

/// <param name="bRequiresReboot"></param>

/// <returns>true if the operation succeeds</returns>

bool Tune(uint nControlId, decimal dValue, out bool bRequiresReboot);

/// <summary>

/// Make a complete tuning operation for exactly one tuning knob. This is the

/// combination of ProposeChange() and ApplyChanges() operation. After this call,

/// the proposed changes including the derived changes will be made to hardware.

/// There is no need to call ApplyChanges() after this API call.

/// </summary>

/// <param name="nControlId"></param>

/// <param name="dValue"></param>

/// <param name="bRequiresReboot"></param>

/// <returns>true if the operation succeeds</returns>

bool Tune(uint nControlId, uint dValue, out bool bRequiresReboot);

/// <summary>

/// User can adjust/set the delay between changes in voltage and ratio during the

/// tuning. This provides a time delay to allow hardware to respond to the

/// hardware configuration changes. A larger value will make the tuning process

/// longer. A short value may cause system crash. The default value is 1ms.

/// </summary>

/// <param name="millisecond"></param>

void SetTuningDelay(int millisecond);

/// <summary>

/// User have an option to choose which tuning controls should be restored to

/// default on suspend or shutdown.

/// restoreBootValueAtSuspend: if this is false, SDK will skip the restore to

/// boot value operation, if this is true, SDK will look at the second parameter.

/// RestoreExcludeControlList: if a control matches the exclude

/// list, SDK will not restore it to boot value at suspend or shutdown.

/// </summary>

/// <param name="restoreBootValueAtSuspend">true or false</param>

/// <param name="restoreExcludeControlList">a list of control IDs</param>

void SetSuspendRestoreOptions(bool restoreBootValueAtSuspend,

List<uint> restoreExcludeControlList);

/// <summary>

/// User have an option to choose which tuning controls should be restored to

/// default on suspend or shutdown.

/// if restoreBootValueAtSuspend = false, ignore other parameters.

/// if restoreBootValueAtSuspend = true, look at restoreControlList.

/// If restoreControlList is not NULL, the controls in the list will be restored

/// to default at suspend. The restoreExcludeControlList will be ignored.

/// If restoreControlList is NULL, look at restoreExcludeControlList.

/// if restoreExcludeControlList is not NULL, all other controls except the

/// controls in the list will be restored to default at suspend.

/// if restoreExcludeControlList is NULL, all controls will be restored to

/// default at suspend.

/// The priority order will be restoreBootValueAtSuspend, restoreControlList, and

/// restoreExcludeControlList.

/// </summary>

/// <param name="restoreBootValueAtSuspend"></param>

/// <param name="restoreExcludeControlList"></param>

/// <param name="restoreControlList"></param>

void SetSuspendRestoreOptions(bool restoreBootValueAtSuspend,

List<uint> restoreExcludeControlList,

List<uint> restoreControlList);

}

}

### Tuning ID Definition

Please note that not all controls are available on every platform.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Subsystem | Control IDs | Definition | Type | Units |
| Processor | 00h | Max Non-Turbo Processor Multiplier (also known as Flex Ratio) | Numeric | None |
| 02h | Processor Voltage Override Target | Numeric | Volts |
| 1Ah | Turbo Mode Enable | En/Dis | None |
| 1Dh | 1-Active Core Ratio Limit | Numeric | None |
| 1Eh | 2-Active Core Ratio Limit | Numeric | None |
| 1Fh | 3-Active Core Ratio Limit | Numeric | None |
| 20h | 4-Active Core Ratio Limit | Numeric | None |
| 2Ah | 5-Active Core Ratio Limit | Numeric | None |
| 2Bh | 6-Active Core Ratio Limit | Numeric | None |
| 60h | 7-Active Core Ratio Limit | Numeric | None |
| 61h | 8-Active Core Ratio Limit | Numeric | None |
| 6Bh | 9-Active Core Ratio Limit | Numeric | None |
| 6Ch | 10-Active Core Ratio Limit | Numeric | None |
| 22h | Processor Voltage Offset | Numeric | mV |
| 29h | Enhanced Intel® SpeedStep Technology Enable | En/Dis | None |
| 2Fh | Short Window Package Total Design Power Limit (PL2) | Numeric | Watts |
| 30h | Extended Window Package Total Design Power Limit (PL1) | Numeric | Watts |
| 43h | Short Window Time (Tau for PL2) | Numeric | Seconds |
| 42h | Extended Window Time (Tau for PL1) | Numeric | Seconds |
| 31h | Short Window Package Total Design Power Enable | En/Dis | None |
| 32h | Package Total Design Power Lock Enable | En/Dis | None |
| 33h | Processor Core Total Design Power Limit | Numeric | Watts |
| 34h | Processor Core Total Design Power Enable | En/Dis | None |
| 35h | Processor Core Total Design Power Lock Enable | En/Dis | None |
| 39h | Processor Core Current Maximum | Numeric | Amps |
| 47h | Runtime Turbo Override Enable | En/Dis | None |
| 50h | Overclocking Enable | En/Dis | None |
| 54h | Package Current Limit | Numeric | Amps |
| 58h | Processor Core Voltage Mode  *Enumeration Definition:*   * 0 – Adaptive * 1 – Static | Enum | None |
| 66h | Process Core Current Limit Maximum | Numeric | Amps |
| 72h | AVX2 (AVX256) Core Ratio Negative Offset | Numeric | None |
| 73h | AVX3 (AVX512) Core Ratio Negative Offset | Numeric | None |
| Graphics | 36h | Processor Graphics Core Total Design Power Limit | Numeric | Watts |
| 37h | Processor Graphics Core Total Design Power Enable | En/Dis | None |
| 38h | Processor Graphics Core Total Design Power Lock Enable | En/Dis | None |
| 3Ah | Processor Graphics Core Current Maximum | Numeric | Amps |
| 3Bh | Processor Graphics Slice Turbo Ratio Limit | Numeric | None |
| 51h | Processor Graphics Core Voltage Override Target | Numeric | Volts |
| 52h | Processor Graphics Core Voltage Mode  *Enumeration Definition:*   * 0 – Adaptive * 1 – Static | Enum | None |
| 53h | Processor Graphics Core Voltage Offset | Numeric | mV |
| 62h | Processor Graphics Unslice Voltage mode  *Enumeration Definition:*   * 0 – Adaptive * 1 – Static | Enum | None |
| 63h | Processor Graphics Unslice Voltage Override Target | Numeric | mV |
| 64h | Processor Graphics Unslice Voltage Offset | Numeric | mV |
| 65h | Processor Graphics Unslice Turbo Ratio Limit | Numeric | None |
| 68h | Processor Graphics Slice Current Limit Maximum | Numeric | Amps |
| 69h | Processor Graphics Unslice Current Limit Maximum | Numeric | Amps |
| System Agent | 25h | System Agent Voltage Override Target | Numeric | Volts |
| 55h | System Agent Voltage Offset | Numeric | mV |
| 67h | System Agent Current Limit Maximum | Numeric | Amps |
| Ring/CLR | 4Ch | Ring Ratio (HSW and later) | Numeric | None |
| 4Eh | Ring Voltage Mode  *Enumeration Definition:*   * 0 – Adaptive * 1 – Static | Enum | None |
| 4Dh | Ring Voltage Override Target | Numeric | Volts |
| 4Fh | Ring Voltage Offset | Numeric | mV |
| 6Ah | Ring Current Limit Maximum | Numeric | None |
| Clocking | 01h | Reference Clock Frequency | Numeric | MHz |
| 45h | Reference Clock Ratio | Numeric | None |
| 4Ah | Filter PLL Frequency | Numeric | MHz |
| 5Ah | PEG/DMI Ratio | Numeric | None |
| Voltage | 05h | Memory Voltage | Numeric | Volts |
| 3Dh | Processor PLL Voltage | Numeric | Volts |
| 3Eh | Processor IO Voltage | Numeric | Volts |
| 46h | Vboot Voltage | Numeric | Volts |
| 4Bh | Dynamic SVID Control | En/Dis | None |
| 57h | FIVR Efficiency Management Enable | En/Dis | None |
| 56h | FIVR Fault Enable | En/Dis | None |
| 59h | SVID Voltage Override | Numeric | V |
| 5Bh | I/O Analog Voltage Offset | Numeric | mV |
| 5Ch | I/O Digital Voltage Offset | Numeric | mV |
| 70h | VccU Voltage Offset | Numeric | mV |
| Memory | 13h | DDR Multiplier | Numeric | None |
| 49h | Memory Clock Multiplier | Numeric | None |
| 07h | CAS Latency (tCL) | Numeric | Clocks |
| 08h | Row Address to Column Address Delay (tRCD) | Numeric | Clocks |
| 09h | Row Precharge Time (tRP) | Numeric | Clocks |
| 0Ah | Row Active Time (tRAS) | Numeric | Clocks |
| 0Bh | Write Recovery Time (tWR) | Numeric | Clocks |
| 15h | Minimum Refresh Recovery Time (tRFC) | Numeric | Clocks |
| 16h | Row Active to Row Active delay (tRRD) | Numeric | Clocks |
| 17h | Internal Write to Read Command Delay (tWTR) | Numeric | Clocks |
| 18h | System Command Rate Mode | Numeric | None |
| 19h | Read to Precharge delay (tRTP) | Numeric | Clocks |
| 27h | Row Cycle Time (tRC) | Numeric | Clocks |
| 28h | Four Active Window Delay (tFAW) | Numeric | Clocks |
| 2Ch | Average Periodic Refresh Interval (tREFI) | Numeric | Clocks |
| 2Dh | Minimum CAS Write Latency Time (tCWL) | Numeric | Clocks |
| 40h | XMP Profile Selection  *Enumeration Definition:*   * 0 – Default SPD Profile * 1 – Custom Timing Profile * 2 – XMP Profile 1 * 3 – XMP Profile 2 | Enum | Profile |
| 5Dh | All Banks Row Pre-Charge Delay Time (tRPab) | Numeric | Clocks |
| 5Eh | ED RAM PLL Ratio | Numeric | None |

Table 2‑3 Tuning Control ID Definition

### Operation Theory

namespace OverclockingSdkTuningSampleApp

{

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using Intel.Overclocking.SDK;

using Intel.Overclocking.SDK.Tuning;

class Program

{

static void Main(string[] args)

{

List<ClientTuningControl> theTuningControlList = new

List<ClientTuningControl>();

IIntelOverclockingLibrary theSdk = new IntelOverclockingLibrary();

ITuningLibrary theTuningLib = theSdk.TuningLib;

theTuningLib.Initialize();

if (!theTuningLib.IsProcessorCoreTunable())

{

System.Console.WriteLine("Processor core cannot be tuned!");

return;

}

theTuningControlList = theTuningLib.GetAvailableControls();

if (theTuningControlList.Count > 0)

{

System.Console.WriteLine("There are total of " +

theTuningControlList.Count + "tuning knobs available.");

}

List<ClientTuningProposal> theTuningProposalList = new

List<ClientTuningProposal>();

// set 1-Active Core Ratio limit to 2GHz

theTuningProposalList.Add(new ClientTuningProposal() {

Id = 0x1D, Value = 20 });

// set 2-Active Core Ratio limit to 1.9GHz

theTuningProposalList.Add(new ClientTuningProposal() {

Id = 0x1E, Value = 19 });

List<ClientTuningProposalResult> theTuningProposalResultList = new

List<ClientTuningProposalResult>();

bool bRestartRequired = false;

if (!theTuningLib.ProposeChange(theTuningProposalList, out

theTuningProposalResultList, out bRestartRequired))

{

System.Console.WriteLine("Unable to propose the change to Processor Core

Ratio Limits");

return;

}

// The watchdog timer will be started before the changes are made to the

// hardware.

if (!theTuningLib.ApplyChanges(bRestartRequired))

{

System.Console.WriteLine("Unable to apply changes to the hardware!");

}

if (bRestartRequired)

{

System.Console.WriteLine("System will reboot in 10 seconds");

Thread.Sleep(10 \* 1000);

theTuningLib.Restart();

}

}

}

}

## Interface IMonitoringLibrary

### API Definitions

namespace Intel.Overclocking.SDK.Monitoring

{

/// <summary>

/// This class defines all properties associated with a monitor. A monitor

/// can be identified by either an ID or Name.

/// All properties except "Value" are static, which means they do not change

/// during runtime. The "Value" property changes overtime. The client software

/// must call the API GetValue(unit id) to get the current value for the

/// monitor.

/// </summary>

public class ClientMonitor

{

public uint Id { get; set; }

public decimal Value { get; set; }

public string Name { get; set; }

public string Units { get; set; }

public string FormatString { get; set; }

public decimal Minimum { get; set; }

public decimal Maximum { get; set; }

}

/// <summary>

/// This interface defines all monitoring APIs that are available to the

/// overclocking applications. The caller of this API must call Initialize()

/// API first before accessing any other properties or APIs.

/// </summary>

public interface IMonitoringLibrary

{

/// <summary>

/// Initialize this subsystem. It creates the connection to the XTU service.

/// and retrieves all monitoring items from the service. The available monitoring

/// items are cached in the SDK as a list of objects.

/// </summary>

/// <returns></returns>

void Initialize();

/// <summary>

/// Return the number of available monitors

/// </summary>

/// <returns>0 or more than 0</returns>

int GetNumberOfAvailableMonitors();

/// <summary>

/// Returns a list of available monitors on this system.

/// </summary>

/// <returns></returns>

List<ClientMonitor> GetAvailableMonitors();

/// <summary>

/// Returns the value for the monitor item as identified by the unique ID if the

/// specific monitor is ready to update. The available monitor IDs are predefined

/// at the design time.

/// The caller should call GetValue(id) only if MonitorIsReady(id) returns true.

/// </summary>

/// <param name="id"></param>

/// <returns>the current value of the monitoring item as identified by the

/// parameter</returns>

decimal GetValue(uint id);

/// <summary>

/// Returns the status of the specified monitor. If the value is set for this

/// monitor, the monitor is ready to report to the client.

/// The caller should call GetValue(id) only if MonitorIsReady(id) returns true.

/// </summary>

/// <param name="id"></param>

/// <returns>true or false</returns>

bool MonitorIsReady(uint id);

/// <summary>

/// Starts monitors on this system.

/// </summary>

void Start();

/// <summary>

/// Stops monitors on this system.

/// </summary>

void Stop();

/// <summary>

/// Returns status of monitor thread.

/// </summary>

bool IsRunning();

}

}

### Monitor ID Definitions

Please note not all monitor IDs are available on every platform.

|  |  |  |  |
| --- | --- | --- | --- |
| Monitor ID | Name | Units | Description |
| 0 | CPU Utilization | % | The percentage of the CPU's processing power currently being used. |
| 1 | CPU Thread {0} Utilization | % | The percentage of CPU Thread {0}'s processing power currently being used. |
| 2 | Memory Utilization | MB | The amount of system memory currently in use. |
| 3 | Thermal Throttling |  | The state of the CPU throttling itself to maintain temperature. |
| 4 | Active Core Count |  | The number of cores that are currently active. |
| 5 | Processor Frequency | GHz | The frequency of the fastest CPU core currently executing. |
| 6 | Graphics Frequency | MHz | The frequency of the processor graphics cores. |
| 7 | CPU Core Temperature 0 | 0C | Temperature of CPU Core 0. |
| 8 | CPU Core Temperature 1 | 0C | Temperature of CPU Core 1. |
| 9 | CPU Core Temperature 2 | 0C | Temperature of CPU Core 2. |
| 10 | CPU Core Temperature 3 | 0C | Temperature of CPU Core 3. |
| 11 | CPU Core Temperature 4 | 0C | Temperature of CPU Core 4. |
| 12 | CPU Core Temperature 5 | 0C | Temperature of CPU Core 5. |
| 13 | CPU Core Temperature 6 | 0C | Temperature of CPU Core 6. |
| 14 | CPU Core Temperature 7 | 0C | Temperature of CPU Core 7. |
| 15 | Reference Clock Frequency | MHz | The frequency of the reference clock. |
| 16 | Config TDP Ratio | x | The current configurable TDP maximum non-turbo ratio. |
| 17 | Memory Frequency | MHz | The frequency of the memory. |
| 18 | Turbo Boost Short Power Max | W | The currently active turbo boost short window power max. |
| 19 | Turbo Boost Power Max | W | The currently active turbo boost power max. |
| 20 | Processor Cache Frequency | GHz | The frequency of the interface that connects the processor’s cache to its cores. |
| 21 | EDRAM Frequency | MHz | The frequency of the EDRAM clock. |
| 22 | Power Limit Throttling |  | The state of the CPU throttling itself to maintain power limit. |
| 23 | Current Limit Throttling |  | The state of the CPU throttling itself to maintain current limit. |
| 24 | CPU Core 8 Temperature | 0C | Temperature of CPU Core 8. |
| 25 | CPU Core 9 Temperature | 0C | Temperature of CPU Core 9. |
| 1000 | CPU Total TDP | W | Amount of energy used by the CPU Package. |
| 1001 | CPU Core TDP | W | Amount of energy used by the IA Cores. |
| 1002 | Graphics TDP | W | Amount of energy used by the Graphics Core(s). |
| The Following Monitors are valid only if BIOS supports VSDD, TSDD, and FSDD ACPI Methods. | | | |
| 2701131777 | CPU Temperature | 0C | The temperature measured on the CPU. |
| 2701131778 | CPU Die Temperature | 0C | The temperature measured on the CPU Die. |
| 2701131779 | ICH Temperature | 0C | The temperature of the ICH. |
| 2701131780 | MCH Temperature | 0C | The temperature of the MCH. |
| 2701131781 | Voltage Regulator Temperature | 0C | The temperature of the voltage regulator. |
| 2701131782 | DIMM Temperature | 0C | The temperature of memory DIMM |
| 2701131783 | Motherboard Ambient Temperature | 0C | The ambient temperature of the motherboard. |
| 2701131784 | System Ambient Temperature | 0C | The ambient temperature of the system. |
| 2701131785 | CPU Inlet Temperature | 0C | The CPU inlet temperature. |
| 2701131786 | System Inlet Temperature | 0C | The system inlet temperature. |
| 2701131787 | System Outlet Temperature | 0C | The system outlet temperature. |
| 2701131788 | Power Supply Temperature | 0C | The system power supply temperature. |
| 2701131789 | Power Supply Inlet Temperature | 0C | The system power supply inlet temperature. |
| 2701131790 | Power Supply Outlet Temperature | 0C | The system power supply outlet temperature. |
| 2701131791 | Hard Drive Temperature | 0C | The temperature of the hard drive. |
| 2701131792 | Graphics Processor Unit (GPU) Temperature | 0C | The temperature of the graphics processor unit (GPU). |
| 2701131793 | Skin Temperature | 0C | The laptop skin temperature. |
| 2701131794 | Optical Disk Drive Temperature | 0C | The temperature of the optical disk drive. |
| 2701131795 | PCMCIA Slot Temperature | 0C | The temperature of the PCMCIA slot. |
| 2701131796 | PCH Temperature | 0C | The temperature of the PCH. |
| 2701131797 | Battery Temperature | 0C | The temperature of the battery unit. |
| 2717908993 | +12V Rail | V | The +12V rail monitor. |
| 2717908994 | -12V Rail | V | The -12V rail monitor. |
| 2717908995 | +5V Rail | V | The +5V rail monitor. |
| 2717908996 | +5V Backup Rail | V | The +5V backup rail monitor. |
| 2717908997 | -5V Rail | V | The -5V rail monitor. |
| 2717908998 | +3.3V Rail | V | The +3.3V rail monitor. |
| 2717908999 | +2.5V Rail | V | The +2.5V rail monitor. |
| 2717909000 | +1.5V Rail | V | The +1.5V rail monitor. |
| 2717909001 | Processor Core Voltage | V | Processor's Core Voltage. |
| 2717909002 | PSU Inlet Voltage | V | Power Supplie's Inlet Voltage Monitor. |
| 2717909003 | MCH Voltage | V | MCH Voltage Monitor. |
| 2717909004 | +3.3V Standby Rail | V | The +3.3V standby rail monitor. |
| 2717909005 | System Agent Voltage | V | System Agent Votlage Monitor. |
| 2717909006 | +1.8V Rail | V | The +1.8V rail monitor. |
| 2717909007 | PCH Voltage | V | PCH Voltage Monitor. |
| 2717909008 | Memory Voltage | V | Memory Voltage Monitor. |
| 2717909009 | Battery Voltage | V | Battery Voltage Monitor. |
| 2717909010 | CPU IO Voltage | V | CPU IO Voltage Monitor. |
| 2717909011 | CPU PLL Voltage | V | CPU PLL Voltage Monitor. |
| 2751463425 | CPU Fan Speed | RPM | The speed of the CPU fan. |
| 2751463426 | System Fan Speed | RPM | The speed of the system fan. |
| 2751463427 | MCH Fan Speed | RPM | The speed of the Memory Controller Hub fan. |
| 2751463428 | VR Fan Speed | RPM | The speed of the Voltage Regulator fan. |
| 2751463429 | Chassis Fan Speed | RPM | The speed of the chassis fan. |
| 2751463430 | Chassis Inlet Fan Speed | RPM | The speed of the chassis inlet fan. |
| 2751463431 | Chassis Outlet Fan Speed | RPM | The speed of the chassis outlet fan. |
| 2751463432 | PSU Fan Speed | RPM | The speed of the Power Supply Unit fan. |
| 2751463433 | PSU Inlet Fan Speed | RPM | The speed of the Power Supply Unit inlet fan. |
| 2751463434 | PSU Outlet Fan Speed | RPM | The speed of the Power Supply Unit outlet fan. |
| 2751463435 | Hard Disk Drive Fan Speed | RPM | The speed of the Hard Disk Drive fan. |
| 2751463436 | GPU Fan Speed | RPM | The speed of the Graphics Processing Unit fan. |
| 2751463437 | Auxiliary Fan Speed | RPM | The speed of the auxiliary fan. |
| 2751463438 | PCH Fan Speed | RPM | The speed of the Platform Controller Hub fan. |
| 2751463439 | Battery Fan Speed | RPM | The speed of the battery fan. |
| 2768240641 | Memory Temperature | RPM | The temperature of the memory. |
| 2768240642 | Power Supply Hot-Spot Temperature | 0C | The temperature of the power supply hot-spot. |
| 2768240643 | IOH Temperature | 0C | The temperature of the IOH. |
| 2768240644 | CPU Core Voltage | V | The CPU Core Voltage. |
| 2768240645 | SDRAM Voltage | V | Voltage measured at the SDRAM. |
| 2768240646 | Memory Channel A/B Voltage | V | The voltage of memory channels A and B. |
| 2768240647 | Memory Channel C/D Voltage | V | The voltage of memory channels C and D. |
| 2768240648 | +12V ATX Voltage | V | The positive 12 volt ATX power rail monitor. |
| 2768240649 | +12V ATX CPU Voltage | V | The positive 12 volt ATX CPU voltage rail monitor. |
| 2768240650 | IOH Cooling Fan Speed | RPM | The speed of the IOH cooling fan. |
| 2768240651 | Memory Cooling Fan Speed | RPM | The speed of the memory cooling fan. |
| 2768240652 | Graphics Voltage | V | The voltage of the processor graphics cores. |
| 2768240653 | Graphics Temperature | 0C | The temperature of the processor graphics cores. |

Table 2‑4 Monitor ID Defintion

### Operation Therory

namespace OverclockingSdkMonitoringSampleApp

{

using System;

using System.Collections.Generic;

using System.Linq;

using Intel.Overclocking.SDK;

using Intel.Overclocking.SDK.Monitoring;

class Program

{

static void Main(string[] args)

{

List<ClientMonitor> theMonitorList = new List<ClientMonitor>();

IIntelOverclockingLibrary theSdk = new IntelOverclockingLibrary();

IMonitoringLibrary theMonitor = theSdk.MonitoringLib;

theMonitor.Initialize();

theMonitorList = theMonitor.GetAvailableMonitors(false);

if (theMonitorList.Count > 0)

{

System.Console.WriteLine("There are total of " + theMonitorList.Count +

"monitors available.");

}

if (theMonitor.MonitorIsReady(5))

{

System.Console.WriteLine("The processor frequency is " +

theMonitor.GetValue(5) + "MHz");

}

}

}

}

## Interface IServiceInfoLibrary

### API Definitions

namespace Intel.Overclocking.SDK.ServiceInfo

{

/// <summary>

/// Report information related to watchdog timer

/// </summary>

public interface IWatchdogTimerInfo

{

/// <summary>

/// Returns true if the overclocking watchdog timer is detected on the system.

/// </summary>

/// <returns></returns>

bool IsWatchdogTimerPresent();

/// <summary>

/// Returns true if the overclocking watchdog timer is running on system startup.

/// The watchdog timer is used to allow system to automatically recover from

/// system crash that is caused by overclocking activity

/// </summary>

/// <returns>true or false</returns>

bool IsWatchdogTimerRunning();

/// <summary>

/// Returns true if the overclocking watchdog timer has indicated a failure on

/// system startup.

/// </summary>

/// <returns></returns>

bool HasWatchdogTimerFailed();

}

public interface IProcessorInfo

{

/// <summary>

/// Returns the branding string for the processor

/// </summary>

/// <returns></returns>

string GetBrandString();

/// <summary>

/// Returns the number of physical CPU cores.

/// </summary>

/// <returns></returns>

uint GetPhysicalCpuCoreCount();

/// <summary>

/// Returns the number of logical CPU cores.

/// </summary>

/// <returns></returns>

uint GetLogicalCpuCoreCount();

/// <summary>

/// Returns the available OC bins on the system

/// </summary>

/// <returns></returns>

uint GetOverclockableTurboBins();

/// <summary>

/// Returns true if system is unlocked. An unlocked system allows the software to

/// fine tune the system performance.

/// </summary>

/// <returns></returns>

bool IsSystemUnlocked();

/// <summary>

/// Check MSR FLEX\_RATIO[20] and [19:17] for OC support and also OC\_MAILBOX

/// command 0x1 to determine if

/// the system is overclockable. The system could be partially overclockable or

/// fully overclockable, but

/// as long as system is overclockable, this API returns true.

/// </summary>

/// <returns>true or false</returns>

bool IsOverclockSupported();

/// <summary>

/// Check to see if Intel(R) Turbo Boost Technology is enabled on this system.

/// The Intel(R) Turbo Boost

/// Technology is required for ratio overclocking for processor core, graphics,

/// and others in the CPU package.

/// </summary>

/// <returns></returns>

bool IsTurboBoostTechnologyEnabled();

/// <summary>

/// Returns the CPU feature flags in string format. The feature flags include ECX

/// and EDX flags for CPUID

/// information. The example feature flags include VMX, EIST, ACPI, SS, HTT, TM,

/// APIC, and etc.

/// </summary>

/// <returns></returns>

string GetCpuFeatureFlags();

/// <summary>

/// Returns the available instruction set that is available on the current CPU.

/// The example instruction

/// set includes AVX, SSE, SSE2, TSC, MMX, AES, and etc.

/// </summary>

/// <returns></returns>

string GetSupportedCpuInstructions();

/// <summary>

/// Get the CPU microcode update version. The version is represented as an

/// unsigned integer.

/// </summary>

/// <returns></returns>

uint GetMicrocodeUpdateVersion();

/// <summary>

/// Return true if Intel integrated graphics is available

/// </summary>

/// <returns></returns>

public bool IsIntegratedGraphicsPresent()

/// <summary>

/// Returns the number of extra turbo bins available. This API returns the value

/// of MSR 0x194 bit[19:17].

/// </summary>

/// <returns></returns>

public string NumAllowedOCTurboBins()

/// <summary>

/// Returns true if the CPU is unlocked. The part is unlocked if the number of

/// turbo bins is not 0

/// </summary>

/// <returns></returns>

public bool TurboOverclockable()

/// <summary>

/// Returns true if Enhanced Intel® SpeedStep Technology is enabled.

/// </summary>

/// <returns></returns>

public bool IsEistIsEnabled()

/// <summary>

/// Returns true if turbo is enabled and turbo is capable.

/// </summary>

/// <returns></returns>

public bool IsTurboEnabled()

/// <summary>

/// Returns true if CPUID.EAX(6):EAX(1) == 1

/// </summary>

/// <returns></returns>

public bool IsTurboCapable()

/// <summary>

/// Returns true if GFX supports ratio-based overclocking

/// </summary>

/// <returns></returns>

public bool IsProcessorGfxOCEnabled()

/// <summary>

/// Returns true if Processor supports ratio-based overclocking

/// </summary>

/// <returns></returns>

public bool IsProcessorIACoreOCEnabled()

/// <summary>

/// Returns true if Ring and Cache supports ratio-based overclocking

/// </summary>

/// <returns></returns>

public bool IsProcessorClrOCEnabled()

}

public interface IOverclockingServiceInfo

{

/// <summary>

/// Returns the overclocking service version in a string. The version is in the

/// following format:

/// Major.minor.rev.build#.

/// </summary>

/// <returns></returns>

string GetOverclockingServiceVersion();

/// <summary>

/// Returns the overclocking SDK version in a string. The version is in the

/// following format:

/// Major.minor.rev.build#.

/// </summary>

/// <returns></returns>

string GetOverclockingSdkVersion();

/// <summary>

/// Returns the session ID for the session that the service runs in.

/// </summary>

/// <returns></returns>

Guid GetOverclockingServiceSessionId();

}

public interface IServiceInfoLibrary

{

/// <summary>

/// Initialize this subsystem. It creates the connection to the overclocking

/// service and retrieves all system information items

/// from the service. The available system information items are cached in the

/// SDK object

/// </summary>

/// <returns></returns>

void Initialize();

/// <summary>

/// Returns an instance of the WatchdogTimerInfo class. The client should use

/// this property to get information about the watchdog timer.

/// </summary>

IWatchdogTimerInfo WatchdogTimer { get; }

/// <summary>

/// Returns an instance of the ProcessorInfo class. The client should use this

/// property to get any information related to the processor. For example,

/// Processor.GetBrandString()

/// </summary>

IProcessorInfo Processor { get; }

/// <summary>

/// Returns an instance of the OverclockingServiceInfo class. The client

/// should use this property to get the service information. For example,

/// OcServiceInfo.GetOverclockingServiceVersion()

/// </summary>

IOverclockingServiceInfo ServiceInfo { get; }

}

}

### Operation Theory

The client instantiates the class ServiceInfoApi. It then calls the initialize() function to start the operation. Once the initialize function is called, the client is free to get the service information. Here is the example of how this code can be used:

namespace OverclockingSdkClientApp

{

using Intel.Overclocking.SDK;

using Intel.Overclocking.SDK.ServiceInfo;

class Program

{

static void Main(string[] args)

{

IIntelOverclockingLibrary theSdk = new IntelOverclockingLibrary();

IServiceInfoLibrary theServiceInfo = theSdk.SystemInfoLib;

if (theServiceInfo.Initialize() == true);

{

if (theServiceInfo.WatchdogTimer.IsWatchdogTimerPresent())

{

System.Console.WriteLine("Watchdog Timer is present on the system");

}

if (theServiceInfo.Processor.IsOverclockSupported())

{

System.Console.WriteLine("This platform supports overclocking");

}

System.Console.WriteLine("The Intel overclocking service version is " +

theServiceInfo.ServiceInfo.GetOverclockingServiceVersion());

}

}

}

}

## Interface IWatchdogTimerInfo

### API Definitions

namespace Intel.Overclocking.SDK.ServiceInfo

{

/// <summary>

/// Report information related to watchdog timer

/// </summary>

public interface IWatchdogTimerInfo

{

/// <summary>

/// Returns true if the overclocking watchdog timer is detected on the system.

/// </summary>

/// <returns></returns>

bool IsWatchdogTimerPresent();

/// <summary>

/// Returns true if the overclocking watchdog timer is running on system startup.

/// The watchdog timer is used to allow system to automatically recover from

/// system crash that is caused by overclocking activity.

/// </summary>

/// <returns>true or false</returns>

bool IsWatchdogTimerRunning();

/// <summary>

/// Returns true if the overclocking watchdog timer has indicated a failure on

/// system startup.

/// </summary>

/// <returns></returns>

bool HasWatchdogTimerFailed();

}

}

## Interface IProcessorInfo

### API Definitions

namespace Intel.Overclocking.SDK.ServiceInfo

{

public interface IProcessorInfo

{

/// <summary>

/// Returns the branding string for the processor

/// </summary>

/// <returns></returns>

string GetBrandString();

/// <summary>

/// Returns the number of physical CPU cores.

/// </summary>

/// <returns></returns>

uint GetPhysicalCpuCoreCount();

/// <summary>

/// Returns the number of logical CPU cores.

/// </summary>

/// <returns></returns>

uint GetLogicalCpuCoreCount();

/// <summary>

/// Returns the available OC bins on the system

/// </summary>

/// <returns></returns>

uint GetOverclockableTurboBins();

/// <summary>

/// Returns true if system is unlocked. An unlocked system allows the software to

/// fine tune the system performance.

/// </summary>

/// <returns></returns>

bool IsSystemUnlocked();

/// <summary>

/// Check MSR FLEX\_RATIO[20] and [19:17] for OC support and also OC\_MAILBOX

/// command 0x1 to determine if

/// the system is overclockable. The system could be partially overclockable or

/// fully overclockable, but

/// as long as system is overclockable, this API returns true.

/// </summary>

/// <returns>true or false</returns>

bool IsOverclockSupported();

/// <summary>

/// Check to see if Intel(R) Turbo Boost Technology is enabled on this system.

/// The Intel(R) Turbo Boost

/// Technology is required for ratio overclocking for processor core, graphics,

/// and others in the CPU package.

/// </summary>

/// <returns></returns>

bool IsTurboBoostTechnologyEnabled();

/// <summary>

/// Returns the CPU feature flags in string format. The feature flags include ECX

/// and EDX flags for CPUID

/// information. The example feature flags include VMX, EIST, ACPI, SS, HTT, TM,

/// APIC, and etc.

/// </summary>

/// <returns></returns>

string GetCpuFeatureFlags();

/// <summary>

/// Returns the available instruction set that is available on the current CPU.

/// The example instruction

/// set includes AVX, SSE, SSE2, TSC, MMX, AES, and etc.

/// </summary>

/// <returns></returns>

string GetSupportedCpuInstructions();

/// <summary>

/// Get the CPU microcode update version. The version is represented as an

/// unsigned integer.

/// </summary>

/// <returns></returns>

uint GetMicrocodeUpdateVersion();

}

}

## Interface IOverclockingServiceInfo

### API Definitions

namespace Intel.Overclocking.SDK.ServiceInfo

{

public interface IOverclockingServiceInfo

{

/// <summary>

/// Returns the overclocking service version in a string. The version is in the

/// following format:

/// Major.minor.rev.build#.

/// </summary>

/// <returns></returns>

string GetOverclockingServiceVersion();

/// <summary>

/// Returns the overclocking SDK version in a string. The version is in the

/// following format:

/// Major.minor.rev.build#.

/// </summary>

/// <returns></returns>

string GetOverclockingSdkVersion();

/// <summary>

/// Returns the session ID for the session that the service runs in.

/// </summary>

/// <returns></returns>

Guid GetOverclockingServiceSessionId();

}

}

## Interface IProfileLibrary

This interface contains all APIs and data structures for the client application to manipulate the overclocking profile. The client application can use this interface to enumerate available profiles, add/remove/update/import/export profiles.

### Data Structures

namespace Intel.Overclocking.SDK.Profile

{

using System.Collections.Generic;

using System.Linq;

public class TuningControlItem

{

/// <summary>

/// Is this control enabled/disabled based on the current tuning mode

/// </summary>

public bool IsTuningModeEnabled { get; set; }

public uint TrailLength { get; set; }

/// <summary>

/// Order to display item in the DAP

/// </summary>

public int SortOrder { get; set; }

public bool Hidden { get; set; }

/// <summary>

/// Is this a compound control (E.G Turbo)

/// </summary>

public bool IsCompoundControl { get; set; }

/// <summary>

/// Order it should be displayed in for manual tuning

/// </summary>

public int TuningOrder { get; set; }

/// <summary>

/// Id of the control

/// </summary>

public string Id { get; set; }

/// <summary>

/// Type of the control

/// </summary>

public ControlType ControlType { get; set; }

public string FormatString { get; set; }

/// <summary>

/// Raw category string. E.G. Processor.Turbo.Core1

/// </summary>

public string Category { get; set; }

/// <summary>

/// Main category, E.G. Processor

/// </summary>

public string PrimaryCategory { get; set; }

/// <summary>

/// Subcategory, E.G. Turbo

/// </summary>

public string SubCategory1 { get; set; }

/// <summary>

/// Second subcategory E.G. Core1

/// </summary>

public string SubCategory2 { get; set; }

public string SubCategory3 { get; set; }

/// <summary>

/// Should this control be made visible even if not settable/read only?

/// </summary>

public bool ShowDisabled { get; set; }

/// <summary>

/// Can this control be modified

/// </summary>

public bool IsSettable { get; set; }

public string Error { get; set; }

public decimal Ratio { get; set; }

// Is this part locked

public bool IsLockedPart { get; set; }

/// <summary>

/// Name of the control

/// </summary>

public string Name { get; set; }

/// <summary>

/// Is this control displayed in the DAP table

/// </summary>

public bool IsDisplayedInDap { get; set; }

/// <summary>

/// Description of the control

/// </summary>

public string Description { get; set; }

public string RelatedControls { get; set; }

/// <summary>

/// Value of control at boot

/// </summary>

public TuningValue BootValue { get; set; }

/// <summary>

/// Default system value for control

/// </summary>

public TuningValue DefaultValue { get; set; }

/// <summary>

/// Active value that the system is running for this control

/// </summary>

public TuningValue ActiveValue { get; set; }

/// <summary>

/// Value we are proposing to set this control to

/// </summary>

public TuningValue ProposedValue { get; set; }

/// <summary>

/// Is a reboot required to activate this setting

/// </summary>

public bool IsRebootRequired { get; set; }

/// <summary>

/// Is this control enabled or not

/// </summary>

public bool IsEnabled { get; set; }

/// <summary>

/// Value of this control at the start of the UI tuning session

/// </summary>

public TuningValue SessionStartValue { get; set; }

/// <summary>

/// Can reboot be forced even if the control does not require a reboot, e.g. HCF on sandybridge which is more performant with a reboot

/// </summary>

public bool AllowForceReboot { get; set; }

/// <summary>

/// List of possible valid values to set the control to

/// </summary>

public List<TuningValue> PossibleValues { get; set; }

public string Units { get; set; }

public string Enumeration { get; set; }

}

public struct TuningValue

{

public string Display { get; set; }

public decimal Value { get; set; }

public string Metadata1 { get; set; }

public string Metadata2 { get; set; }

}

public class SystemHardwareItem

{

public string Name { get; set; }

public string Value { get; set; }

public string Category { get; set; }

public string Id { get; set; }

public Guid ProfileId { get; set; }

}

public class XtuTuningProfile

{

public List<TuningControlItem> Controls { get; set; }

public List<SystemHardwareItem> Hardware { get; set; }

public List<TuningMode> TuningModes { get; set; }

public string ProfileName { get; set; }

public string ProfileId { get; set; }

public int DisplayOrder { get; set; }

public string BenchmarkScore { get; set; }

public string MaxProcFrequency { get; set; }

public string HighestCPUTemperature { get; set; }

public DateTime ModifiedDate { get; set; }

public DateTime CreationDate { get; set; }

public XtuTuningChangeList TuningProfile { get; set; }

public bool IsReadOnly { get; set; }

public bool IsUserVisible { get; set; }

public bool PreventExport { get; set; }

public bool OverclockingRestricted { get; set; }

public bool EngineeringSample { get; set; }

public bool PowerSourceChange { get; set; }

public bool PowerPlanChange { get; set; }

}

public struct IncompatibleImportResolutionItem

{

public string ControlName { get; set; }

public string ControlId { get; set; }

public TuningValue ImportedValue { get; set; }

public string ImportedDisplayValue

{

get

{

if (this.ImportedValue.Display != null)

{

return this.ImportedValue.Display;

}

else

{

return this.ImportedValue.Value.ToString();

}

}

}

}

public struct TuningItem

{

public string ControlId;

public bool IsEnabled;

public bool IsRebootRequired;

public TuningValue ProposedValue;

}

public class XtuTuningChangeList

{

public List<TuningItem> ProposedValues { get; set; }

public bool RebootRequired { get; set; }

}

/// <summary>

/// Imported Profile's List of matched, modified & incompatible controls

/// </summary>

public struct ImportResolutionControlsList

{

public List<ImportResolutionItemList> MatchedImportResolutionItems { get; set; }

public List<ImportResolutionItemList> ModifiedImportResolutionItems { get; set; }

public List<IncompatibleImportResolutionItem> IncompatibleImportResolutionItems { get; set; }

public List<XtuTuningChangeList> ProposedControlsList { get; set; }

public string UserId { get; set; }

}

public class ImportResolutionItemList

{

public ImportResolutionItemList(TuningControlItem control, TuningValue importedValue, TuningValue proposedValue)

{

this.Control = control;

this.ImportedValue = importedValue;

this.ProposedValue = proposedValue;

}

public TuningControlItem Control { get; set; }

public TuningValue ImportedValue { get; set; }

public TuningValue ProposedValue { get; set; }

}

}

### Enumerations

namespace Intel.Overclocking.SDK.Profile

{

/// <summary>

/// State of importing profile

/// </summary>

public enum ImportState

{

ImportCanceled,

InvalidFile,

ProcessorMismatch,

ImportSucceeded,

OverclockingLocked,

EistDisabled,

TurboDisabled,

ImportInvalidPlatform,

ProfileAlreadyExist

}

/// <summary>

/// State of the exporting profile

/// </summary>

public enum ExportState

{

ExportCanceled,

InvalidFile,

InvalidFilePath,

ExportSucceeded,

ExportPrevented,

InvalidProfileId

}

/// <summary>

/// ControlType Summary

/// </summary>

public enum ControlType

{

None,

Range,

OnOff,

Turbo,

MemoryTimingGrid,

Xmp,

DriveRange,

DriveComposite,

DriveThreeWay,

DriveOnOff,

ThreeWayToggle,

Checkbox,

Derived,

ReadOnly

}

}

### API Definitions

namespace Intel.Overclocking.SDK.Profile

{

using System;

using System.Collections.Generic;

public interface IProfileLibrary

{

/// <summary>

/// Initialize the library. This should be the first API to be called

/// </summary>

void Initialize();

/// <summary>

/// Return the default profile. The default profile is generated by

/// ProfileLibrary on startup. Once it it generated,

/// the default profile will be cached in SDK for future reference.

/// </summary>

/// <returns></returns>

XtuTuningProfile GetDefaultProfile();

/// <summary>

/// Return the list of all available profiles at runtime. The source of profiles

/// could be one of the following:

/// - Default Profile: generated at startup. It cannot be saved or exported.

/// - Saved Profiles: These are profiles that are added by the end user.

/// - Imported Profiles: User could import any profiles to runtime.

/// The SDK is the only place to cache the profile list.

/// </summary>

/// <returns></returns>

List<XtuTuningProfile> GetProfiles();

/// <summary>

/// Return the profile that is identified by the profile name. The profile

/// name is typically the same as the filename for the profile.

/// </summary>

/// <param name="profileName">The name for the profile.

/// </param>

/// <returns></returns>

XtuTuningProfile GetProfile(string profileName);

/// <summary>

/// Import a profile into the system. There are several things that are

/// happening when a profile is imported to the system:

/// 1. The profile will be validated to make sure it is compatible with the

/// system. If the profile with the same profile ID already exists, the

/// profile will not be imported. The return value of The API will be

/// ProfileAlreadyExist. If the profile is not valid for this platform,

/// the return code should be ImportInvalidPlatform.

/// 2. The profile will be added to the profile list for future reference.

///

/// The caller will be updated with the ImportState and the Profile name.

/// If the profile with the same profileID exists in the system, the profile

/// will not be imported. The API will return a ProfileAlreadyExist.

/// </summary>

/// <param name="filepath">The complete file path for the profile to be imported.

/// </param>

/// <param name=" profileName">The profile name will be passed out to the caller.

/// </param>

/// <returns>ImportState</returns>

ImportState ImportProfile(string filepath, out string profileName);

/// <summary>

/// This contains the Matched, Modified & Incompatible controls in the imported

/// profile. This should be called after ImportProfile API.

/// </summary>

/// <returns></returns>

ImportResolutionControlsList GetImportedControlList();

/// <summary>

/// Export an existing profile to a profile if the profile is exportable.

/// </summary>

/// <param name="profileId">The ID to identify the profile to be exported</param>

/// <param name="filepath">The complete file path for the new profile</param>

ExportState ExportProfile(string profileId, string filepath);

/// <summary>

/// Remove a profile from the XtuTuningProfile list

/// </summary>

/// <param name="profileName"></param>

/// <returns>True if the profile is found and successfully deleted</returns>

bool RemoveProfile(string profileName);

/// <summary>

/// Update the existing profile. The profile will be identified by the profile ID

/// </summary>

/// <param name="profile"></param>

void UpdateProfile(XtuTuningProfile profile);

/// <summary>

/// Propose the OC settings in the specified profile to be the system. If this

/// API is successful, the caller can call ApplyChanges from the ITuningLibrary API to

/// apply changes to the hardware.

/// The caller application can register for the OnProposeChanges event from

/// the ITuningLibrary interface. The change proposal result will be passed

/// to the caller via this event. The caller application can examine the

/// result.

/// </summary>

/// <param name="profileName"></param>

/// <returns>True if profile was proposed successfully</returns>

bool ProposeProfile(string profileName);

}

}

### Operation Theory

namespace OverclockingSdkProfileSampleApp

{

using System;

using System.Collections.Generic;

using System.Linq;

using Intel.Overclocking.SDK;

using Intel.Overclocking.SDK.Tuning;

using Intel.Overclocking.SDK.Profile;

class Program

{

private static bool bRebootRequired = false;

private static void OnProposeChangesComplete(List<ClientTuningProposalResult>

proposalDelta, bool requiresReboot)

{

bRebootRequired = requiresReboot;

System.Console.WriteLine("The number of changed controls = " +

proposalDelta.Count);

}

private static void OnApplyChangeComplete(ApplyChangeStatus theStatus)

{

System.Console.WriteLine("The proposed changes have been applied to hardware,

status=" + theStatus);

}

static void Main(string[] args)

{

// Instantiate the SDK library

IIntelOverclockingLibrary theSdk = new IntelOverclockingLibrary();

// Get reference to sub libraries

IProfileLibrary theProfileLib = theSdk.ProfileLib;

ITuningLibrary theTuningLib = theSdk.TuningLib;

// Initialize sub libraries

theProfileLib.Initialize();

theTuningLib.Initialize();

// Register for event notification

theTuningLib.OnProposeChanges += OnProposeChangesComplete;

theTuningLib.OnApplyChanges += OnApplyChangeComplete;

// Import a profile to the profile list

string theImportedProfileName = null;

ImportState theImportState =

theProfileLib.ImportProfile(@"c:\temp\performance\_profile.xtu",

out theImportedProfileName);

if (theImportState == ImportState.ImportSucceeded)

{

System.Console.WriteLine("The imported profile name is {0}.",

theImportedProfileName);

theTuningLib.ApplyChanges(bRebootRequired);

}

else

{

System.Console.WriteLine("The profile is not imported, status={0}",

theImportState);

}

// It should have 5 profiles: Default, low\_performance\_profile,

// mid\_performance\_profile, high\_performance\_profile, and External Profile

List<XtuTuningProfile> theTuningProfileList = theProfileLib.GetProfiles();

if (theTuningProfileList.Count > 0)

{

System.Console.WriteLine("There are total of " +

theTuningProfileList.Count + "profiles available.");

}

// Export the imported profile to a file

ExportState theExportState =

theProfileLib.ExportProfile(theImportedProfileName,

@"c:\temp\exportprofile.xtu");

if (theExportState == ExportState.ExportSucceeded)

{

System.Console.WriteLine("The profile ID ({0}) is exported to a file.",

theImportedProfileName);

}

else

{

System.Console.WriteLine("The profile is not exported, status={0}",

theExportState);

}

// Remove the imported profile. It should have 4 profiles.

theProfileLib.RemoveProfile(theImportedProfileName);

theTuningProfileList = theProfileLib.GetProfiles();

if (theTuningProfileList.Count > 0)

{

System.Console.WriteLine("There are total of " +

theTuningProfileList.Count + "profiles available.");

}

// Set the active profile

if (theProfileLib.SetActiveProfile("high\_performance\_profile"))

{

System.Console.WriteLine("The proposed profile name is {0}.",

theImportedProfileName);

theTuningLib.ApplyChanges();

}

// Update the dummy profile

XtuTuningProfile myTuningProfile = theTuningProfileList[1];

myTuningProfile.ModifiedDate = DateTime.Now;

theProfileLib.UpdateProfile(myTuningProfile);

}

}

}

## Interface IAppProfileLibrary

### API Definitions

namespace Intel.Overclocking.SDK.Profile

{

public delegate void UpdateActiveProcessHandler(string processName);

public delegate void UpdatePowerSourceHandler(bool bAcPower);

public interface IAppProfileLibrary

{

/// <summary>

/// The SDK uses this event to notify the client application when the topmost

/// application has changed. This implicates the change of the associated profile

/// as well. This event is only available when the AppProfile feature is turned

/// on (API Start() has called).

/// </summary>

event UpdateActiveProcessHandler OnActiveProcessChanged;

/// <summary>

/// The SDK uses this event to notify the client application when the power

/// source has

/// changed from AC to DC or DC to AC. This could implicate the change of the

/// associated profile. This event is only available when the AppProfile feature

/// is turned on.

/// </summary>

event UpdatePowerSourceHandler OnPowerSourceChanged;

/// <summary>

/// The SDK uses this event to notify the client application that the

/// availability of profiles has changed. The client application should update

/// its profile list in response to this event.

/// </summary>

event Action OnProfilesChanged;

/// <summary>

/// Initialize the library. This should be the first API to be called for this

/// interface.

/// </summary>

void Initialize();

#region BATTERY\_LEVEL

/// <summary>

/// Enable Battery Level Monitoring.

/// </summary>

void EnableBatteryLevelWatch();

/// <summary>

/// Disable Battery Level Monitoring.

/// </summary>

void DisableBatteryLevelWatch();

/// <summary>

/// This property holds the battery level threshold in percentage. If the

/// Battery level is below this value, the app-profile monitoring will be

/// stopped.

/// </summary>

decimal BatteryThreshold { get; set; }

#endregion

#region APP\_PROFILE\_MONITORING\_PROCESS

/// <summary>

/// Start the application - profile monitoring. This causes the tuning profile

/// automatically to be switched at runtime in response to the top layer user

/// application. When this happens, the manual tuning will be disabled.

/// </summary>

void Start();

/// <summary>

/// Stop the application - profile monitoring. This causes the baseline profile

/// to be loaded automatically. User is now allowed to manually tune the system.

/// </summary>

void Stop();

/// <summary>

/// Add an application - profile pair to the app-profile pair list.

/// </summary>

/// <param name="processName"></param>

/// <param name="acProfileName"></param>

/// <param name="dcProfileName"></param>

/// <returns></returns>

bool AddProcessProfile(string processName, string acProfileName,

string dcProfileName);

/// <summary>

/// Remove an application-profile pair from the app-profile pair list.

/// </summary>

/// <param name="processName"></param>

/// <returns></returns>

bool RemoveProcessProfile(string processName);

/// <summary>

/// Update the existing application-profile pair.

/// </summary>

/// <param name="processName"></param>

/// <param name="acProfileName"></param>

/// <param name="dcProfileName"></param>

/// <returns></returns>

bool UpdateProcessProfile(string processName, string acProfileName,

string dcProfileName);

/// <summary>

/// Enable or disable an application-profile pair

/// </summary>

/// <param name="processName"></param>

/// <param name="enable"></param>

void EnableProcessProfile(string processName, bool enable);

#endregion

}

}

## Interface IProcessLibrary

### API Definitions

namespace Intel.Overclocking.SDK.ProcessLauncher

{

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

public interface IProcessLibrary

{

/// <summary>

/// Launch and execute a shell application in user session.

/// </summary>

/// <param name="path"></param>

/// <param name="args"></param>

/// <param name="launchTopMost"></param>

/// <returns></returns>

Process LaunchShellExecAsUser(string path, string args, bool launchTopMost);

/// <summary>

/// Launch an user application in user session.

/// </summary>

/// <param name="path"></param>

/// <param name="args"></param>

/// <returns></returns>

Process LaunchProcess(string path, string args);

/// <summary>

/// Verify the signing certificate of the process before launch it in an user session.

/// </summary>

/// <param name="path"></param>

/// <param name="args"></param>

/// <returns></returns>

Process ValidateAndLaunchProcess(string path, string args);

}

}

## Interface IBenchmarkLibrary

### API Definitions

namespace Intel.Benchmark.SDK.Profile

{

using System;

using Intel.Overclocking.Benchmark.Profile;

/// <summary>

/// BenchMarkFailure can happen due to Benchmark failure and Benchmark run stopped

/// by user

/// </summary>

public enum BenchMarkStatus ////benchmarkstatus

{

BenchmarkCanceledByUser,

BenchmarkFailed,

BenchmarkCanceledByMonitors,

BenchmarkSuccess

}

public struct XtuBenchmarkData

{

public decimal BenchmarkScore;

public decimal MaxCpuFrequency;

public decimal HighestCpuTemperature;

public BenchMarkStatus Status;

public BenchmarkId BenchmarkId;

}

public enum HwbotProfileUploadStatus

{

HWBOT\_UPLOAD\_SUCCESSFUL = 0,

HWBOT\_UPLOAD\_INVALID\_PROFILE,

HWBOT\_UPLOAD\_NO\_NETWORK\_CONNECTION,

HWBOT\_UPLOAD\_INVALID\_URL,

HWBOT\_UPLOAD\_NO\_DEFAULT\_BROWSER,

HWBOT\_UPLOAD\_EXCEPTION,

HWBOT\_UPLOAD\_FAILED

}

public struct BenchmarkType

{

public BenchmarkId Id;

public string Name;

public string ShortName;

public bool HWBOTSupported;

public string Description;

}

public enum BenchmarkId

{

IntelXtuBenchmark,

IntelXtuBenchmark2

}

public interface IBenchmarkLibrary

{

/// <summary>

/// Event notification for benchmark completion.

/// The completion status is recorded in XtuBenchmarkData.

/// </summary>

event Action<XtuBenchmarkData> BenchmarkfullRunCompleted;

/// <summary>

/// Event notification for benchmark progress.

/// 0 means the benchmark is just get started.

/// </summary>

event Action<double> BenchMarkPercentDone;

/// <summary>

/// Returns a list of information about the benchmarks available on this system.

/// The list is meant to provide display information for a user interface that

/// uses the Benchmarking SDK

/// </summary>

IEnumerable<BenchmarkType> AvailableBenchmarks { get; }

/// <summary>

/// Initialize the benchmark library.

/// Application must call this API before accessing other functions.

/// </summary>

/// <returns></returns>

void Initialize();

/// <summary>

/// Function retained for backwards compatibility.

/// Equivalent to StartBenchmarkRun(BenchmarkId.IntelXTUBenchmark)

/// </summary>

void StartBenchmarkRun();

/// <summary>

/// Starts a benchmark run. Previous to calling this method, caller should

/// register the BenchmarkfullRunCompleted and BenchMarkPercentDone events.

/// </summary>

/// <param name="benchmarkToRun">The ID of the benchmark to run</param>

void StartBenchmarkRun(BenchmarkId benchmarkToRun);

/// <summary>

/// Cancel the benchmark operation. No score will be generated when the

/// benchmark is cancelled or failed.

/// </summary>

/// <param name="canceledByUser">Set to True if the benchmark is stopped by user.

/// </param>

/// <returns></returns>

void StopBenchmarkRun(bool canceledByUser);

/// <summary>

/// Query if benchmark is currently running

/// </summary>

/// <returns>true or false</returns>

bool IsBenchmarkRunning();

/// <summary>

/// Submit the benchmark score along with the XTU profile to HWBOT.org.

/// HWBOT.org could reject the submission for many reasons, for example,

/// there is no valid benchmark score or profile integrity is compromised.

/// </summary>

/// <param name="profileId">Set to Null or Empty String for online completion.

/// If you want to submit a profile that is saved previously, you can specify

/// the profileID. Otherwise, use Null or Empty string.

/// </param>

/// <returns></returns>

HwbotProfileUploadStatus CompareOnline(string profileId);

}

}

### Operation Theory

namespace XtuCLI

{

using System;

using System.ComponentModel;

using System.IO;

using System.Threading;

using System.Windows;

using System.Windows.Forms;

using Intel.Benchmark.SDK.Profile;

using Intel.Overclocking.Benchmark.Profile;

using Intel.Overclocking.SDK.Profile;

public partial class MainWindow : Window

{

private static XtuBenchmarkData benchmarkRunResult = new XtuBenchmarkData();

private static bool controlFocus = false;

private static bool benchmarkRunValid = false;

private static bool benchmarkRunning = false;

private IBenchmarkLibrary benchLib = BenchmarkLibrary.Instance;

public MainWindow()

{

this.InitializeComponent();

this.benchLib.Initialize();

RunBenchmark.IsEnabled = true;

StopBenchmark.Visibility = Visibility.Collapsed;

CompareButton.IsEnabled = false;

ScoreGrid.Visibility = Visibility.Visible;

SpinnerGrid.Visibility = Visibility.Hidden;

this.benchLib.BenchmarkfullRunCompleted += new

Action<XtuBenchmarkData>(this.CaptureBenchResults);

}

private void OnBenchControlStart()

{

this.UpdateSpinnerVisibility(true);

}

private void UpdateSpinnerVisibility(bool spinnerVisible)

{

if (spinnerVisible)

{

Dispatcher.BeginInvoke((ThreadStart)(() =>

{

ScoreGrid.Visibility = Visibility.Hidden;

SpinnerGrid.Visibility = Visibility.Visible;

}));

}

else

{

Dispatcher.BeginInvoke((ThreadStart)(() =>

{

ScoreGrid.Visibility = Visibility.Visible;

SpinnerGrid.Visibility = Visibility.Hidden;

}));

}

}

private void OnBenchmarkingRunningFail(BenchMarkStatus failure)

{

switch (failure)

{

case BenchMarkStatus.BenchmarkCanceledByUser:

Dispatcher.BeginInvoke((ThreadStart)(() =>

{

this.benchmarkingScore.Text = "--";

this.maximumFrequency.Text = "--";

this.highestCpuFrequency.Text = "--";

}));

break;

case BenchMarkStatus.BenchmarkFailed:

Dispatcher.BeginInvoke((ThreadStart)(() =>

{

this.benchmarkingScore.Text = "Error";

this.maximumFrequency.Text = "Error";

this.highestCpuFrequency.Text = "Error";

}));

break;

default:

Dispatcher.BeginInvoke((ThreadStart)(() =>

{

this.benchmarkingScore.Text = "--";

this.maximumFrequency.Text = "--";

this.highestCpuFrequency.Text = "--";

}));

break;

}

this.UpdateSpinnerVisibility(false);

}

private void CaptureBenchResults(XtuBenchmarkData benchResultsList)

{

Dispatcher.BeginInvoke((ThreadStart)(() =>

{

this.benchmarkingScore.Text = benchResultsList.BenchmarkScore + " Marks";

this.maximumFrequency.Text = Math.Round(benchResultsList.MaxCpuFrequency,

2) + " MHz";

this.highestCpuFrequency.Text =

Math.Round(benchResultsList.HighestCpuTemperature, 2) + " °C";

RunBenchmark.IsEnabled = true;

StopBenchmark.Visibility = Visibility.Hidden;

CompareButton.IsEnabled = true;

}));

benchmarkRunResult.BenchmarkScore = benchResultsList.BenchmarkScore;

benchmarkRunResult.MaxCpuFrequency = benchResultsList.MaxCpuFrequency;

benchmarkRunResult.HighestCpuTemperature =

benchResultsList.HighestCpuTemperature;

this.UpdateSpinnerVisibility(false);

}

private void Exit\_onClick(object sender, RoutedEventArgs e)

{

this.Close();

}

private void ImportTextBlock\_OnClick(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Multiselect = true;

openFileDialog.DefaultExt = ".xtu";

openFileDialog.InitialDirectory =

Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments),

"Intel XTU Profiles");

if (openFileDialog.ShowDialog() != System.Windows.Forms.DialogResult.None)

{

foreach (string filename in openFileDialog.FileNames)

{

ImportTextBlock.Text = Path.GetFullPath(filename);

}

}

}

private void BenchMarkControl\_OnGotFocus(object sender, RoutedEventArgs e)

{

controlFocus = true;

benchmarkRunning = true;

benchmarkRunValid = controlFocus || benchmarkRunning;

}

private void BenchMarkControl\_OnLostFocus(object sender, RoutedEventArgs e)

{

controlFocus = false;

benchmarkRunValid = controlFocus || benchmarkRunning;

}

private void RunBenchmark\_OnPreviewKeyDown(object sender, KeyEventArgs e)

{

e.Handled = true;

}

private void RunBenchmark\_Click(object sender, RoutedEventArgs e)

{

Dispatcher.BeginInvoke((ThreadStart)(() =>

{

RunBenchmark.IsEnabled = false;

StopBenchmark.Visibility = Visibility.Visible;

CompareButton.IsEnabled = false;

}));

this.benchLib.StartBenchmarkRun();

}

private void StopBenchmark\_Click(object sender, RoutedEventArgs e)

{

RunBenchmark.IsEnabled = true;

StopBenchmark.Visibility = Visibility.Hidden;

this.benchLib.StopBenchmarkRun(true);

}

private void CompareOnline(object sender, RoutedEventArgs e)

{

this.benchLib.CompareOnline(String.Empty);

}

}

}

## Interface IEventBroadcastLibrary

### API Definition

namespace Intel.Overclocking.SDK.Event

{

using System;

public delegate void MonitorStateChangeDelegate(bool monitorOn);

public interface IEventBroadcastLibrary

{

/// <summary>

/// this event will inform client application that the monitor state is changed.

/// </summary>

event MonitorStateChangeDelegate OnMonitorStateChanged;

/// <summary>

/// During the overclocking process, the graphics device may experience

/// difficulties.

/// In this case, the graphic driver will reset the hardware and send

/// notification

/// to the overclocking service. The overclocking service uses this event to

/// notify

/// the caller if the graphic driver reset has happened. In response, the caller

/// should update all graphics related tuning controls, such as ratio, voltage,

/// IccMax, power limit, and etc.

/// </summary>

event Action GraphicsDriverReset;

/// <summary>

/// Tuning controls could be updated by XTU and other overclocking software that

/// are derived from XTU SDK.This event will be set by XTU SDK when this happens.

/// The client application should register a callback for this event. When this

/// event happens, the client application should query the XTU SDK for active

/// tuning controls and update them accordingly in response to this event.

///

/// However, this event does not cover a scenario such as the hardware has been

/// changed by any application that is not derived from XTU SDK.

/// </summary>

event Action TuningControlReset;

/// <summary>

/// This should be the first API that the caller should call before any API is

/// called. This API connects the client to the overclocking service. It also

/// updates all available tuning controls including the XMP profiles

/// </summary>

void Initialize();

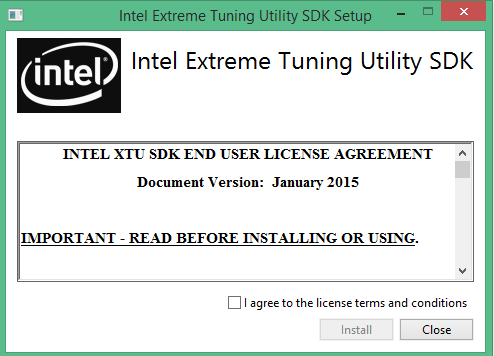
}

}

# XTU SDK Installation

The XTU SDK installer is **NOT** generally available from Intel Download Center. The XTU SDK installer is **NOT** part of the regular XTU installer either. The XTU SDK installer will be posted to Intel VIP site once it is available. The interesting party should contact the Intel AE to obtain the access.

The XTU SDK installer filename should be called as IntelOverclockingSdk.exe. Double clicking on the executable, you should see the following dialog. The user must check “I agree to the license terms and conditions” in order to continue the installation.



The Intel® Extreme Tuning Utility SDK can be installed on any computer running Windows 7 or above operating system. By default, it is installed at **C:\Program Files (x86)\Intel\Intel(R) Extreme Tuning Utility\SDK**. The SDK folder contains the following sub-folders and components in distribution:

* **\Binary**
  + IntelOverclockingSdk.dll: this module contains APIs for performance tuning, monitoring, system information service, profile management, and App-Profile pairing service. This assembly requires ProfileHelperModel.dll to be present.
  + ProfileHelperModel.dll:this module contains helper functions for profile management.
  + IntelBenchmarkSdk.dll: this module contains user controls for benchmark operation and HWBOT profile submission operation. This assembly requires IntelOverclockingSdk.dll and ProfileHelperModel.dll to be present.
  + P95bench.exe: this executable is the XTU benchmarking executable file.
  + IntelXTUBenchmark2.exe: This is another variety of p95 benchmark we brand as “Intel XTU Benchmark 2.0”
* **\Redistributable**
  + CommonDrivers.msm: This is the merge module which can be used to install all common device drivers that the XTU SDKs rely on. This merge module contains drivers to support Windows 10 on IvyBridge up to the latest platform.
  + CommonDriverEssentials.msm: As CommonDrivers.msm, but does not install the XTU ACPI Driver. The OC application developer should consider to use this merge module if their OC application does not expose BCLK overclocking and reboot-required overclocking.
  + XtuCore.msm: No longer exists. All functions and files from XtuCore have been moved in to CommonDrivers.msm and CommonDriversEssentials.msm. Any installer project that referenced both XtuCore and one of the CommonDrivers msms now only needs to reference one of the CommonDriver msms to get the same functionality.
  + \Drivers: installs the loose driver files for INF installation. All drivers in this folder are Declarative and Componentized drivers with the latest Microsoft OS certification. While any customers building a desktop application may be able to use the .msm files provided that handle the installation of the drivers, any customers who intend to support Universal Windows Application models (as part of DCHU compliance) will need to deploy drivers via Windows Update and INF installation.
    - IOC Extension Driver: is the XTU device extension to the processor. The driver is required for XTU and the SDK to operate successfully. This driver is supported only for Windows 10 RS3 and later. It is serviced by Intel and will not be resold to customers.
    - IOC Component Driver: is the XTU component that installs the core of the XTU service and SDK to the extension device created in the previous step. It is important to install the extension first before the component. The driver is required for XTU and the SDK to operate successfully. This driver is supported only for Windows 10 RS3 and later. It is serviced by Intel and will not be resold to customers.
    - XTU ACPI Driver: installs against the XTU ACPI device for setting BIOS controls and displaying certain ACPI monitors. This is an optional driver for customers implementing the XTU BIOS interface and will be available for resale to customers who request it.
    - Watchdog Timer Driver: installs against the Watchdog Timer Device in ACPI. This is an optional driver for customers implementing the Watchdog Timer in BIOS. It is available for resale to customers who request it.
* **\Document**
  + XTU SDK Programmer’s Guide.pdf (this document)
* **\Sample Application**
  + SampleApplication.zip: the sample application demonstrates how a third party application (.NET or native applications) can use the SDK to tune or monitor the system performance.

# XTU SDK Integration

The XTU SDK is developed based on Microsoft .NET 3.5 or later .NET framework. This section shows how it can be installed on the development machine and used in a .NET or a native application.

## Integration With a .NET Application

Here are steps to add a reference to the XTU SDK assembly in a sample application.

* Create a solution using .NET framework (e.g. SampleApplication)
* Right click on **References** in **Solution Explorer**.
* Select **Add Reference** (Figure 1‑2). The reference manager shows up.

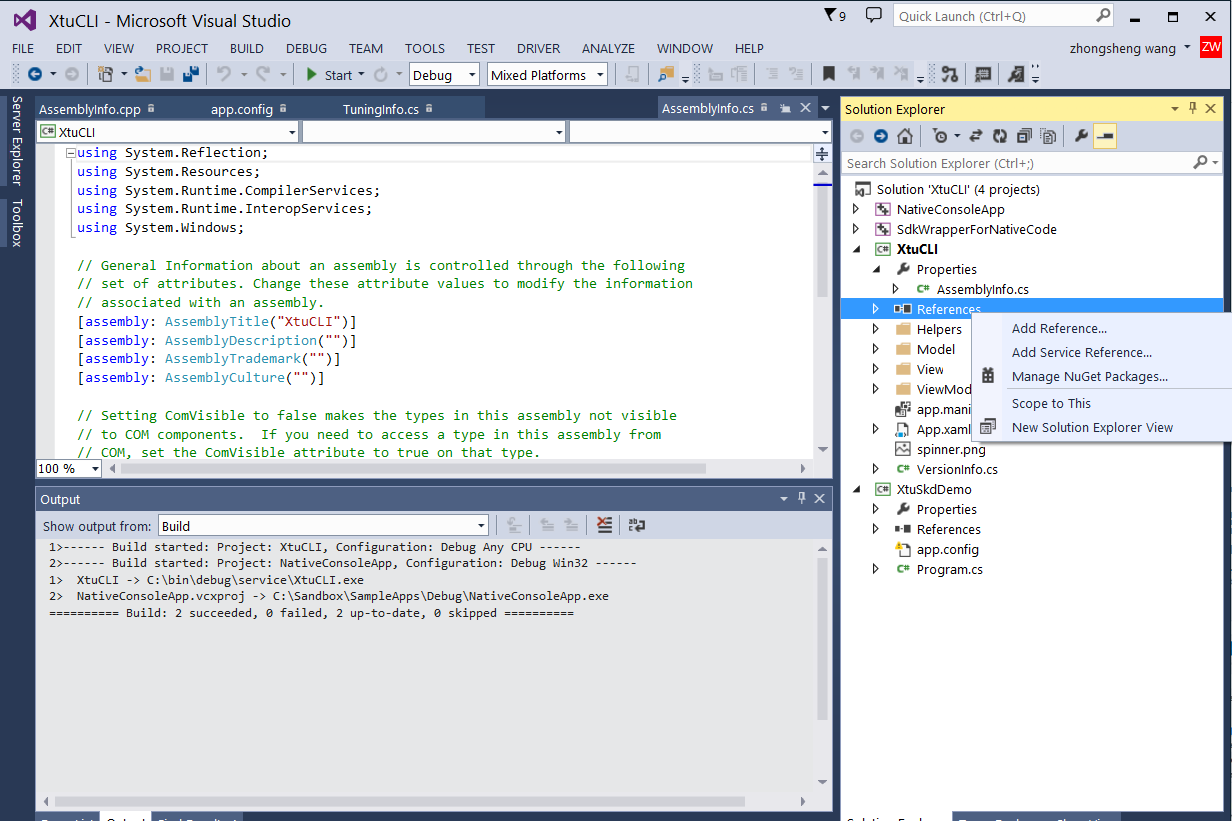


Figure 1‑2 Add Reference to Intel XTU SDK

* Click on **Browse** in the left navigation panel in **Reference Manager** (Figure 1‑3)
* Click on **Browse…** button at the bottom of the **Reference Manager.** Delevoper needs to let the Visual Studio know where the assembly is located on the sytem.The XTU SDK by default is saved in **C:\Program Files (x86)\Intel\Intel(R) Extreme Tuning Utility\SDK\Binary** folder. However, it is recommended that developer copies the assembly and other binaries in the folder to anywhere close to the workspace. The SDK binaries need to be packaged along with the application.
* Click on the checkbox next to IntelOverclockSdk.dll in the Reference Manager.
* Click OK button to close the Reference Manager
* Make sure the IntelOverclockingSdk Reference is added to the solution (Figure 1‑4).

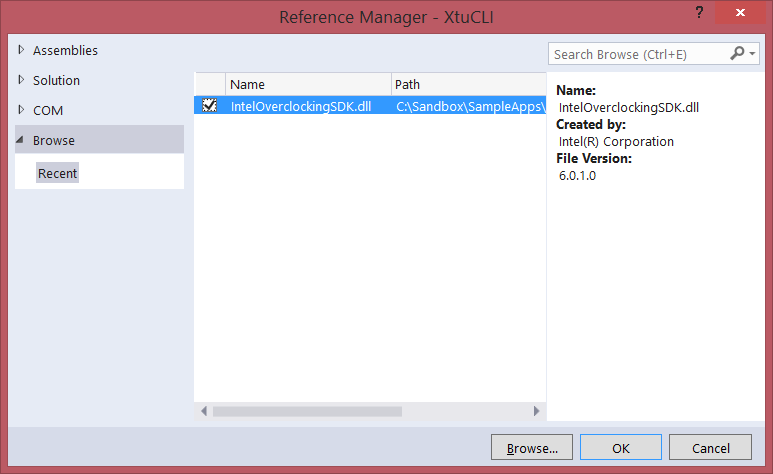


Figure 1‑3 Browse to Intel XTU SDK Assembly

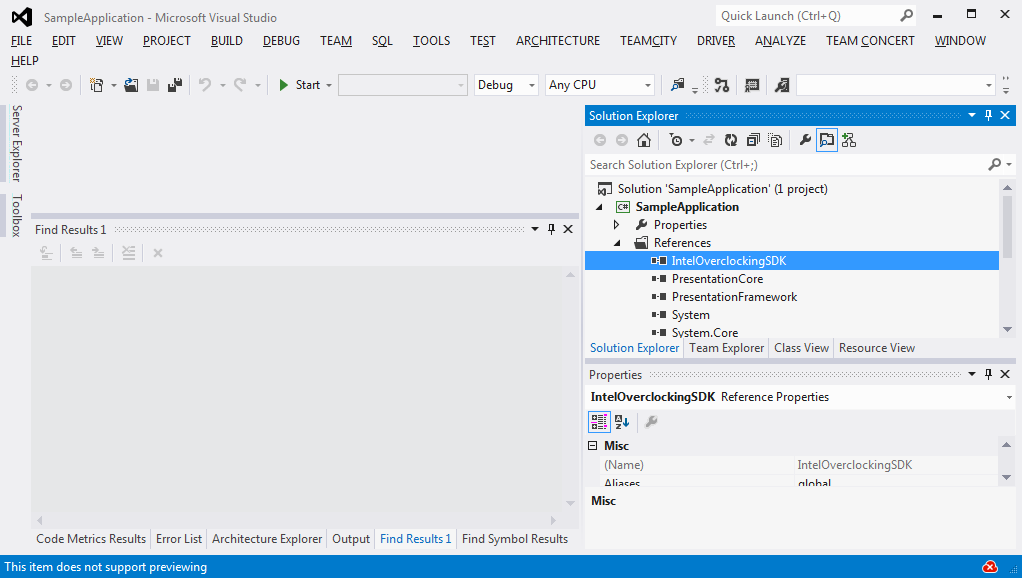


Figure 1‑4 IntelOverclockingSdk Reference Added to the Solution

## Integration With a Native Application

The native application can integrate the SDK using the C++/CLI wrapper. The sample code is provided for this as the part of the SDK installation.

The integration with the native application needs to be done in two steps. The first step is to create a wrapper library to bridge between the native C/C++ application and the managed .NET DLL.

### SDK Wrapper DLL for Native Application

* Create a new project in the solution like Figure 1‑5

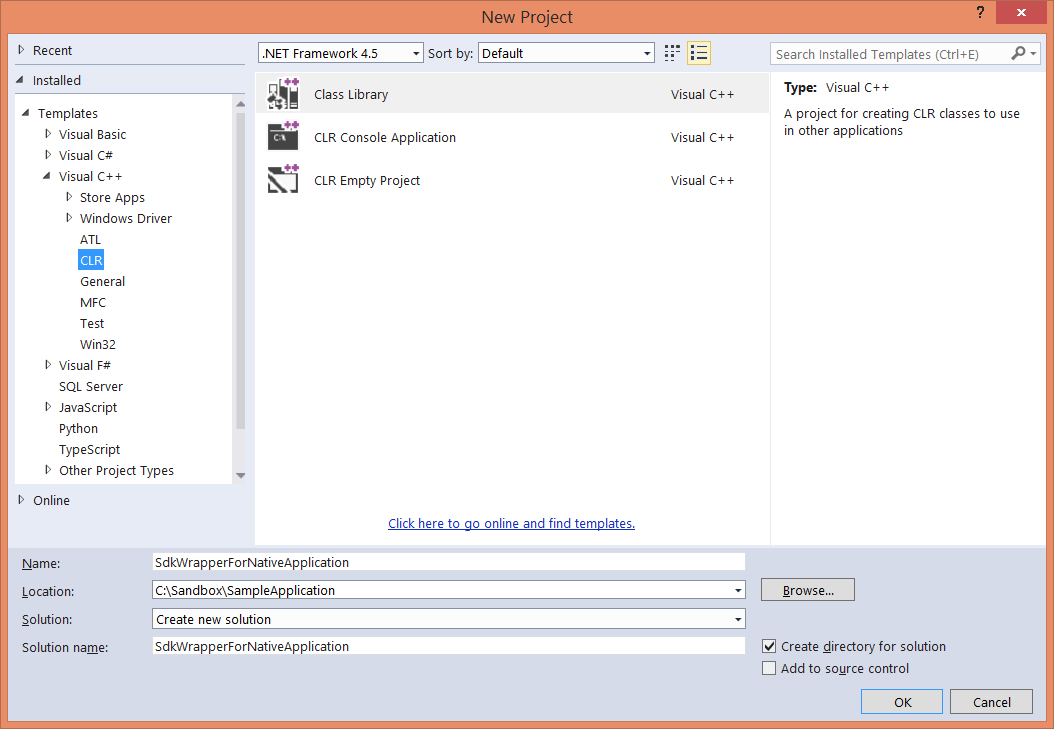


Figure 1‑5 Create a SDK Wrapper DLL for Native Application

* The project property should look like Figure 1‑6

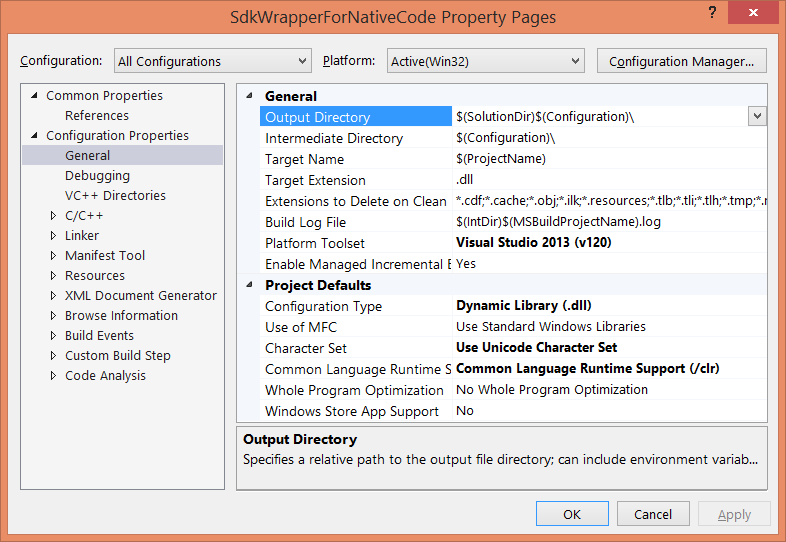


Figure 1‑6 SDK Wrapper DLL Project Property

* Click on **Common Properties** in the Project Property page as shown in Figure 1‑6
* Click on **Add New Reference …** button at the bottom of the page. The **Add Reference** dialog pops up.

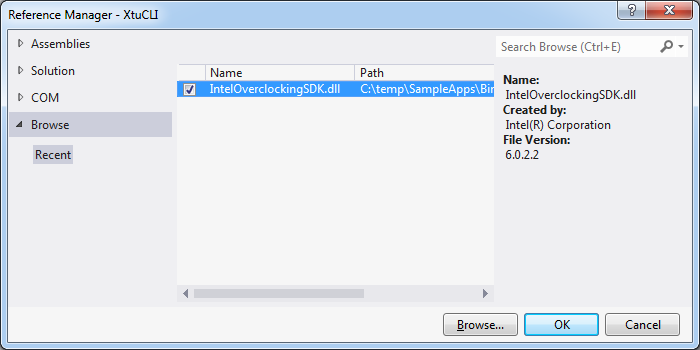


Figure 1‑7 Add Reference to IntelOverclockingSDK DLL

* Click on **Browse…** button at the bottom of the **Reference Manager.** Delevoper needs to let the Visual Studio know where the assembly is located on the sytem.The XTU SDK by default is saved in **C:\Program Files (x86)\Intel\Intel(R) Extreme Tuning Utility\SDK\Binary** folder. However, it is recommended that developer copies the assembly and other binaries in the folder to anywhere close to the workspace. The SDK binaries need to be packaged along with the application.
* After the reference is added, the Common Properties page will look like Figure 1‑8.

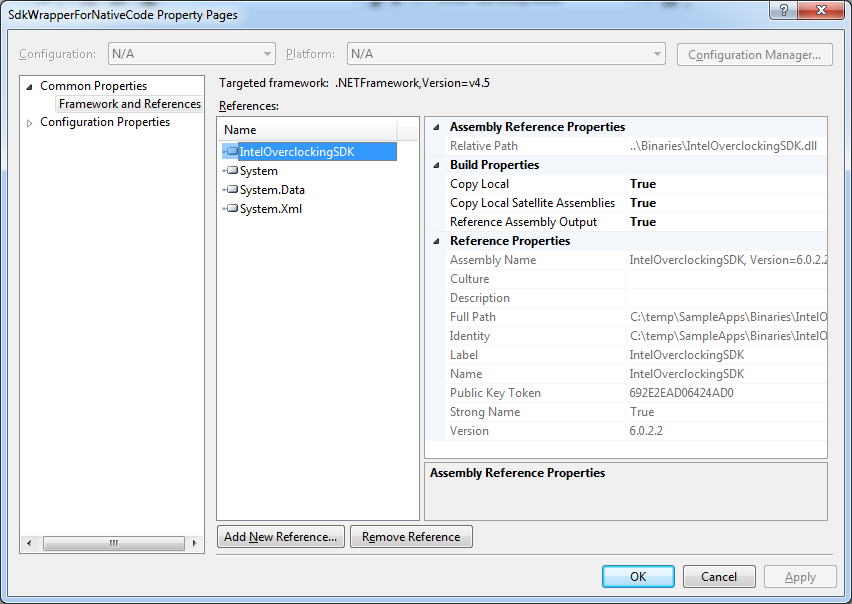


Figure 1‑8 Reference is added to SDK Wrapper Project

* Inside the **Solution Explorer**, the IntelOverclockingSDK.DLL is listed in **External Dependencies** folder (Figure 1‑9).
* Create a class to represent the IntelOverclockingSDK. This class will be used to wrap the IntelOverclockingSDK APIs, for example, the SdkWrapperForNativeCode class as shown in Figure 1‑9. The next several paragraphs are based on this SdkWrapperForNativeCode example. You can certainly create your own project with different names.

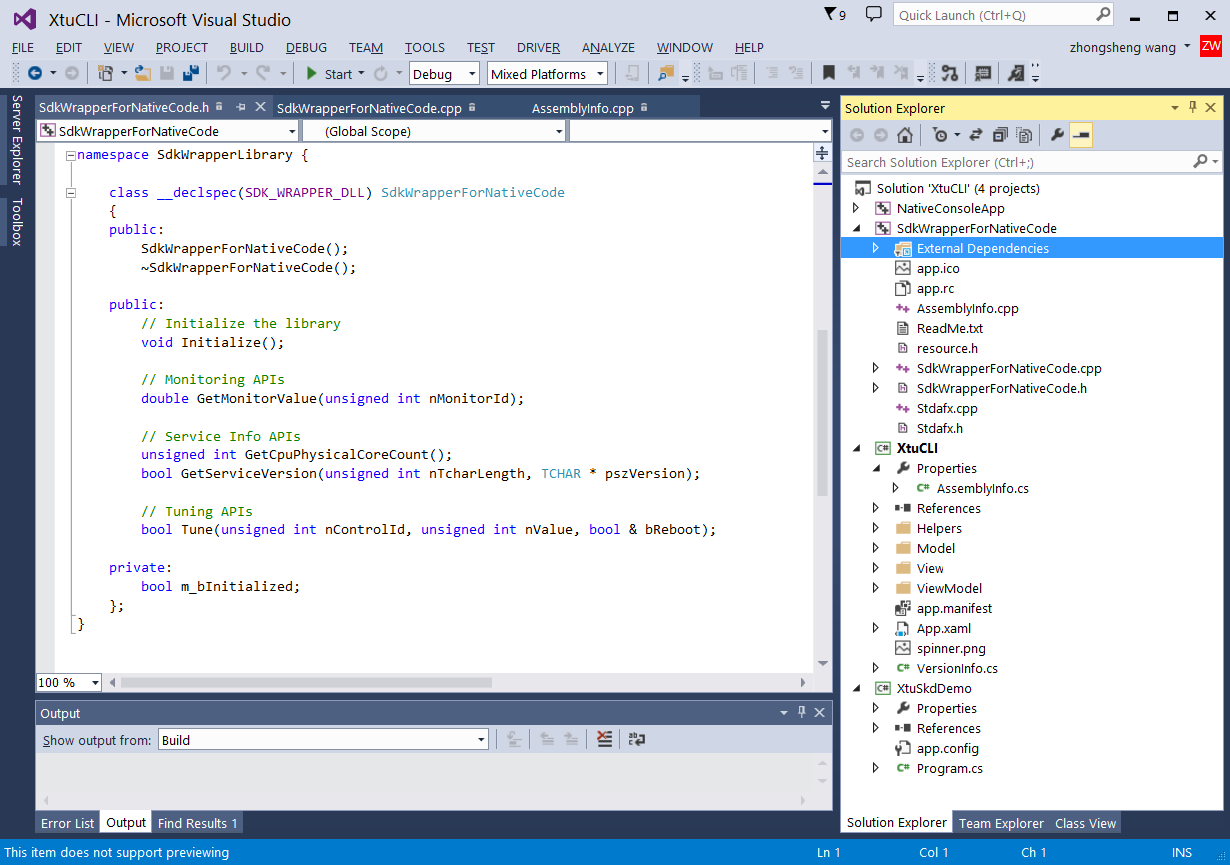


Figure 1‑9 SDK Wrapper For Native Application Project

* In managed environment, you can only define CLR objects within function scope. Therefore, you need a reference class to hold the global handle for the IntelOverclocking.DLL (Figure 1‑10). Note this is necessary only if the IntelOverclockingSDK library is not static.

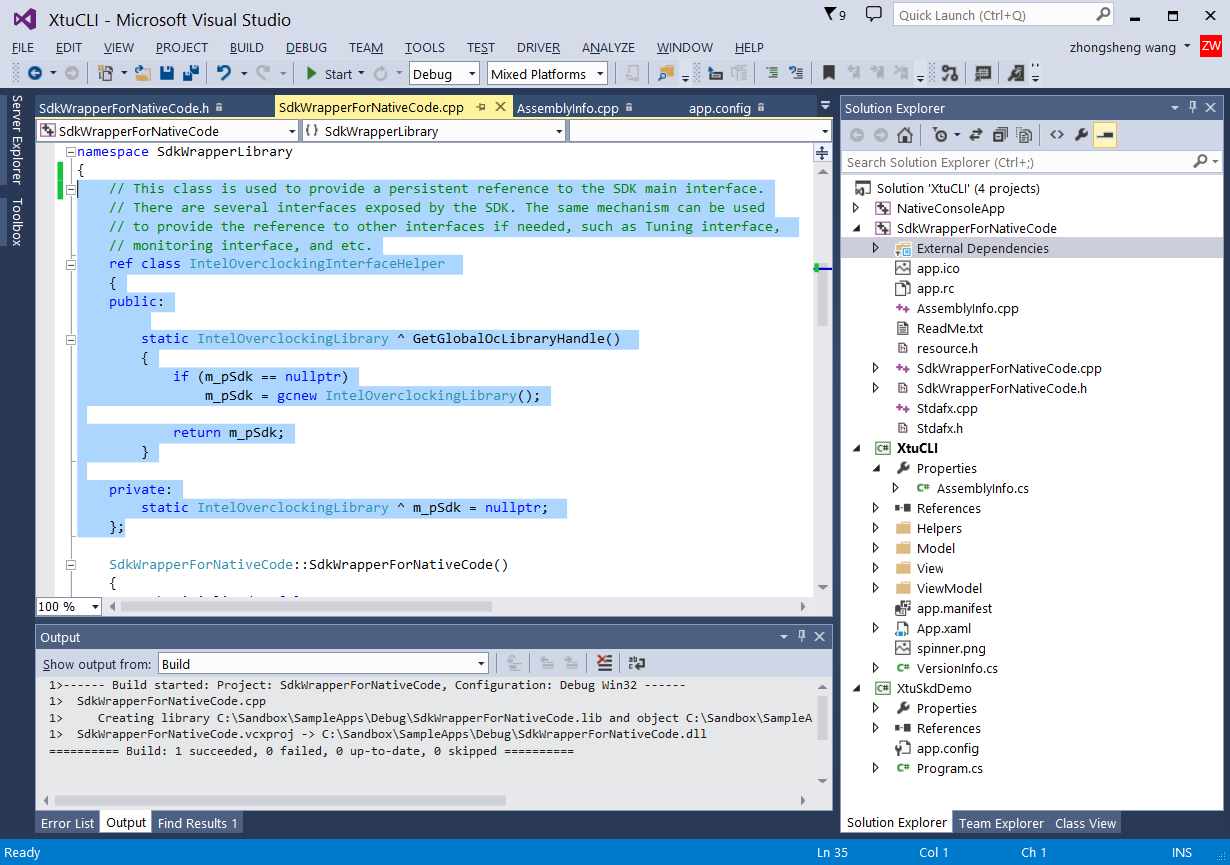
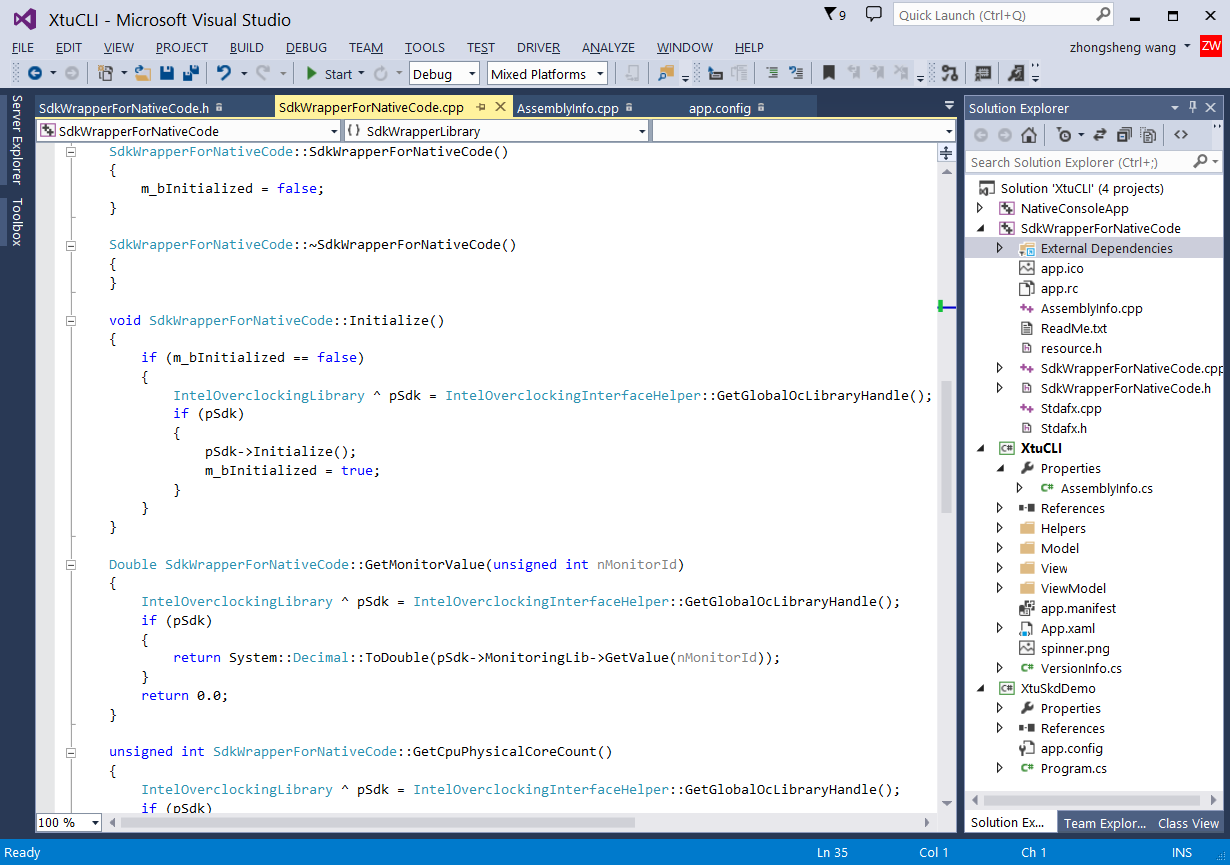


Figure 1‑10 Global Handle to IntelOverclockingSDK

* The wrapper API will call IntelOverclockingInterfaceHelper::GetGlobalOcLibraryHandle() to get the global reference to IntelOverclockingSDK.



* After the wrapper DLL is created, you can start to create the native application for IntelOverclockingSDK.

### Native Application Development

* Create a Win32 project, such as NativeConsoleApp in the sample application in SDK package.
* Add the path to *SdkWrapperForNativeCode.h* to **Additional Include Directories**

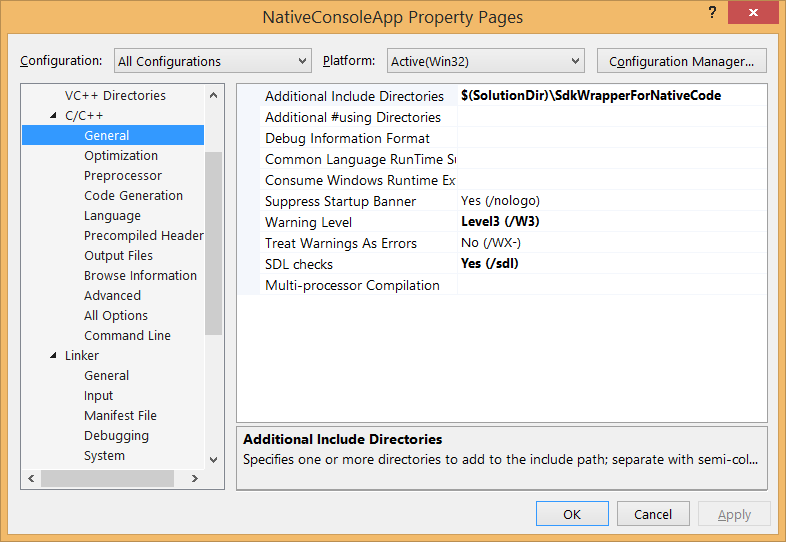


Figure 1‑11 Add Include Path to Native Application

* Add SdkWrapperForNativeCode.lib to the Linker->Input **Additional Dependencies**

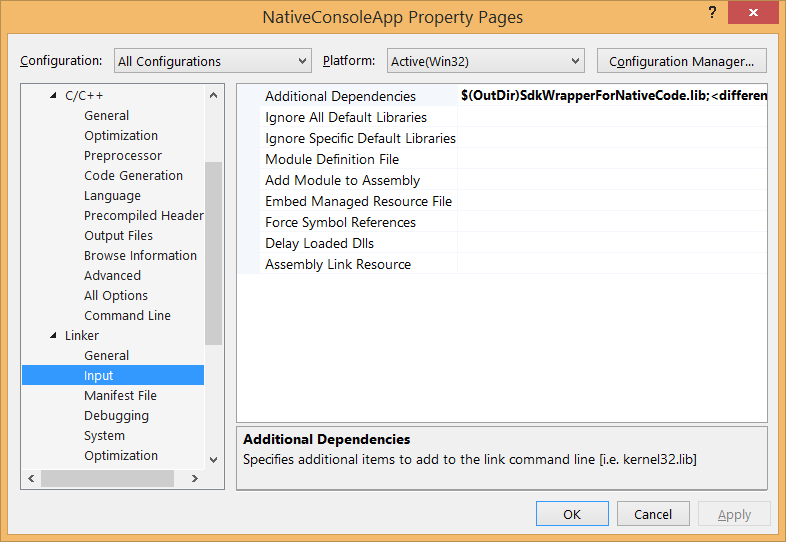


Figure 1‑12 Add Additional Dependencies to Native Application

* Now you are ready to talk to IntelOverclockingSDK

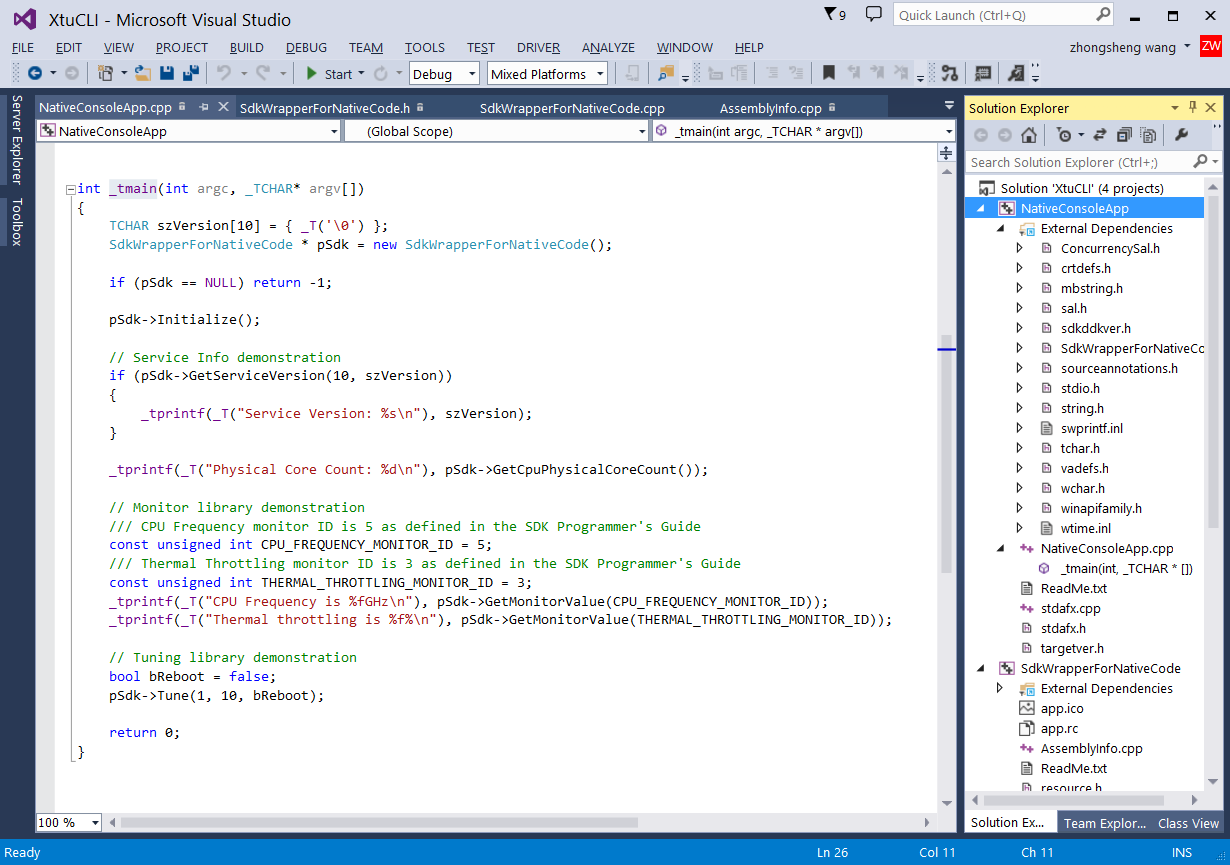


Figure 1‑13 NativeConsoleApp Sample Code

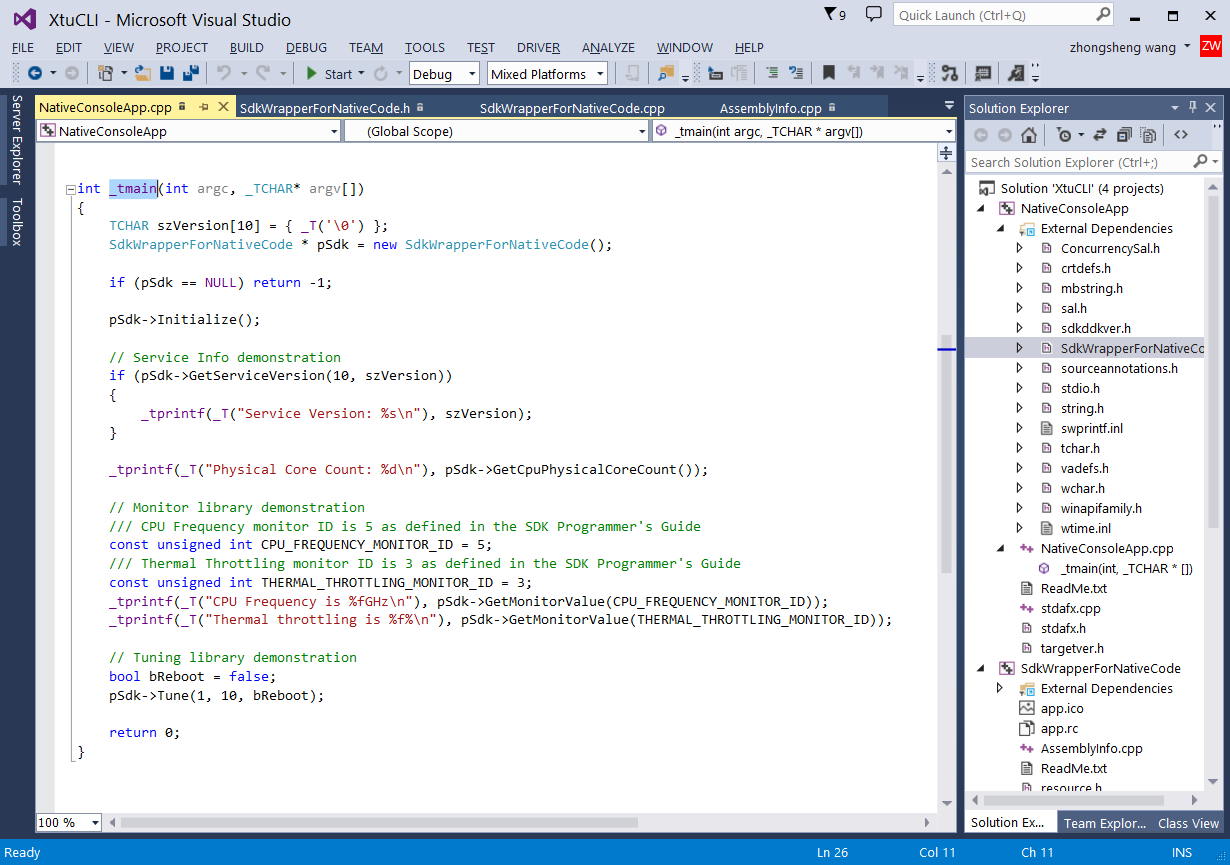


Figure 1‑14 SdkWrapperForNativeCode.dll is listed as an External Dependency

### Debugging SdkWrapperForNativeCode.DLL

The best way to debug SdkWrapperForNativeCode.dll is to set the SdkWrapperForNativeCode project as the startup project and then configure the Command for debugging to your application. For example, the Debugging Command is pointed to NativeConsoleApp.exe for the sample application.

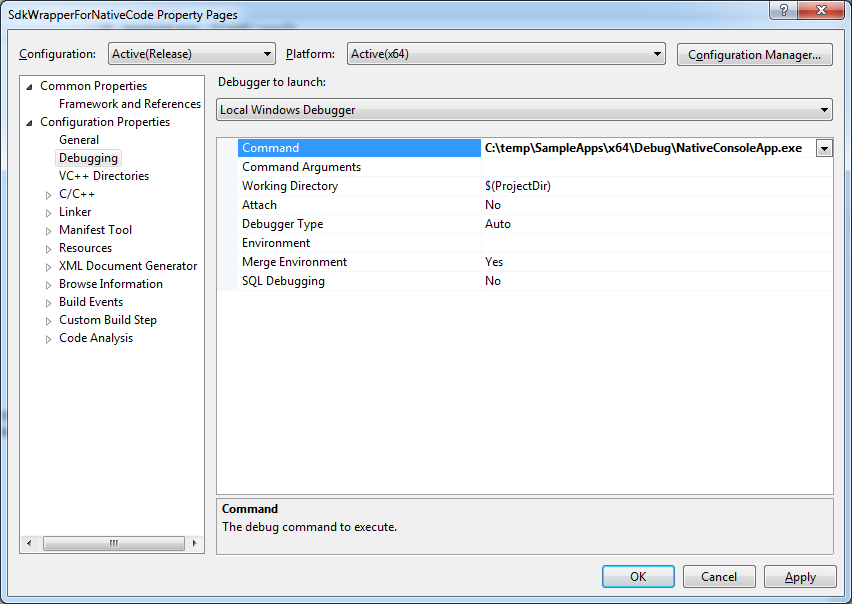


Figure 1‑15 Configure Debugging Command for SdkWrapperForNativeCode.DLL to NativeConsoleApp.exe

# OC Application Deployment

## Merge Modules

The SDK installer has provided several merge modules for convenience. Not all merge modules are required for the application distribution. The following paragraph describes the requirement for the merge module redistribution.

* Common Device Driver Installation

There are 2 merge modules (CommonDriver.msm and CommonDriverEssentials.msm) for the common device driver installation. Only one of them is needed for any application. The Table 1 describes the usage condition for each merge module for driver installation.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Essential (Windows 10 OC Drivers) | XTU BIOS Support | BCLK Overclocking Support | Windows 7 Support | Prior to 6th Generation Platform Support |
| CommonDriver.msm | X | X | X | X | X |
| CommonDriverEssentials.msm | X |  |  | X | X |

Table 5‑1 Common Driver Merge Module Usage Requirement

In summary, if the application does not support BCLK overclocking and XTU BIOS ACPI interface, the CommonDriverEssentials.msm should be used. If none of the above condition is met, the CommonDriver.msm must be used.

If merge modules do not meet the need, refer to Section 5 for customization guidelines.

## Runtime Libraries

The installer must include all runtime libraries that the application has referenced to. However, depending on the functionality of the application, some of runtime libraries may not have to be redistributed. The Table 2 describes the usage condition for each runtime library.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Module | Tuning | Monitoring | Profile Management | App-Profile Pairing | Benchmarking and HWBOT Submission |
| IntelOverclockingSDK.dll | X | X | X | X | X |
| ProfileHelperModel.dll | X | X | X | X | X |
| IntelBenchmarkSDK.dll |  |  |  |  | X |

Table 5‑2 Overclocking SDK Runtime Library Redistribution

In summary, if the application does not support benchmarking operation, the application installer does not have to include the IntelBenchmarkSDK.dll. The runtime libraries IntelOverclockingSDK.dll and ProfileHelpderModel.dll must be included in the application distribution all the time.

Besides the runtime libraries that XTU SDK have provided, the VC++ runtime library may be needed in certain cases. Here are some download links for Microsoft Visual Studio C++ Redistributables.

* Visual Studio 2010 C++ Redistributable (x86): <https://www.microsoft.com/en-us/download/details.aspx?id=8328>.
* Visual Studio 2012 C++ Redistributable: <http://www.microsoft.com/en-us/download/details.aspx?id=30679>
* Visual Studio 2013 C++ Redistributable: <http://www.microsoft.com/en-us/download/details.aspx?id=40784>

## Benchmark Executables

The XTU SDK comes with multiple benchmarking applications that can be run exclusively through the IBenchmarkLibrary. These executables are in the \Binary directory at the SDK install path.

|  |  |
| --- | --- |
| BenchmarkId | Executable Name |
| IntelXtuBenchmark | p95bench.exe |
| IntelXtuBenchmark2 | IntelXTUBenchmark2.exe |

In order to run one of these benchmarks through the IBenchmarkLibrary, the corresponding executable must be deployed in a folder named “Binaries” located as a sibling directory to the directory that contains the executable that is calling IBenchmarkLibrary.

## Coexistence with other overclocking applications

It is very likely that multiple XTU SDK based applications are running on the same platform. The most common example is the Intel® Extreme Tuning Utility and a dedicated overclocking application for the OEM. Typically OEM pre-installs its own overclocking application on the system. User can download the XTU application from [Intel Download Center](https://downloadcenter.intel.com/search?keyword=extreme+tuning) and install the application on any OEM platform that supports overclocking. Since both applications could rely on the same XTU service and drivers, the coexistence becomes very important.

By design, all XTU SDK based applications can coexist, which means one application should not cause another to crash or malfunctional. In an addition, when one application is uninstalled or reinstalled, there should have no visible impact on any other applications. Even though all these requirements are validated by Intel before the software release, OEM should make sure the above behavior is met before the platform is released to customer.

## Overclocking Warning

The following warning message must be presented to the end user if the Intel Extreme Tuning Unility SDK is used in the application. This warning message could be copied to the EULA or presented to the user separately.

WARNING: Altering clock frequency and/or voltage

may: (i) reduce system stability and useful life of the

system and processor; (ii) cause the processor and

other system components to fail; (iii) cause

reductions in system performance; (iv) cause

additional heat or other damage; and (v) affect

system data integrity. Intel has not tested, and does

not warranty, the operation of the processor beyond

its specifications. Intel assumes no responsibility that

the processor, including if used with altered clock

frequencies and/or voltages, will be fit for any

particular purpose. For more information, visit

<http://www.intel.com/consumer/game/gaming-power.htm>

## 64bit Support

The same XTU SDK is configured to support both 32bit and 64bit applications. The sample application has already provided configurations for both 32bit and 64bit managed and unmanaged applications.

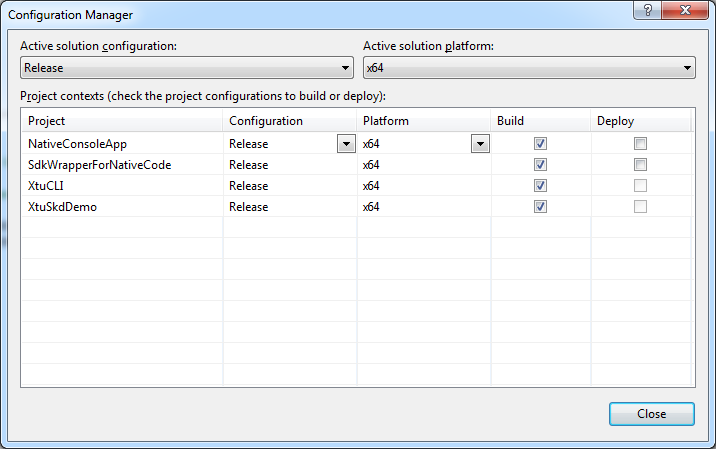


Figure 1‑16 64bit Configuration for Sample Applications

# XTU SDK Debugging

## Local Debug

In order to test the XTU SDK, the XTU core service and drivers must be running. However, the XTU SDK installation does not install the XTU core service and XTU common driver on the development machine. There are two ways to mitigate this issue:

* Develop your own installer to install the XTU core service and XTU common driver as described in the next section.
* Install the XTU application with the same version as the XTU SDK. By default, the XTU application will be installed in C:\Program Files (x86)\Intel\Intel(R) Extreme Tuning Utility. The easiest way is to copy your application binaries to C:\Program Files (x86)\Intel\Intel(R) Extreme Tuning Utility\Client folder and then run your application from there.

## Remote Debug

Remote debugging is also possible with this setup. As long as the XTU core service and XTU common drivers are installed and running on the target machine, you should be able to attach to the target machine and run remote debugging process to debug your application.

## SDK Log

By default, XTU SDK generates log messages. If XTU UI is launched, the XTU SDK message will be part of XTU GUI log. The best way to see XTU SDK log message is to launch DebugView (<https://technet.microsoft.com/en-us/Library/bb896647.aspx>). The XTU SDK log message will be visible in DebugView tool.

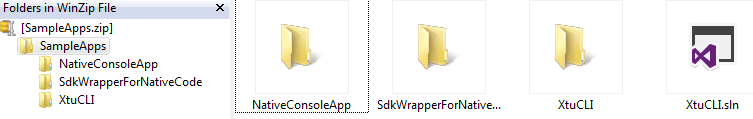
# XTU SDK Sample Code

## Sample Application Package

The sample code is provided as part of the SDK package. When the SDK is installed, the sample code is installed at the following location by default:

C:\Program Files (x86)\Intel\Intel(R) Extreme Tuning Utility\SDK\SampleApplication

The sample code is provided as a zip file. The layout looks like this in the compressed format.



The folder XtuCLI contains the .NET application sample code. The folder SdkWrapperForNativeCode contains the C++/CLI wrapper library for the native code. The folder NativeConsoleApp is the sample native console application using the C++/CLI wrapper library.

## XtuCLI Demo Application

The XtuCLI demo application not only shows how the IntelOverclockingSdk can be integrated with a custom application, it also can be used as a validation tool. This tool can run either as a GUI application or a console application. If there is no command line option, it will show GUI. Otherwise, it will behave like a console application.

In GUI form, the application can be used to tune or monitor a single control manually.

In Console form, the application can be used to dump all available controls on the platform. It can be used to automate the tuning process as well.

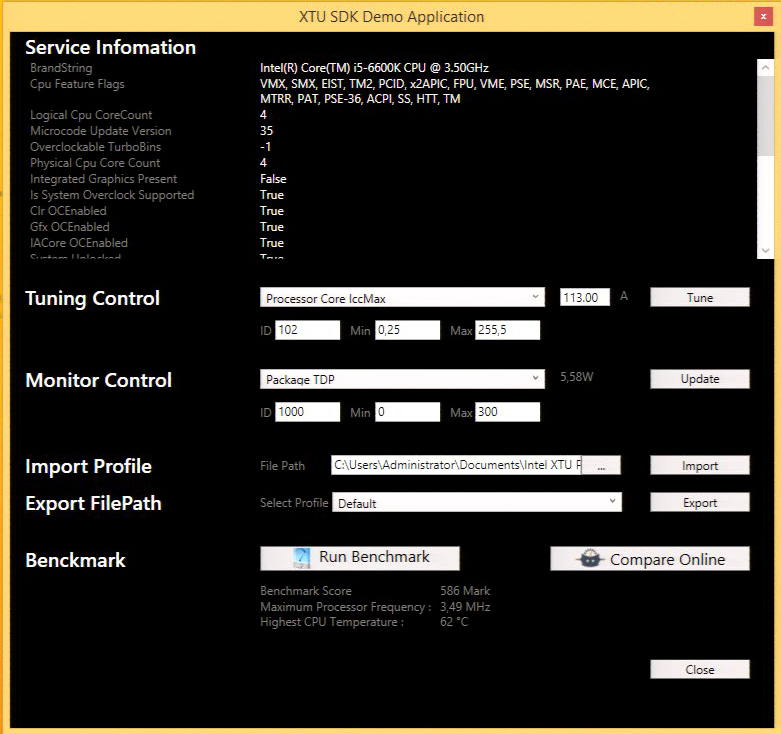


Figure 4‑1 XTU SDK Demo Application UI

The dropdown list contains all available tuning and monitoring controls.User can select one item at a time to set or update a control. The unit for the control value should also be display properly. If reboot is required, the status message must indicates that system will reboot automatically when Tune button is clicked.

If reboot is required, clicking on Tune button will automatically reboot the system.

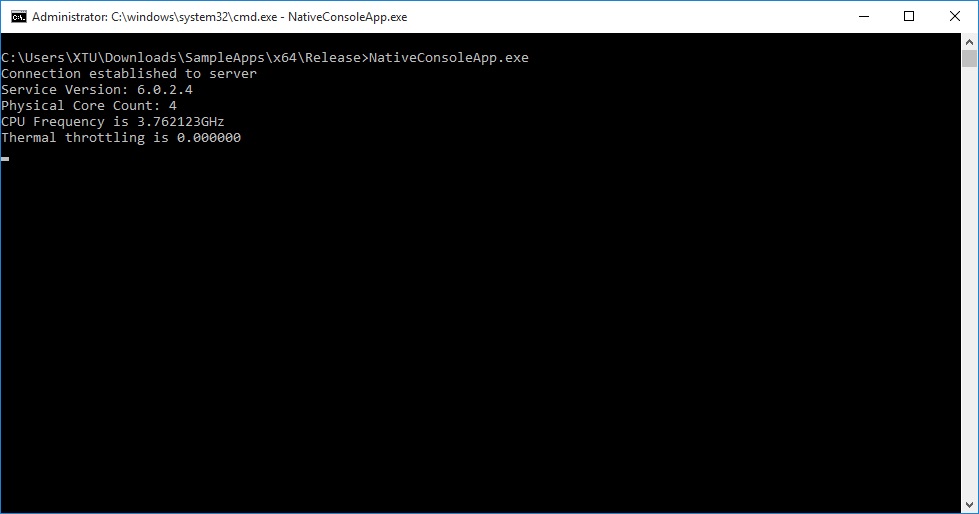
The UI sample application should take command line input. When the command line option is present, UI should be hidden. The preliminary command options are listed below. The command line option is not case sensitive.

|  |  |  |
| --- | --- | --- |
| Option | Arguments | Comments |
| -i or -info | <System>: dump out CPU info, watchdog timer, and XTU service info to the console.  <Tuning>: dump out all tuning controls as a list of data structures  <Monitoring>: dumpt out all monitoring controls as a list of data structures  <All>: dump all information. | The default argument is <All> if argument is missing.  For example, XtuCli.exe –i All    The format of output should be like this. A single line will be used for each control and each line should have 3 segments: ID, Name, and Value with unit.  For example:  ID Name Value  29 1-Active Core Max Ratio 35  5 Processor Frequency 3.5GHz |
| -t or -tune | -ID <TuningControlId>: Required, must be the ID for one of available tuning controls  -V <TuningControlValue> Required, must be appropriate for the control. | Provide the tuning options for a control. If reboot is required, system will be rebooted by default. User can override this behavior using –f option.  For example, to set 1-Active Core Max Ratio to 40:  XtuCli.exe –t –ID 29 –V 40 |
| -m or –monitor | -ID <MonitorId>: Required, reports the current value for the monitor | Reports the monitoring value for a monitor  For example, XtuCli.exe –m –ID 5 |
| -ip or –import | -Path <complete profile filepath>: Required. | Import a profile and apply the changes to the system. |
| -ep or -export | -ID <ProfileID>: Required,  -Path <complete profile filepath>: Required. | Export a profile to a file. |
| -h or –help | None | Printout the command line options. |

Table 7‑1 XTU SDK Demo Application Command Line Interface Definition

## NativeConsoleApp Application

This sample application demonstrates how the overclocking SDK can be integrated in a native application. The Admin privilege is required in order to run this application properly..



There are two modules for this application.

* SdkWrapperForNativeApp.dll: using managed C++ to wrap the .NET CS APIs for the native C++ client. This module depends on the IntelOverclockingSdk.dll.
* NativeConsoleApp: this is the native executable. It calls the API exposed in SdkWrapperForNativeApp.dll.

# XTU SDK Redistributables Customization Guide

If the merge modules provided by the overclocking SDK installer do not meet your need for some reason, you can customize the module distribution in your application installer. The following paragraph elaborates the best practices for the redistributables customization that Intel would recommend. The recommendation is based on the latest overclocking SDK distribution (6.0.2.4 at the time).

## Merge Module Installer

The first step is to build a Merge Module installer. There are many ways to do this. The simplest method is to use Visual Studio Setup and Deployment project in Visual Studio 2010. Here is the link:

<https://msdn.microsoft.com/en-us/library/8x727h8b%28VS.80%29.aspx?f=255&MSPPError=-2147217396>

Add CommonDriver.msm to the merge module installer.

Once the merge module installer is generated, you can install the installer on any machine. The installer will create the following folder on your machine:

C:\Program Files (x86)\Intel\Intel(R) Extreme Tuning Utility

C:\Program Files (x86)\Intel\Intel(R) Extreme Tuning Utility\Drivers

## Install XTU Service

The XTU service binaries are in C:\Program Files (x86)\Intel\Intel(R) Extreme Tuning Utility folder after installation. All individual files in the folder (not files in the subfolder) are required for the XTU service. On Windows 10 RS3 or later OS, the service is installed automatically by the XTUComponent Driver. On previous OS, one can use this procedure:

To install the XTU service, execute the following command from an elevated shell from the folder:

*XtuService.exe –install or XtuService.exe –i*

To uninstall the XTU service, execute the following command from an elevated shell from the folder:

*XtuService.exe –uninstall or XtuService.exe -u*

To start the XTU service, execute the following command from an elevated shell:

*Net start Xtu3Service*

To stop the XTU service, execute the following command from an elevated shell:

*Net stop Xtu3Service*

## Device Driver Requirement

Not all device drivers are required for overclocking via XTU. The certain modules are needed only if the associated usage is required. The XTU application will continue to be functional as long as the essential modules are available. The following table describes the dependents and their usage.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module | Essential | XTU BIOS | BCLK OC | Comments |
| XTU ACPI Driver |  | X |  | Needed only if BIOS ACPI/INT3394 is enabled in BIOS. |
| WDT Driver | X\* |  |  | Required for crash recovery, however, XTU application will continue to work without this module. |
| IOC Driver | X\* |  |  | Required for runtime overclocking for Windows 7 and Windows 10 RS1 and RS2. |
| IOC Extension Driver | X |  |  | Required for runtime overclocking for Windows 10 RS3 and later. This driver replaces the old IOC Driver. |
| IOC Component Driver | X |  |  | Required for runtime overclocking for Windows 10 RS3 and later. This driver replaces the old IOC Driver. |
|  |  |  |  |  |

Table 8‑1 Overclocking Common Driver Module Usages

You can simplify the deployment if you do not plan to support all features in XTU. For example, if the XTU BIOS interface (ACPI/INT3394) is not enabled on the platform, you don’t have to distribute the XTU ACPI driver.

## Install XTU Drivers

### Folder AcpiDriver

This folder contains the XTU ACPI device driver binaries needed only if the XTU ACPI device (DID: ACPI\INT3394) is enabled in BIOS. This driver is Declarative and Componentized based on Microsoft’s DCHU model and is available for resale on customer request.

To install the XTU ACPI device driver, execute the following command from an elevated shell from the folder:

*Dpint.exe /SA /SW /Path “C:\Program Files (x86)\Intel\Intel(R) Extreme Tuning Utility\Drivers\AcpiDriver\64bit”*

The tool DPInst.exe is available from Windows Driver Kits. For convenience, these tools are included in the XTU SDK installer.

### Folder IccWdtDriver

This folder contains the OC Watchdog Timer driver installation package. This is required for crash recovery at runtime. When this driver is available, the XTU core service arms the watchdog timer when any overclocking occurs. If system crashes during the tuning operation, the BIOS will recover the system from the crash. The hardware settings will be rolled back to default values.

This driver is Declarative and Componentized based on Microsoft’s DCHU model and is available for resale on customer request.

To install the WDT device driver, execute the following command from an elevated shell from the folder:

*Dpint.exe /SA /SW /Path “C:\Program Files (x86)\Intel\Intel(R) Extreme Tuning Utility\Drivers\IccWdtDriver”*

### Folder IocDriver

This folder contains the XTU legacy driver. This is required for the XTU core service. This is a pure software driver. This driver is only supported for Windows 7 through Windows 10 RS2 and is not ‘Declarative’ and ‘Componentized’ based on Microsoft’s DCHU model.

To install IOC driver, execute the following commend from an elevated shell from the folder:

*InstallDriver.exe –s*

To uninstall IOC driver, execute the following commend from an elevated shell from the folder:

*InstallDriver.exe –s -u*

### Folder IocExtensionDriver

This folder contains XTU Extension Device driver. This is required for XTU core operations to work. This driver installs an extension device against the CPU device to allow for runtime overclocking. This driver is only supported for Windows 10 RS3 and later and will be ‘Declarative’ and ‘Componentized’ based on Microsoft’s DCHU model. The driver will be serviced by Intel and is not available for resale.

To install the Extension Device, use the standard INF installation methods or execute the following command from an elevated shell from the folder:

*Pnputil.exe /add-driver XTUExtension.inf /install*

More information about the command line syntax for pnputil.exe can found on Microsoft’s [website](https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/pnputil-command-syntax).

### Folder IocComponentDriver

This folder contains the XTU Component driver and is required for XTU or the SDK to perform any basic operations. This is a pure software driver that installs against the extension device. This driver is only supported for Windows 10 RS3 and later and is ‘Declarative’ and ‘Componentized’ based on Microsoft’s DCHU model. The driver will be serviced by Intel and is not available to customers for resale.

To install the XTU Component driver, be sure to install the Extension device first. Use the standard INF installation methods or execute the following commend from an elevated shell from the folder:

*Pnputil.exe /add-driver XTUComponent.inf /install*

More information about the command line syntax for pnputil.exe can found on Microsoft’s [website](https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/pnputil-command-syntax).

To uninstall the component driver, use standard INF uninstallation methods or execute the following command from an elevated shell from the folder:

*Dpint.exe /SW /U /Path “[Location of XtuComponent.inf]”*