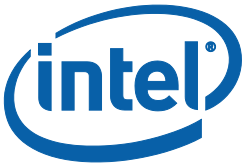


Firmware Update Library

(Windows 32/64 bit)

Intel® 9 Series Chipset Family Intel® Management Engine Firmware 11.6 SKUs

White Paper

May 2016

Revision: 1.0

Intel Confidential

Contents

Revision History

Revision Number	Description	Revision Date
1.0	Initial Release	May 2016

§ §

1 Introduction

The purpose of this document is to describe the Firmware Update Libraries that will be used for Intel® Management Engine (Intel® ME) update. It contains a description of the various APIs to be used. Flow charts will describe the general flow of the library and the functions.

1.1 Terminology

Acronym / Term	Definition
API	Application Programming Interface
FPT	Flash Partition Table
FTP	Fault Tolerant Partition
Full Image	A full image starts with a FPT and contains FTP and NFTP partitions.
Full Update	Updates all the regions
FW	Firmware
FWUpdateLib	Firmware Update Library
Intel® ME	Intel® Management Engine
LOCL	Localization Language
NFTP	Non- Fault Tolerant Partition
OEM ID	Original Equipment Manufacturer Identification Number
Partial Image	A partial image starts with either WCOD or LOCL partitions. NO FPT, FTP, and NFTP in the file
Partial Update	Only updates regions that require an Update such as WCOD or LOCL
Restore Image	Has no FPT but starts with FTP or NFTP
WCOD	Wireless Card Device

§ §

2 Firmware Update Flow

2.1 Purpose

The purpose of the library is to update existing FW image on a platform. The firmware update process is essential for distributing FW images with bug fixes and also updating WCOD & LOCL regions. By utilizing the APIs provided in the Firmware Update Library a program can update the existing FW image on a platform. The library sends the new FW image to the Intel® Management Engine (Intel® ME) FW. Intel ME FW updates the flash device with the new FW image.

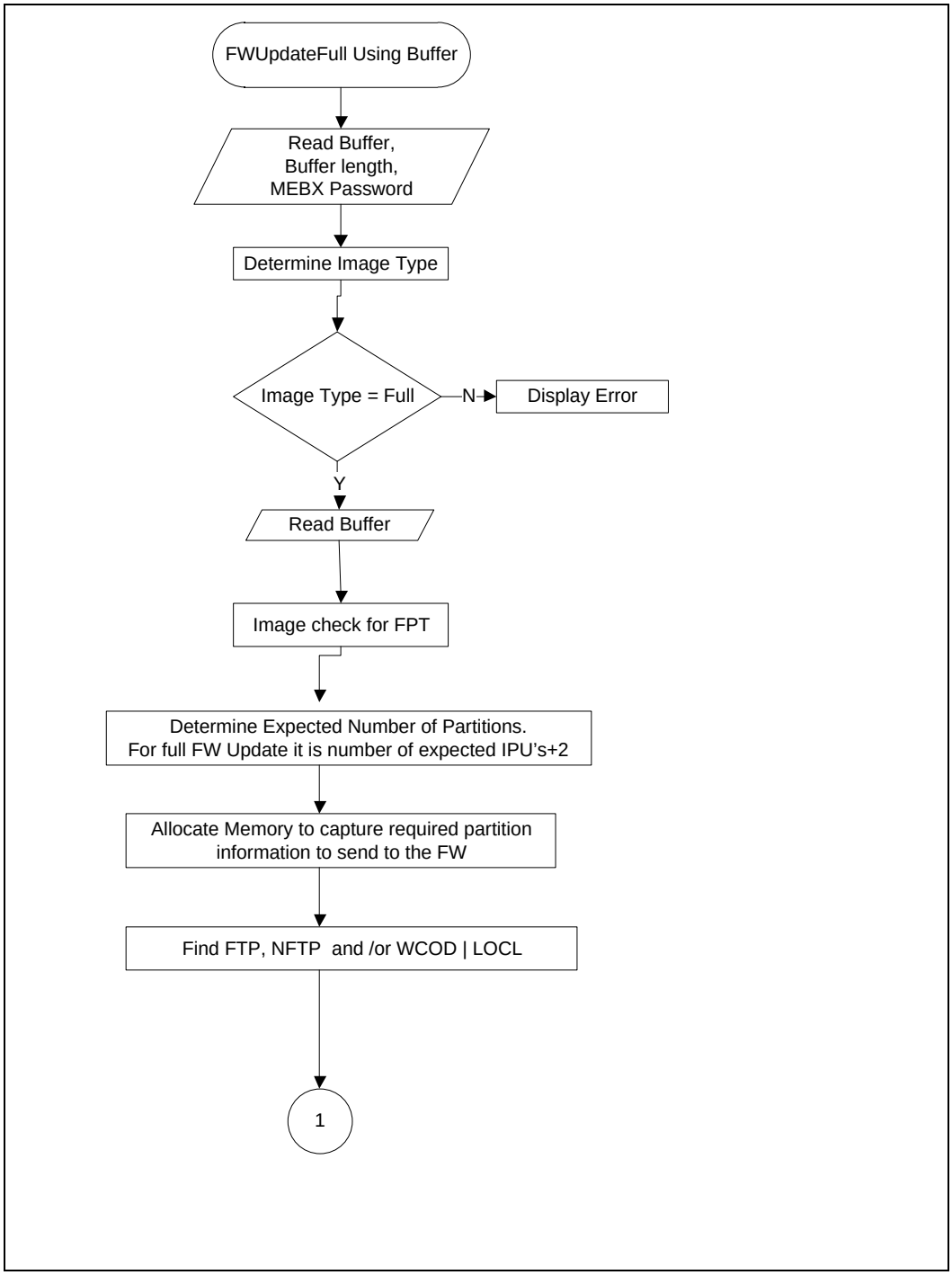
2.2 Supported Operating System

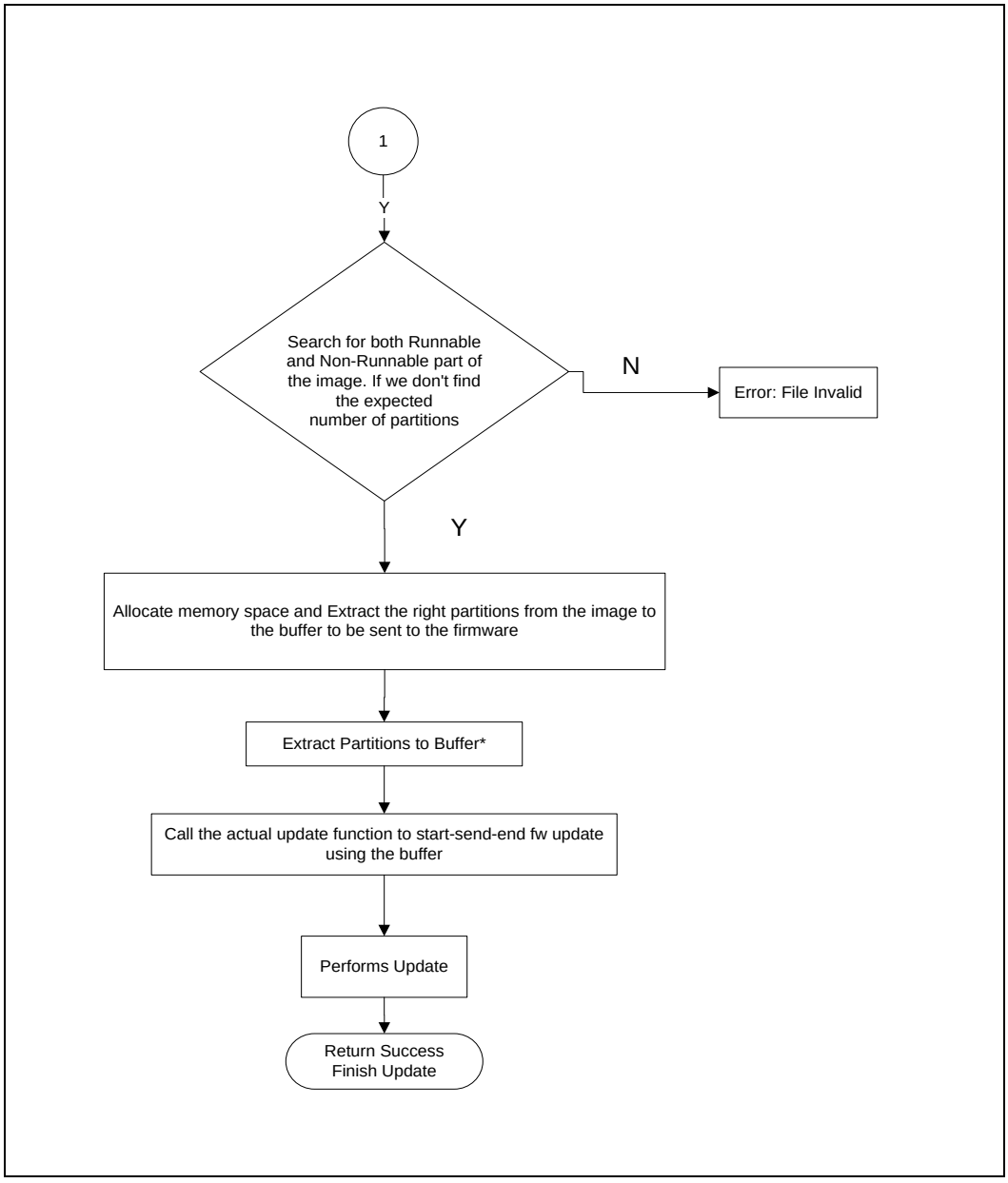
Windows* 7 32/64b and Windows* 8 32/64b

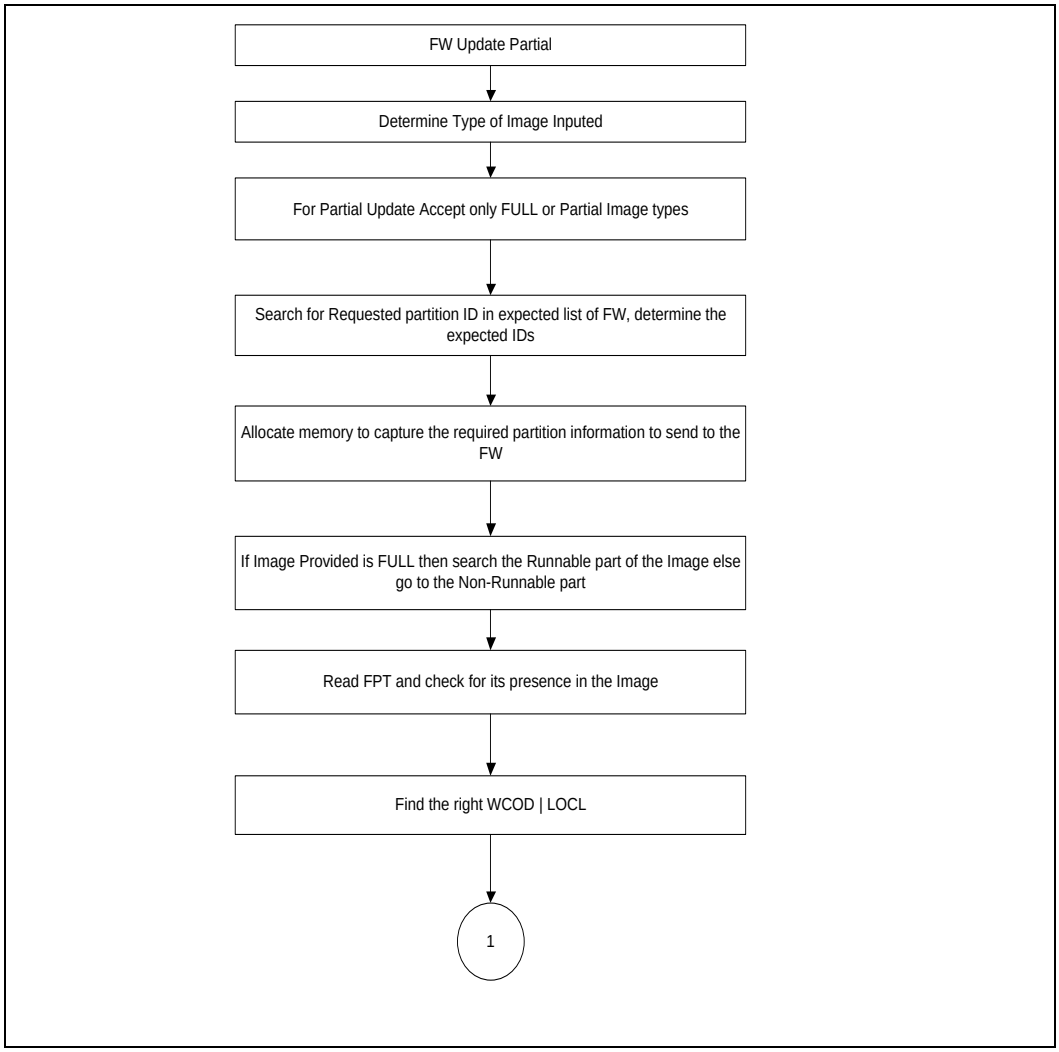
2.3 FW Update Library Description

The following section describes a high level overview of the FW Update process:

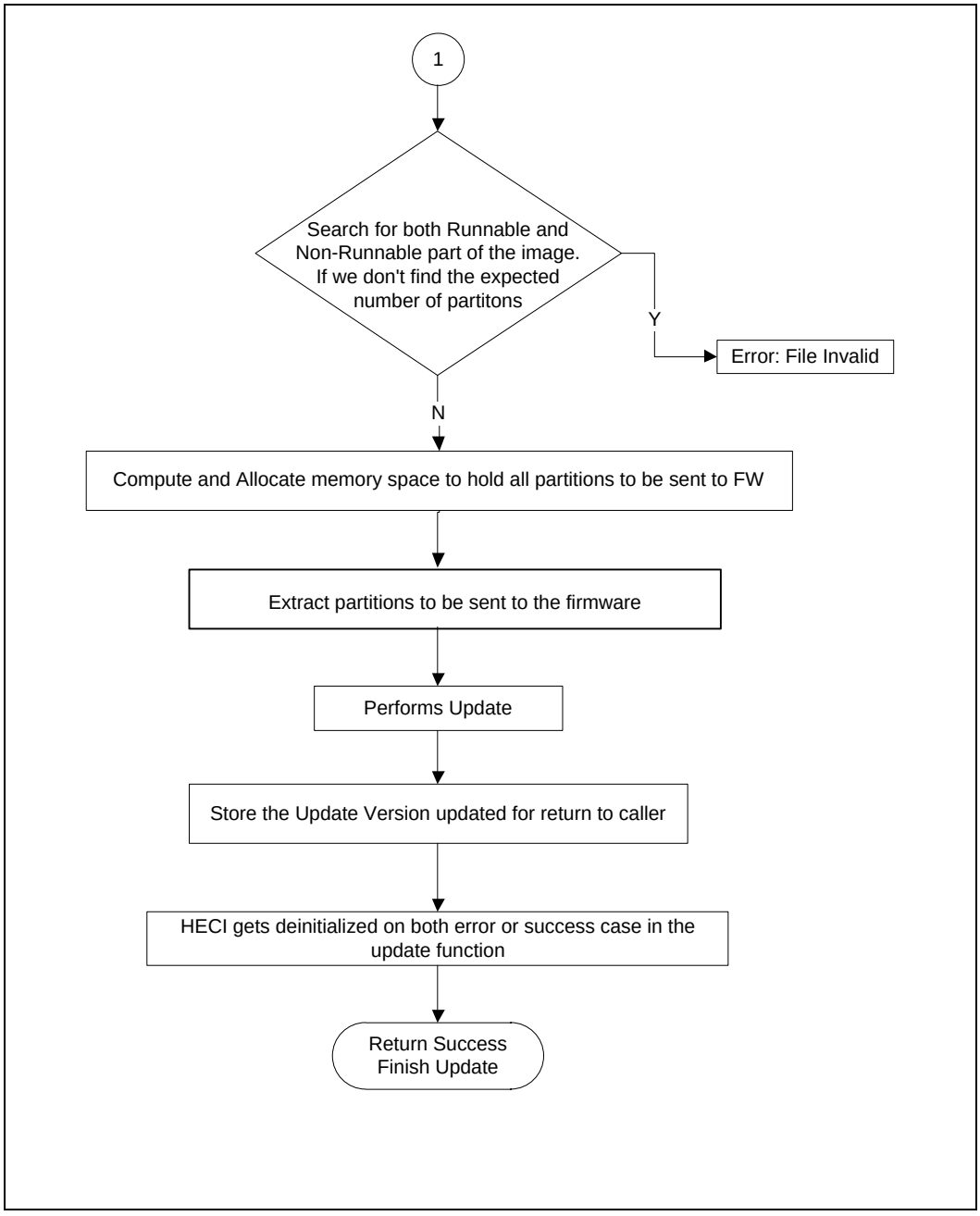
2.3.1 Full Firmware Update Flow




1
Y
Search for both Runnable and Non-Runnable part of the image. If we don't find the expected number of partitions
N
Error: File Invalid
Y
Allocate memory space and Extract the right partitions from the image to the buffer to be sent to the firmware
Extract Partitions to Buffer*
Call the actual update function to start-send-end fw update using the buffer
Performs Update
Return Success
Finish Update



2.3.2 Partial Firmware Update


1
Search for both Runnable and Non-Runnable part of the image. If we don't find the expected number of partitons
Y
Error: File Invalid
N
Compute and Allocate memory space to hold all partitions to be sent to FW
Extract partitions to be sent to the firmware
Performs Update
Store the Update Version updated for return to caller
HECI gets deinitialized on both error or success case in the update function
Return Success
Finish Update

§ §

3 Getting Started – FWUpdate Library

3.1 Environment

The FWUpdate Library provided is compiled using Microsoft Visual Studio 2010.

3.2 Setup

Follow the setting of the references below to get started with using the Firmware Update (FWUpdate) library and integrating it to the application correctly.

1. When building the application and integrating FWUpdate library compiler must know the location of the library and header files.

2. Referencing header files: To reference the header files of the static library, you must modify the include directories path. To do this, from the **Property Pages** dialog, expand the **Configuration Properties** node, then the **C/C++** node, and select **General**. Next to **Additional Include Directories**, type in the path to the location of the FWUpdateLib.h header file.

3. Linking to Static Library: Within Visual Studio, right click on your project in the Solution Explorer and bring up the project's **Properties** dialog. Expand the **Configuration Properties** node, then select the **Linker** category of properties and, pick the **Input** option under that. In the **Additional Dependencies** property, add the library FWUpdateLib.lib

4. You will also need to reference the “FWUpdateLib.h” file in your program.

Note: Please select the appropriate Libraries and header files when building either 32 bit or 64 bit application.

3.3 Security Consideration

For Windows* 7 and 8 (both 32 and 64 bit), the account must have been created with Administrator rights.

Since the library accesses restricted parts of the memory and hardware, it must be run in a window with Administrator Privileges. This is done by right-clicking on the Command Prompt Icon and selecting “Run as Administrator”

3.4 Sample App

The sample code provides you with an example of how to integrate the Windows 32/62 bit FWupdate lib into your application. Error handling, command line processing and loading the update image into memory is left to the customers.

Example – Developing FWUpdate Sample App

```
/*++

Copyright (c) 2012 Intel Corporation

Module Name:

    FwUpdLcl.c

Abstract:

    Sample application demonstrating the usage of the FWU Client interface

Revision History

--*/

#include <stdio.h>
#include "FWUCommon.h"
#include "FWUpdateLib.h"
#include "Common.h"
#include "me_status.h"


// This function handles the callback from the FWU library for displaying
// the percentage of completeness of the FW update
//
void DisplaySendStatus(float BytesSent, float BytestobeSent)
{
  float value = BytesSent/BytestobeSent * 100;

  if (value != 100) {
    printf ("Sending the update image to FW for verification:  [ %3.0f%%
]\r",value);
  } else {
    printf ("Sending the update image to FW for verification:  [ COMPLETE ]
\n");
  }
}

//
```

```
// This is the main entry point for the FW Update application.
// It handles the initialization of the required libraries and
// interfaces to the FW Update Library.
//

Int PerformUpdate (char* ImageBuffer)

{

  //Declare Variables

  //
 // Load the update image into memory and perform other application
specific
  // initialization
  //
  <TO_DO>
  //
  // Query the FW Update Client if a pending reset is required from
  // a previous update that requires a reset
  //
  if (GetLastStatus(&lastStatus)) {
    //
    // <TO_DO> Handle error…
    //
  }

  if (FWU_ERROR_SUCCESS != (status =
GetLastUpdateResetType(&lastResetType))) {
    //
    // Handle notifying user a pending reset is required
    //
  }

  //
  // Check if the update image is supported
  //
  CheckPolicyStatus = CheckPolicyBuffer((char *)ImageBuffer,
                                        (INT32)ImageLength,
                                        (INT32)bAllowSV,
                                        &Upd_Type,
                                        &ver);
  switch (Upd_Type) {
    case DOWNGRADE_SUCCESS:
    case SAMEVERSION_SUCCESS:
    case UPGRADE_SUCCESS:
      break;

    case DOWNGRADE_FAILURE:
    case SAMEVERSION_FAILURE:
      //
      // <TO_DO> Handle error of FW Update not allowed…
      //
      return FWU_ERROR_SUCCESS;
```

```
    default:
      break;
  }

  //
  // Download image to FW Update Client
  //
  FWUpdateStatus = FwUpdateFullBuffer ((char *)ImageBuffer,
                                       (unsigned int)ImageLength,
                                       Password,
                                       0,
                                       FWU_ENV_MANUFACTURING,
                                       mOemId,
                                       update_flags,
                                       &DisplaySendStatus);

  if (FWU_ERROR_SUCCESS != FWUpdateStatus) {

    //
    // <TO_DO> Handle error…
    //
  }

  //
  // Image downloaded to FW Update Client
  // Now query the status of the update being applied
  //
  Print (L"\nFW Update:  [  0 %% ]\r");
  index = 0;

  //
   // Loop through Polling for Fw Update Stages
  //
do{
    symbol = (++index % 2 == 0)?'|': '-';

//Loop to retry 3 times when there is a Heci Send or Receive failure;

    itr = 0;
    do
    {
    Status = FWUpdate_QueryStatus_Get_Response(&UpdateStatus,
                                    &TotalStages,&PercentWritten,
                                     &lastStatus, &lastResetType);

    printf("FW Update:  [ %d%% (Stage: %d of %d) (%c)]\r",PercentWritten,
    UpdateStatus,TotalStages,symbol);

        fflush(stdout);
        itr++;

}while(Status == FWU_IME_NOT_READY && itr < 3);
```

```
Sleep(1000);

//Loop until the percent written is not 100 and the status is a success

}while ((PercentWritten != 100 && Status == FWU_ERROR_SUCCESS)||(lastStatus
== STATUS_UPDATE_NOT_READY));

  if (!done) {
      //
      // <TO_DO> Handle timeout error
      //
  }
}
```

§ §

4 Function Description

This section describes all the functions listed in FWUpdateLib.h. It explains the purpose, Input arguments and return types.

4.1 Get Interfaces

unsigned int GetInterfaces(unsigned short *interfaces);

Purpose: This function gets the local FW update settings from Intel® MEBX to determine whether or not Firmware can be updated.

Arguments	***Interfaces*** - whether the Local FW Update is disabled (0) or enabled (1) or password protected (2)
Returns	Gets the Interfaces from HECI 0 = Success Non-zero value = Failure

4.2 Get Last Status

unsigned int GetLastStatus(unsigned int *lastStatus);

Purpose: This function will get the previous FW update status to ensure that FW update was successfully executed.

Arguments	***Laststatus*** – Last FW Update process Status (E.g Success, Invalid OEM ID, FW Version mismatch etc) Refer “me_status.h” for specific values
Returns	Gets the last FW update status from HECI 0 = Success Non-zero value = Failure

4.3 Get Last Update Reset Type

unsigned int GetLastUpdateResetType(unsigned int *lastResetType);
Purpose: This function will get the last Update Reset type to determine what type of system reset is required to load the partition into the memory.

Arguments	***LastResetType*** - The last FWUpdate reset type No reset – 0 Host reset – 1 ME – 2 Global - 3
Returns	Gets the last FW update reset type from HECI 0 = Success Non-zero value = Failure

4.4 Save Restore Point

int SaveRestorePoint(char * ImageFileLib);

Purpose: This function will retrieve an update image from the FW based on the current FW running. The update image will be saved to the user-specified file.

Arguments	***Image File*** - The user named binary file where the update image will be saved to
Returns	Get the restore point and save to Image File 0 = Success Non-zero value = Failure

4.5 Get FW Version

int GetFwVersion(char * imageFileLib,unsigned short *major,unsigned short *minor, unsigned short *hotfix, unsigned short *build);

Purpose: This function retrieves the current FW version from the Flash to be displayed for the user.

Arguments	***Image File*** - The binary image passed Out parameters : major = Major Version minor = Minor Version hotfix = Hot Fix Version build = Build Version
Returns	0 = Success Non-zero value = Failure

4.6 Check Policy

unsigned int CheckPolicy(char* ImageFileLib, int AllowSV, UPDATE_TYPE *Upd_Type,VersionLib *ver);

Purpose: This function determines whether it is a FW upgrade/downgrade or same version update using a file.

Arguments	***Image File*** - Binary Image file ***AllowSV*** - Allow Same Version flag (Set to 1 to execute same version flow) ***Update Type*** - Update Type Output. Can be DOWNGRADE_SUCCESS = 0, DOWNGRADE_FAILURE = 1, SAMEVERSION_SUCCESS = 2, SAMEVERSION_FAILURE = 3, UPGRADE_SUCCESS = 4, UPGRADE_PROMPT = 5, ***Ver***- FW Version (Major, Minor, Hotfix, Build)
Returns	0 = Success Non-zero value = Failure

4.7 Check Policy Buffer

unsigned int CheckPolicyBuffer(char* buffer, int bufferLength, int AllowSV, UPDATE_TYPE *Upd_Type, VersionLib *ver);

Purpose: This function determines whether it is a FW upgrade/downgrade or same version update using buffer.

Arguments	***Buffer*** - buffer to access ***BufferLength*** - Length of buffer ***AllowSV*** - Allow Same Version flag ***Update Type***– Update Type Output. Can be DOWNGRADE_SUCCESS = 0, DOWNGRADE_FAILURE=1, SAMEVERSION_SUCCESS=2, SAMEVERSION_FAILURE=3, UPGRADE_SUCCESS=4, UPGRADE_PROMPT=5, ***Ver*** – FW Version (Major, Minor, Hotfix, Build)
Returns	0 = Success Non-zero value = Failure

4.8 Verify OEM Id

bool VerifyOemId(_UUID id);

Purpose: This function verifies the OEM ID provided by the user with the one embedded in the FW.

Arguments	***Id*** - OEM id
Returns	True = OEM ID matched False = OEM id mismatch

4.9 Get Ipu Partition Attributes

unsigned int GetIpuPartitionAttributes(FWU_GET_IPU_PT_ATTRB_MSG_REPLY *FwuGetIpuAttrbMsgInfo);

Purpose: This function gets the number of Independent partial update partition attributes that is currently present and also the list of expected IPUs to be updated.

Arguments	Out parameter: FWU_GET_IPU_PT_ATTRB_MSG_REPLY – is a data structure with IPU related information
Returns	0 = Success 8193 = Heci Device not found 8204 = Heci message has incorrect message type 8728 = Heci Buffer Size is Small Error 8710 = Insufficient memory Error 8776 = Failure to Send or Receive the Get Partition Attribute Command Or even when FW returns an error status after receiving command

4.10 Get FW Update Info Status

unsigned int GetFwUpdateInfoStatus(FWU_INFO_FLAGS *StatusFlags);

Purpose: This function gets the current status of the firmware.

Note: This API is not used by the FWUpdate tool. It is being used by the UNS services.

Arguments	***StatusFlags*** - BITS 0:1 (2 bits) 0 = No recovery; 1 = Full Recovery Mode; 2 = Partial Recovery Mode (unused at present). BIT2; IPU_NEEDED bit, if set we are in IPU_NEEDED state. BIT3; FW_INIT_STATUS done. BIT4; FWU_IN_PROGRESS
Returns	0 = Success 8193 = Heci Device not found 8204 = Heci message has incorrect message type 8213 = Heci Buffer Size is Small Error 8710 = Insufficient memory Error 8777 = Failure in Send or Receive of the Get Info Status Command. Or even when FW returns an error status after receiving command

4.11 FW Update Query Status Get Response

unsigned int FWUpdate_QueryStatus_Get_Response(unsigned int* UpdateStatus, unsigned int *TotalStages, unsigned int* PercentWritten, unsigned int * LastUpdateStatus, unsigned int * LastResetType);

Purpose: This function queries FW to get response regarding the different stages of FW Update process.

Arguments	***UpdateStatus*** - indicates the current FW Update stage being executed. ***TotalStages*** – indicates the total number of FW Update stages available. ***PercentWritten*** – indicates the percentage complete of the FW Update process ***LastUpdateStatus –*** indicates the status of the fwupdate process just completed ***LastResetType –*** indicates Reset type required for the fwupdate process just completed
Returns	0= Success 1 = Invalid Manifest Data in partition 8193 = Heci Device not found 8204 = Heci message has incorrect message type 8213 = Heci Buffer Size is Small Error 8710 = Insufficient memory Error 8724 = Failure to send or receive messages to heci to get Status Info 8741 = FW returns incorrect Message Type

4.12 FW Update Full – Using File

unsigned int FwUpdateFull(char* _imageFileLib, char* _pwd,int _forceResetLib, void(*func)(float,float));

Purpose: Performs the full FW update using the update file provided by the calling function.

Arguments	***Image File*** – Binary Image file ***Password*** – MEBX Password ***ForceReset*** – Send reset request forcefully to the API ***Func pointer*** – (bytes of Binary image currently sent to the FW, Total bytes to be sent to the FW)
Returns	0 = Success Non-zero value = Failure

4.13 FW Update Full – Using Buffer

unsigned int FwUpdateFull (char* buffer, unsigned int bufferLength, char* _pwd,int _forceResetLib, unsigned int UpdateEnvironment,_UUID OemID, UPDATE_FLAGS_LIB update_flags, void(*func)(float,float));

Purpose: This function performs the full FW Update using the Buffer provided by the calling function.

Arguments	***Buffer*** – Buffer with the update image ***Buffer Length*** – Length of buffer ***Password*** – MEBX Password ***ForceResetLib*** – Flag to perform system reset ***UpdateEnvironment*** – differentiates various firmware update process environment within the firmware (manufacturing/non-manufacturing) ***UUID OEMID*** – OEM ID ***update_flags*** – flag to indicate FW of recovery/rollback ***Func pointer*** – (bytes of Binary
Returns	0 = Success Non-zero value = Failure

4.14 FW Update Partial

unsigned int FwUpdatePartial(char *ImageFileName, unsigned int PartitionID, unsigned int Flags, IPU_UPDATED_INFO *IpuUpdatedInfo, void(*func)(float, float));

Purpose: This function performs a Partial FW Update by using the file provided by the user.

Arguments	***ImageFileName*** - denotes the name of UPD binary image for input. ***PartitionID*** – denotes the partition ID, which could be WLAN (wcod) or language (locl). ***WOCD ID*** = 0x244f4357 and ***LOCL ID*** = 0x4C434F4C ***Flags*** – Bit 0 of the flags is used to set allow same version update. Other bits are reserved and can be used in the future. ***IpuUpdatedInfo***- Contain the information that is actually used to update the IPU partition. ***Func pointer*** – (bytes of Binary image currently sent to the FW, Total bytes to be sent to the FW)
Returns	0 = Success 2 = No FPT found in the update image

	8708 = FW upate process already started 8714 = Failed to open input file 8710 = Insufficient memory Error 8771 = Invalid file used for partial FW update (only FULL and Partial images are supported) 8741 = FW returns incorrect Message Type or wrong image ordering 8193 = Heci Device not found 8204 = Heci message has incorrect message type 8213 = Heci Buffer Size is Small Error 8707 = Internal error within the library 8719 = FW update is disabled in MEBX 8722 = FW Update start message type error 8724 = HECI device not ready to communicate 8746 = Invalid image size 8747 = Global buffer not available 8748 = Invalid parameters sent to Firmware 8758 = FW image is under version control 8759 = FW image version history check fail 8761 = Firmware write file failure 8762 = firmware read file failure 8763 = firmware delete file failure 8764 = partition layout not compatible 8765 = FW downgrade not allowed due to data mismatch 8766 = MEBX Password mismatch 8767 = MEBX Password retry exceeded threshold 8778 = The partition ID requested for update is not expected by the FW 8784 = Error for no partition ID provided to the API

4.15 FW Update Partial Buffer

unsigned int FwUpdatePartialBuffer(char* buffer,unsigned int bufferLength, unsigned int PartitionID, unsigned int Flags, IPU_UPDATED_INFO *IpuUpdatedInfo, void(*func)(float, float));

Purpose: This function performs the Partial FW Update. If the requested partition is expected by the Firmware, it will search for the expected partition in the image provided, extract it and send it to the FW to perform the update. If the expected partition is not found in the image an invalid file error will be returned by the tool. If the requested partition is not expected by the firmware an error will be returned to the user.

Note: For Partial FW update the image provided must either be a Full or Partial image. A full image starts with a FPT and contains FTP and NFTP partitions. A partial image starts with either WCOD or LOCL partitions.

Arguments	***Buffer*** - Buffer ***Buffer Length*** – Length of buffer
Returns	***Partition ID*** - denotes the partition ID, which could be WLAN (wcod) or language (locl). ***WOCD ID*** = 0x244f4357 and ***LOCL ID*** = 0x4C434F4C ***Flags***: Bit 0 of the flags is used to set allow same version update. Other bits are reserved and can be used in the future. ***IpuUpdatedInfo*** - Contain the information that is actually used to update the IPU partition. 0 = Success Non-zero value = Failure

§ §

5 Return Codes & Error Values

List of all the return codes and Error Values:

5.1 Return codes and Error Values

0	FWU_ERROR_SUCCESS
8193	HECI Device not Found
8199	Failure to send or receive messages to HECI to get Status Info
8204	HECI message has incorrect message type
8213	HECI Buffer Size is Small Error
8707	Internal error within the library
8710	FWU_NO_MEMORY - Insufficient memory Error
8713	Invalid Image file header
8714	FWU_FILE_OPEN - Failed to open input file
8727	Failure to send or receive HECI messages to HECI client
8741	FW returns incorrect Message Type or wrong image ordering
8746	Invalid image size
8747	Buffer not available
8748	Invalid parameters sent to Firmware
8761	Firmware write file failure
8762	Firmware read file failure
8763	Firmware delete file failure
8764	Partition layout not compatible
8765	Downgrade NOT allowed, data mismatched
8771	Invalid file used for partial FW update (only FULL and Partial images are supported)
8776	Failure in Send or Receive of the Get Partition Attribute Command. Or even when FW returns an error status after receiving command
8778	The partition ID requested for update is not expected by the FW
8793	FW Update/downgrade is not allowed to the supplied FW image
8794	FW downgrade is not allowed due to SVN restriction

§ §