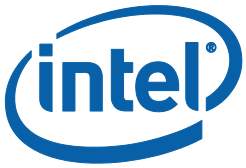


UEFI Sample Application Leveraging FWUpdate API Library

Intel® Management Engine Firmware 11.8 SKUs

White Paper

November 2017

Revision: 1.2

Intel Confidential

Contents

Revision History

Revision Number	Description	Revision Date
1.2	Added API Details for Retrieving ISH Firmware Version, API's for retrieving FW Type and SKU Type	November 2017

§ §

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
WCOD	Wireless Card Device

§ §

2 Firmware Update Flow

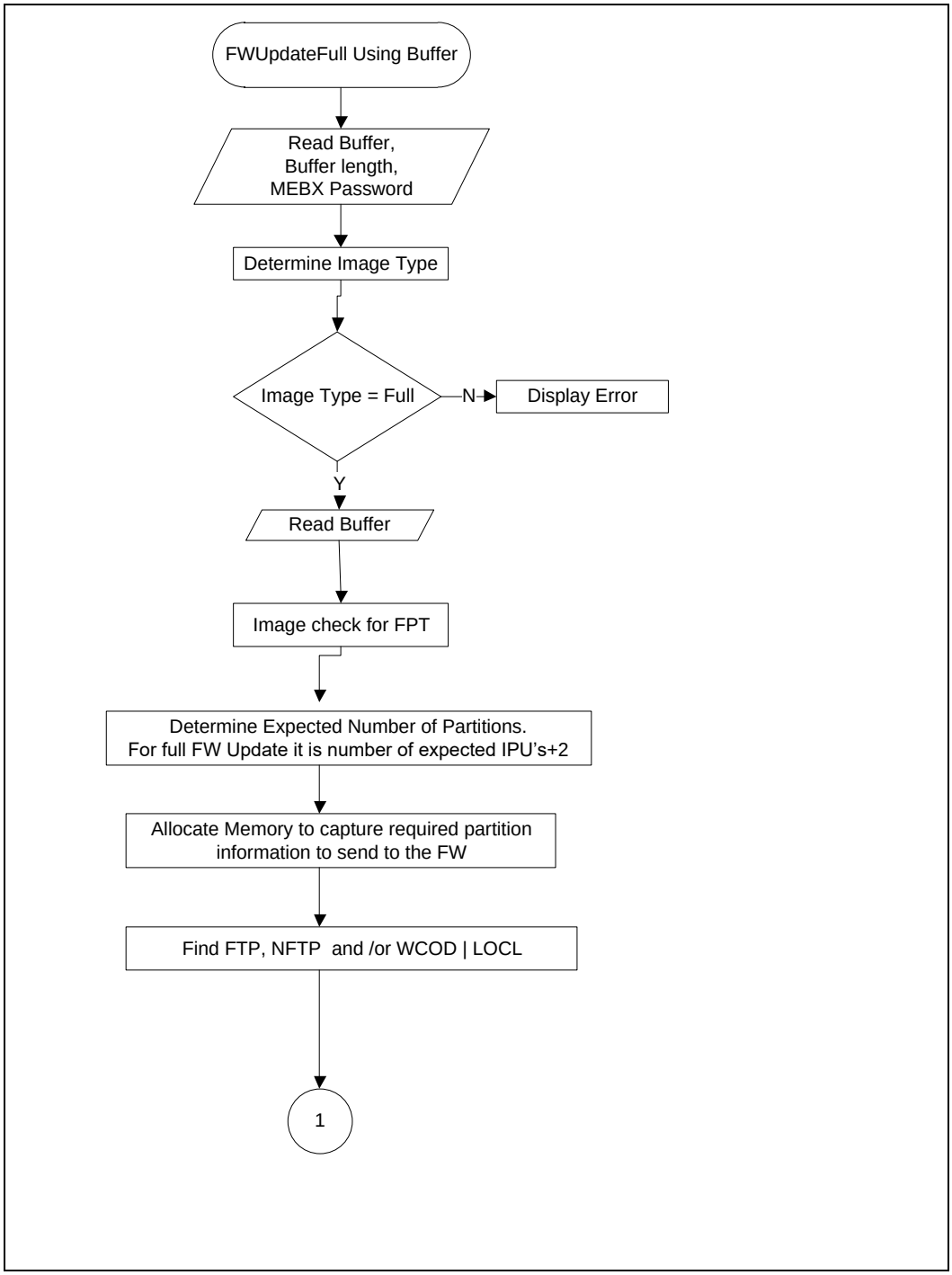
2.1 Purpose

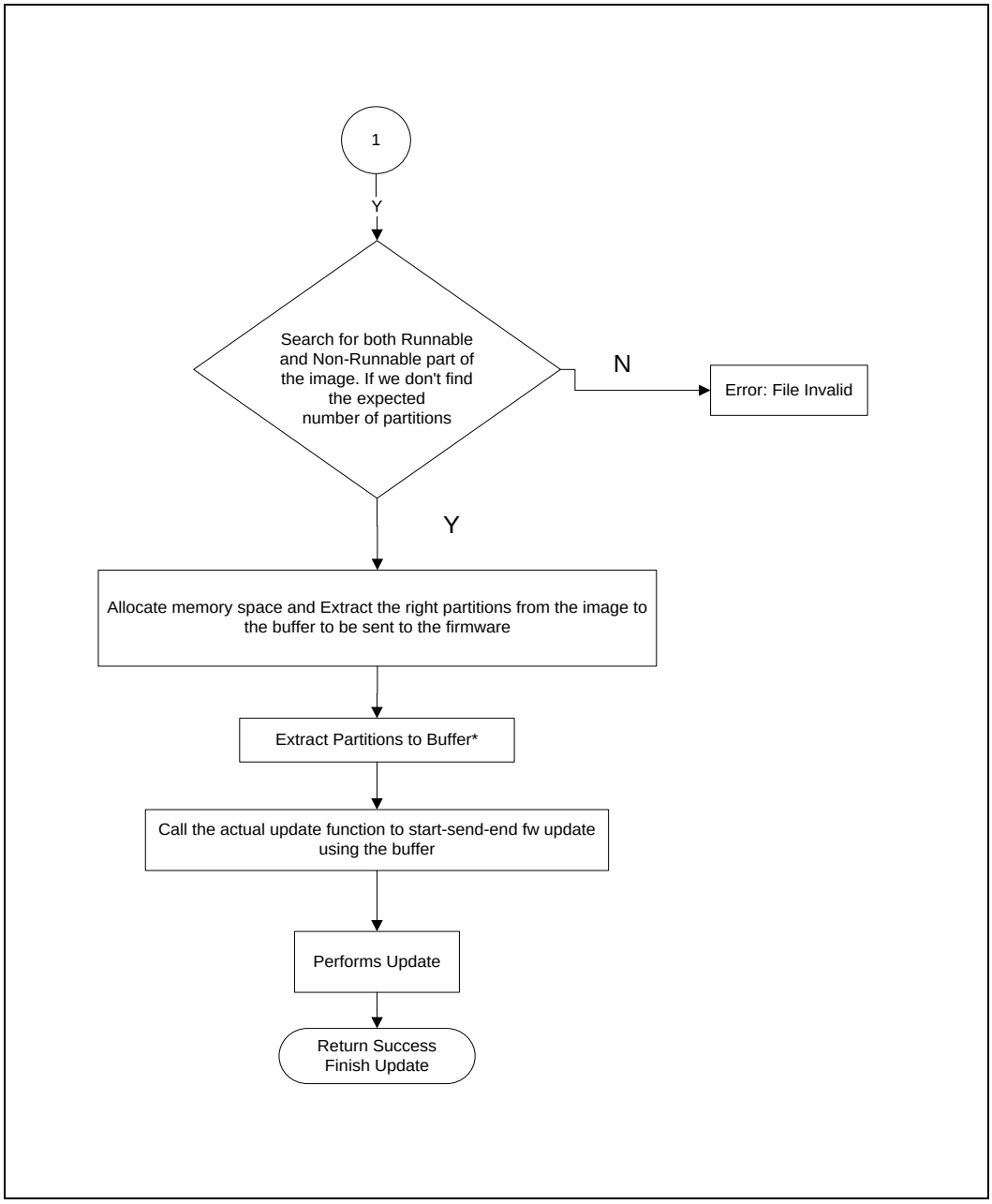
The firmware update process is essential for distributing FW images with bug fixes and also updating WCOD & LOCL regions. By utilizing the APIs provided in the Firmware Update Library, a program can update the existing FW image on a platform. The library sends the new FW image to the Intel® Management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

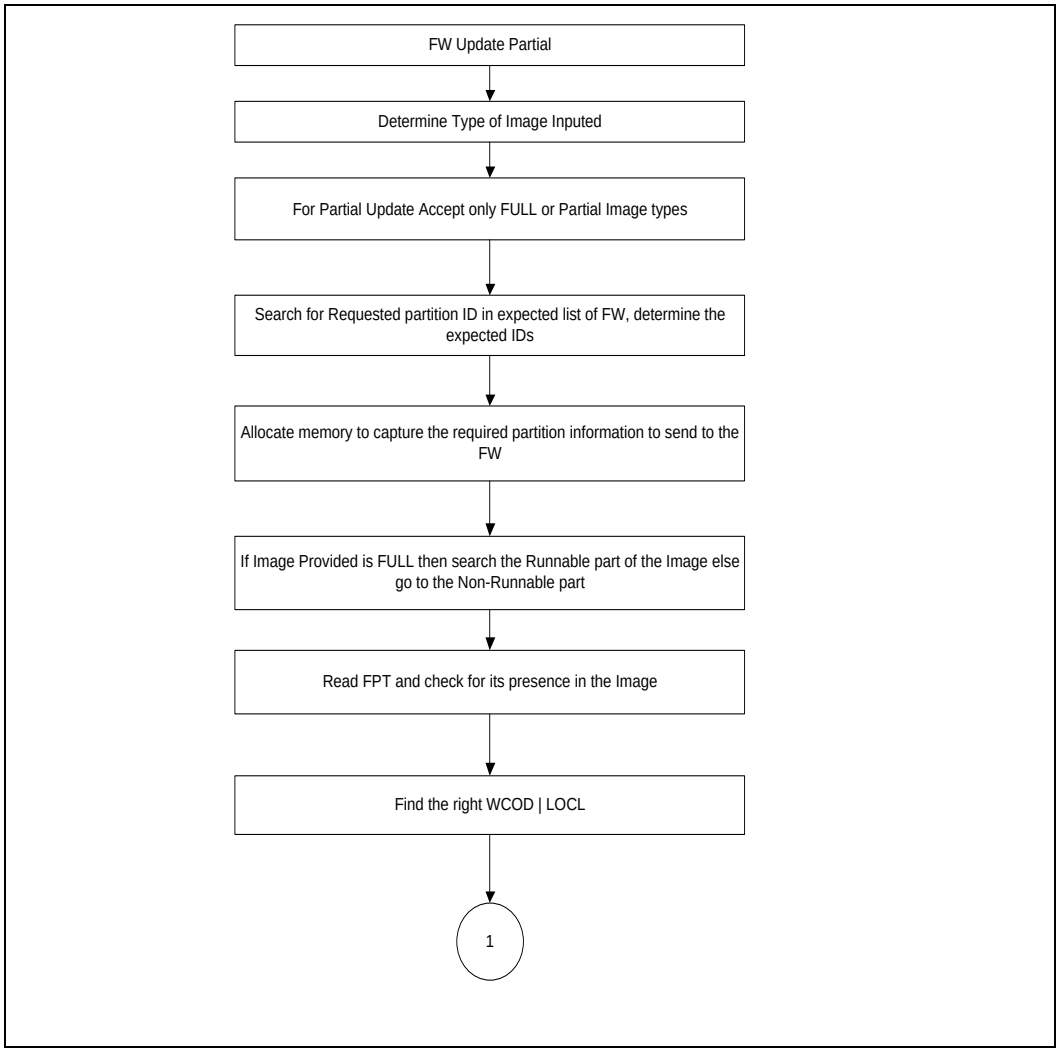
2.2 FW Update Library Description

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

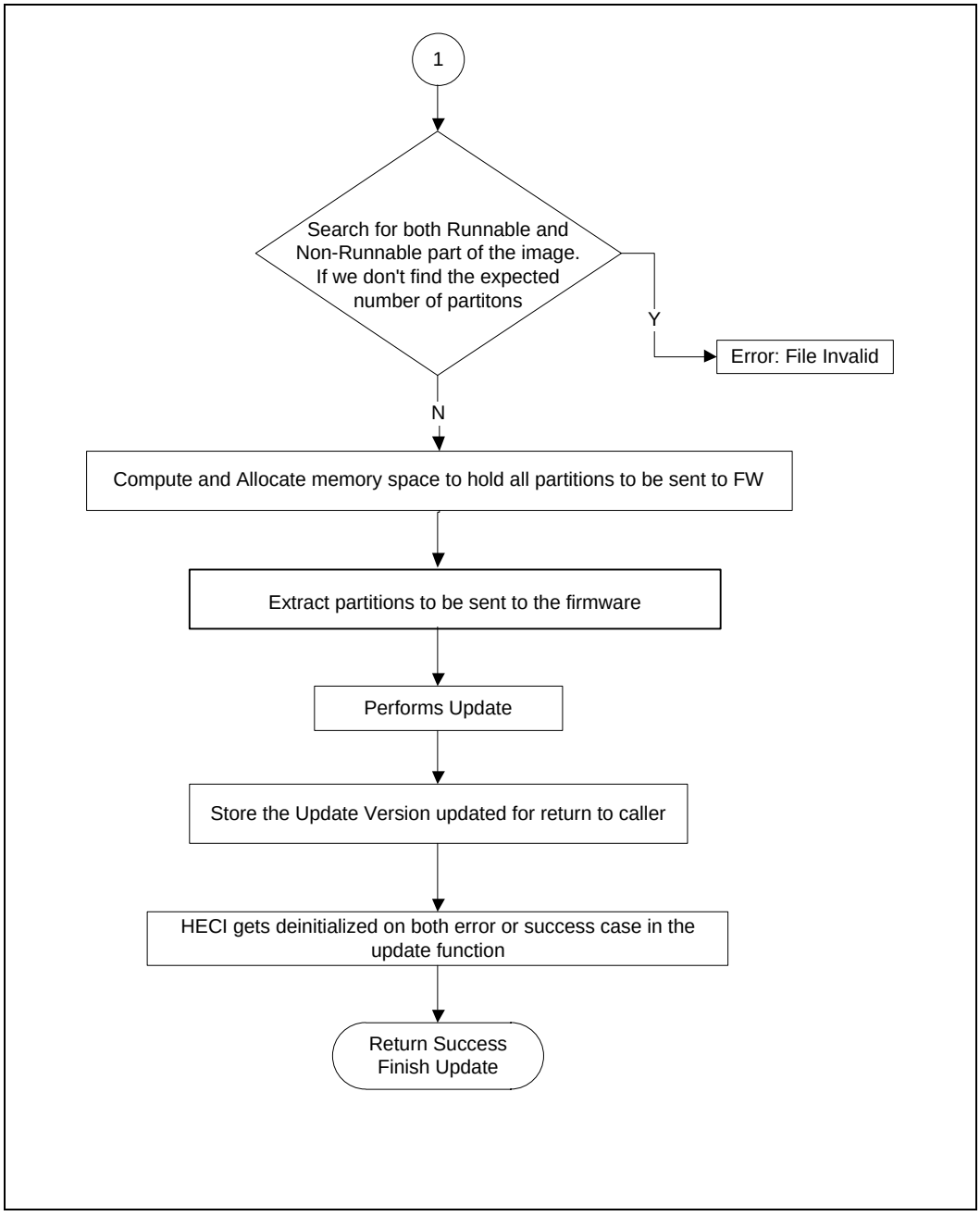
2.2.1 Full Firmware Update Flow




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



2.2.2 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 the EFI toolkit V2.0 and MSDK.

3.2 Setup

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

1. You will need to include/reference the "FWUpdateLib.h" file in your program.

2. A make file referencing the FW Update Library. Libraries to Reference:

```
LIBS = $(LIBS) \
      $(SDK_BUILD_DIR)\lib\libc\libc.lib \
      $(SDK_BUILD_DIR)\lib\libefi\libefi.lib \
      $(SDK_BUILD_DIR)\lib\libsmbios\libsmbios.lib \
      $(SDK_BUILD_DIR)\lib\libefishell\libefishell.lib \
      $(SDK_BUILD_DIR)\lib\FwUpdateEfiLib\FwUpdateEfiLib.lib
```

3.3 Sample App

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

Example – Developing FWUpdate Sample App

Note: Please Refer to the Actual Sample App Source Code under EFI/SampleSource/ Provided in the Kit for Proper Details.

```
/*++

Copyright (c) 2014-2016 Intel Corporation

Module Name:

    FwUpdLcl.c

Abstract:
```


```
    Sample application demonstrating the usage of the FWU Client UEFI
interface

Revision History

--*/


#include "efi.h"
#include "efilib.h"
#include "Fwu_Common.h"
#include "Common.h"
#include "me_status.h"
#include "FWUpdateLib.h"
#include "cse_basic_types.h"
#include "typedef.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;

    UINT32  pValue = (UINT32)Value;

    if (pValue != 100)
    {
        Print (L"Sending the update image to FW for verification:  [ %d%%
]\r",pValue);
    }else
    {
        Print (L"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.
//
EFI_DRIVER_ENTRY_POINT (InitializeFwUpdLclApplication)

    EFI_STATUS
    InitializeFwUpdLclApplication (
    IN EFI_HANDLE           ImageHandle,
    IN EFI_SYSTEM_TABLE     *SystemTable
    )
{
    EFI_STATUS        Status;
    CHAR16            ImageName[256];
    UINTN             ImageLength = 0;
```

```
UINT8              *ImageBuffer = NULL;
BOOLEAN            bAllowSV;
BOOLEAN            bUsePassword;
BOOLEAN            bPid;
BOOLEAN            bF;
BOOLEAN            bPdt;
BOOLEAN            bIshVer;
//CHAR               Password[9];
char               *Password = NULL;
UINT32             FWUpdateStatus;
DWORD              loops = 500;
BOOLEAN            done = FALSE;
UINT32             lastStatus = 0;
UINT32             platCheck = 0;
FWVersion          fwVersion;
INT32              platCheckReturn = 0;
UINT32             CheckPolicyStatus = 0;
UPDATE_TYPE        Upd_Type;
VersionLib         ver;
UINT32             index = 0;
UINT32             status;
UINT32             UpdateStatus = 0;
UINT32             TotalStages = 0;
UINT32             PercentWritten = 0;
CHAR8              symbol;
UINT32             lastResetType;
UPDATE_FLAGS_LIB   update_flags;
UINT16             interfaces;
int                timer30s = 0;
unsigned int        indexMod;
int                 percentage0s = 0;
int                 percentdiff = 0;
UINT32              ComparePartID = 0;
UINT32              hexValueInstId = 0;
IPU_UPDATED_INFO    IpuUpdatedInfo;
UINT32              PartId = 0;
UINT32              fwuError;
FWU_GET_IPU_PT_ATTRB_MSG_REPLY FwuGetIpuAttrbMsgInfo;
bool                found = false;
UINT32              i, j = 0;
UINT16 major = 0;
UINT16 minor = 0;
UINT16 hotfix = 0;
UINT16 build = 0;
UINT32 itr = 0;

//
// Zero out the update flag structure
//
ZeroMem(&update_flags, sizeof(UPDATE_FLAGS_LIB));
ZeroMem((char*)&IpuUpdatedInfo,sizeof(IPU_UPDATED_INFO));

//
// Initialize the EFI Toolkit Library. Set BS, RT, &ST globals
```

```
    //  BS = Boot Services RT = RunTime Services
    //  ST = System Table
    //
    InitializeLib (ImageHandle, SystemTable);


    Print (L"\n Intel (R) Firmware Update Utility Sample Application \n");
    Print (L"\n Intel (R) Firmware Update Utility Version: %d.%d.%d.",
VER_MAJOR, VER_MINOR, VER_HOTFIX);
    Print (L"%d\n", VER_BUILD);

    Print (L"\n");

    Print (ID_INFO_1);

    //
    // Determine the command line arguments
    //
    Status = ParseCommandLine (ImageHandle, ImageName, &bAllowSV,
&bUsePassword, &bPid, &bF, &bPdt, &bIshVer);
    if (EFI_ERROR (Status))
    {
        DEBUG ((D_ERROR, "Unable to process command line - %r\n", Status));
        return Status;
    }

    //
    // Display ISH FW Version with '/g' option
    //
    if (bIshVer){
        Status = GetPartVersion(FPT_PARTITION_NAME_ISHC, &major, &minor,
&hotfix, &build);

        if (EFI_ERROR (Status))
        {
            DEBUG ((D_ERROR, "GetPartVersion Error %r\n", Status));
            return Status;
        }

        Print(L"ISH FW Version: %d.%d.%d.%d \n\n", major, minor, hotfix,
build);
    }

    //
    // Load image into memory buffer
    //
    Print (L"\n Loading image into memory : ...\n");

    Status = GetUpdateImage (ImageHandle, ImageName, &ImageLength,
&ImageBuffer);
    if (EFI_ERROR (Status)) {
        Print(L" %r ",Status);
        return Status;
    }
```

```
    if (bPdt)
    {
        Print(L"Sending Image for Executing PDT Update. \n");

        Status = HeciPdt((char *)ImageBuffer, (unsigned int)ImageLength);
        if (EFI_ERROR(Status)) {
            Print(L"Send Failed. \n");
        }
        else {
            Print(L"Send Succeeded. \n");
        }
    }
    else
    {
        //
        // Get the current status of the ME FWUpdate Client - verifies if
the client is
        // installed
        //

        if (GetLastStatus(&lastStatus))
        {
            Print (ID_ERROR_19, FWU_LAST_STATUS);
            return EFI_SUCCESS;
        }
        //
        // Is there a pending reset?
        //

        if (GetLastUpdateResetType (&lastResetType))
        {
            Print (ID_ERROR_19, FWU_LAST_STATUS);
            return EFI_SUCCESS;
        }
        if (STATUS_UPDATE_HOST_RESET_REQUIRED == lastStatus)
        {
            Print (ID_ERROR_51, FWU_REBOOT_NEEDED);
            return EFI_SUCCESS;
        }

        if (IsUpdateStatusPending (lastStatus))
        {
            Print (ID_ERROR_20, FWU_UPD_PROCESS);
            return EFI_SUCCESS;
        }

        switch (lastResetType)
        {
        case MFT_PART_INFO_EXT_UPDATE_ACTION_HOST_RESET:

        case MFT_PART_INFO_EXT_UPDATE_ACTION_GLOBAL_RESET:
            Print (ID_ERROR_51, FWU_REBOOT_NEEDED);
            return EFI_SUCCESS;
```

```
            break;
        default:

            break;
        }


        Print (ID_INFO_3);

        //
        // Is update supported?
        //
        if (GetInterfaces (&interfaces))
        {
            Print (ID_ERROR_19, FWU_LAST_STATUS);
            return EFI_SUCCESS;
        }

        switch (interfaces)
        {
        case FW_UPDATE_DISABLED:
            Print (L"Local FWUpdate is Disabled\n");
            return EFI_SUCCESS;
        case FW_UPDATE_PASSWORD_PROTECTED:
            Print (L"Local FWUpdate is Password Protected\n");
            break;
        case FW_UPDATE_ENABLED:
            break;
        default:
            break;
        }

        Print (L"\n Checking Firmware Parameters ... \n \n");
        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:
            Print (L"FW Update downgrade not allowed\n");
            return EFI_SUCCESS;
            break;
        case SAMEVERSION_FAILURE:
            Print (L"FW Update same version not allowed, specify /s on
command line\n");
            return EFI_SUCCESS;
            break;
```

```
        default:
            break;
        }

        if(bPid)
        {
            Print(L"\n Executing ISH Partial FWUpdate");

            ComparePartID = FPT_PARTITION_NAME_ISHC;
            //Print(L"\n compareID: 0x%x",ComparePartID);

            //Get Partition Attribute from Firmware
            if (FWU_ERROR_SUCCESS != (fwuError =
GetExtendedIpuPartitionAttributes(&FwuGetIpuAttrbMsgInfo,
FWU_IPU_UPDATE_OPERATION)))
            {
                DisplayTextForReturnErrorCode(fwuError);
                return fwuError;
            }

            PartId = 0;
           //Loop through expected partitions from FW to find partition
requested
            for(j=0;j<FwuGetIpuAttrbMsgInfo.NumOfPartition;j++)
            {
                 if(ComparePartID ==
FwuGetIpuAttrbMsgInfo.PtAttribute[j].PtNameId)
                 {
                      PartId =
FwuGetIpuAttrbMsgInfo.PtAttribute[j].PtNameId;
                      found = true;
                      break;
                 }
            }

            if(!found)
            {
               DisplayTextForReturnErrorCode(FWU_PID_NOT_EXPECTED);
               //Print(L"ParID: 0x%x\tInstId: 0x%x
\n",ComparePartID,hexValueInstId);
                return FWU_PID_NOT_EXPECTED;
            }
            Print(L"%s", ID_WARN_0);
          ////Actual Partial FW update
          //
          // Password hack for testing - replace with OEM version if
password required
          //
            if (!bUsePassword)
            {
                ZeroMem (Password, sizeof (Password));
            }
            if (bUsePassword)
            {
```

```
                FWUpdateStatus = FwUpdatePartialBuffer ((char *)ImageBuffer,
(unsigned
int)ImageLength,PartId,0,&IpuUpdatedInfo,"P@ssw0rd",FWU_ENV_MANUFACTURING,
mOemId, update_flags, &DisplaySendStatus);
          }
          else
          {
                FWUpdateStatus = FwUpdatePartialBuffer ((char *)ImageBuffer,
(unsigned int)ImageLength,PartId,0,&IpuUpdatedInfo,Password,
FWU_ENV_MANUFACTURING, mOemId, update_flags, &DisplaySendStatus);
          }

          if (FWU_ERROR_SUCCESS != FWUpdateStatus)
          {
               DisplayTextForReturnErrorCode(FWUpdateStatus);
              if (ImageBuffer)
              {
                  FreePool (ImageBuffer);
              }
              return EFI_SUCCESS;
          }

          if (ImageBuffer)
          {
               FreePool (ImageBuffer);
          }
       }else{
              //
              // Password hack for testing - replace with OEM version if
password required
              //
              Print(L"\n");
              Print(L"%s \n", ID_WARN_0);
              /*if (!bUsePassword)
              {
                   ZeroMem (Password, sizeof (Password));
              }*/

              //if (bUsePassword)
              //{
                 //  FWUpdateStatus = FwUpdateFullBuffer ((char
*)ImageBuffer, (unsigned int)ImageLength, "P@ssw0rd", 0,
FWU_ENV_MANUFACTURING, mOemId, update_flags, &DisplaySendStatus);
              //}
              //else
              //{
                   FWUpdateStatus = FwUpdateFullBuffer ((char
*)ImageBuffer, (unsigned int)ImageLength, Password, 0,
FWU_ENV_MANUFACTURING, mOemId, update_flags, &DisplaySendStatus);
              //}

              if (FWU_ERROR_SUCCESS != FWUpdateStatus)
              {
                   DisplayTextForReturnErrorCode(FWUpdateStatus);
```

```
                        if (ImageBuffer)
                        {
                            FreePool (ImageBuffer);
                        }
                        return EFI_SUCCESS;
                        //return FWUpdateStatus;
                    }

                    if (ImageBuffer)
                    {
                        FreePool (ImageBuffer);
                    }
                }

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

                index  =  0;
                //
                // Loop through Polling for Fw Update Stages
                //
                ProgressDot();
                do {
                    //We mod4 the index to determine which ascii animation frame to
display for this iteration.
                    indexMod = (++index % 4);
                    //symbol = (++index % 2 == 0)?'|': '-';
                    switch(indexMod)
                        //loop through (|) (/) (-) (\) (|) (/) ...
                    {
                    case CMD_LINE_STATUS_UPDATE_1: symbol =  '|';  break;
                    case CMD_LINE_STATUS_UPDATE_2: symbol =  '/';  break;
                    case CMD_LINE_STATUS_UPDATE_3: symbol =  '-';  break;
                    case CMD_LINE_STATUS_UPDATE_4: symbol =  '\\'; break;
                    }
                    Status = FWUpdate_QueryStatus_Get_Response(&UpdateStatus,
&TotalStages, &PercentWritten, &lastStatus, &lastResetType);
                    if(PercentWritten > 100)
                    {
                        break;
                    }
                    if (Status == FWU_ERROR_SUCCESS)
                    {
                        Print (L"FW Update:  [ %d%% (%c)]\r" ,PercentWritten,
symbol);
                    } else if (lastStatus != STATUS_UPDATE_NOT_READY)
                    {
                        Print (L"\n");
                        break; //break out of the loop
                    }
```

```
                BS->Stall(100000); // Wait 1 sec before polling again
                if(timer30s >= 30)
                {
                    percentdiff = PercentWritten - percentage0s;
                    if(percentdiff < 1)
                    {
                        //TODO: Add timeout when add cmdline option
                        //Status = FWU_UPDATE_TIMEOUT;
                    } else
                    {
                        percentage0s = PercentWritten;
                        timer30s = 0;
                    }
                } else
                {
                    timer30s++;
                }
            } while ((PercentWritten != 100) && (Status == FWU_ERROR_SUCCESS));

            switch (Status)
            {
            case FWU_NO_MEMORY:

            case FWU_IME_NO_DEVICE:
                Print (ID_ERROR_68, FWU_UPDATE_POLLING_FAILED);
                return EFI_SUCCESS;
            case FWU_IME_NOT_READY:
                DisplayTextForReturnErrorCode(status);
                return EFI_SUCCESS;
            case FWU_ERROR_FW:
                Print (ID_ERROR_69, FWU_ERROR_FW, UpdateStatus);
                return EFI_SUCCESS;
            default:
                break;
            }

            switch (lastStatus)
            {
            case STATUS_SUCCESS:
                switch (lastResetType)
                {
                case MFT_PART_INFO_EXT_UPDATE_ACTION_NONE:

                case MFT_PART_INFO_EXT_UPDATE_ACTION_CSE_RESET:
                    Print (L"\nFW Update is completed successfully.\n");
                    break;
                case MFT_PART_INFO_EXT_UPDATE_ACTION_HOST_RESET:

                case MFT_PART_INFO_EXT_UPDATE_ACTION_GLOBAL_RESET:
                    Print (L"\nFW Update is complete and a reboot will run the
new FW.\n");
                    break;
                default:
```

```
                Print (L"\nFW Update is complete and a reboot will run the
new FW.\n");
                break;
            }
            fwuError = FWU_ERROR_SUCCESS;
            break;
        case STATUS_UPDATE_IMAGE_INVALID:
            DisplayTextForReturnErrorCode(FWU_IMG_HEADER);
            break;
        case STATUS_UPDATE_INTEGRITY_FAILURE:
            DisplayTextForReturnErrorCode(FWU_SGN_MISMATCH);
            break;
        case STATUS_UPDATE_SKU_MISMATCH:
            DisplayTextForReturnErrorCode(FWU_SKU_MISMATCH);
            break;
        case STATUS_UPDATE_FW_VERSION_MISMATCH:
            DisplayTextForReturnErrorCode(FWU_VER_MISMATCH);
            break;
        case STATUS_UPDATE_GENERAL_FAILURE:
            DisplayTextForReturnErrorCode(FWU_GENERAL);
            break;
        case STATUS_UPDATE_OUT_OF_RESOURCES:
            DisplayTextForReturnErrorCode(FWU_NO_MEMORY);
            break;
        case STATUS_UPDATE_AUDIT_POLICY_FAILURE:
            DisplayTextForReturnErrorCode(FWU_AUDIT_POLICY_FAILURE);
            break;
        case STATUS_UPDATE_ERROR_CREATING_FT:
            DisplayTextForReturnErrorCode(FWU_ERROR_CREATING_FT);
            break;
        case STATUS_UPDATE_SAL_NOTIFICATION_ERROR:
            DisplayTextForReturnErrorCode(FWU_SAL_NOTIFICATION_ERROR);
            break;
        case STATUS_INVALID_OEM_ID:
            DisplayTextForReturnErrorCode(FWU_INVALID_OEM_ID);
            break;
        case STATUS_DOWNGRADE_NOT_ALLOWED_VCN_RESTRICTION:
            DisplayTextForReturnErrorCode(FWU_IMAGE_UNDER_VCN);
            break;
        case STATUS_DOWNGRADE_NOT_ALLOWED_SVN_RESTRICTION:
            Print("FW downgrade is not allowed due to SVN restriction.\n");
            break;
        case STATUS_UPDATE_IMAGE_BLACKLISTED:
            Print("FW update/downgrade is not allowed to the supplied FW
image.\n");
            break;
        default:
            DEBUG ((D_ERROR, "lastStatus = %d\n",lastStatus));
            DisplayTextForReturnErrorCode(FWU_GENERAL);
            break;
        }
        return EFI_SUCCESS;
    }
}
```

§ §

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® Management Engine BIOS Extension (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 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

4.5 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

4.6 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

4.7 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

4.8 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

4.9 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

4.10 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

4.11 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.

FWUpdate API Library supports only Partial FWUpdate for ISH only. –i is the command line switch.

Example Usage: FwUpdLclApp.efi –i <Image.bin>

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

4.12 PDT Data (Sensor Calibration Data) Update

```
EFI_STATUS
HeciPdt (
    IN  char                *buffer,
    IN  UINT32          bufferLength
  );
```

Purpose: The function performs PDT Data Update i.e. Sensor Calibration Data Update.

Command Line Switch –d needs to be used in order to execute PDT Data Update.

Example for Usage:

FwUpdLclApp.efi –d <Pdt Data Binary>
FWUpdLclApp.efi –d INTC_pdt_SPT_RR3_BOM1_SENSORS

Arguments	***Buffer*** - Buffer ***Buffer Length*** – Length of buffer
Returns	***If Payload is sent to CSME successfully then Send Succeeded Message will be seen.***

4.13 ISH Firmware Version

```
int
GetPartVersion (
    UINT32 partID,
   UINT16 *major,
   UINT16 *minor,
   UINT16 *hotfix,
   UINT16 *build);
```

Purpose: The function helps retrieve ISH Firmware Version flashed on the platform.

4.14 Retrieve Firmware Type

```
UINT32 GetFwType(UINT32 *FwType);
```

4.15 Retrieve PCH type

```
UINT32 GetPchSKU(UINT32 *sku);
```

§ §

5 Return Codes & Error Values

5.1 Return codes and Error Values

ID	Error Message	Developer Actions
0	FWU_ERROR_SUCCESS	
8193	HECI Device not Found	Perform Reset, Try Running FWUpdate Process Again. Try 3 times else Contact Intel for this Error
8199	Failure to send or receive messages to HECI to get Status Info	Perform Reset, Try Running FWUpdate Process Again. Try 3 times else Contact Intel for this Error
8204	HECI message has incorrect message type	Contact Intel for this Error
8213	HECI Buffer Size is Small Error	Contact Intel for this Error
8707	Internal error within the library	Contact Intel for this Error
8710	FWU_NO_MEMORY - Insufficient memory Error	Contact Intel for this Error
8713	Invalid Image file header	Contact Intel for this Error
8714	FWU_FILE_OPEN - Failed to open input file	Make sure Appropriate file is used for updating the image. E.g. On Consumer Platform, ME Consumer Image should be used and not Corporate.
8727	Failure to send or receive HECI messages to HECI client	Perform Reset, Try running FWUpdate Process Again. Try 3 times else Contact Intel for this Error
8741	FW returns incorrect Message Type or wrong image ordering	Check ME Image to be updated is appropriate.

		Try Running FWUpdate Process Again
8746	Invalid image size	Make sure Image passed for FWUpdate is of appropriate size. E.g. Consumer Image, size should be 1,968 KB and for Corporate Image, size should be 7,160 KB
8747	Buffer not available	Make sure
8748	Invalid parameters sent to Firmware	Contact Intel for this Error
8761	Firmware write file failure	Contact Intel for this error. Potential Cause: After Image is passed to the firmware, firmware is seeing issues with file write or Inappropriate Image is passed in FWUpdate Tool.
8762	Firmware read file failure	Contact Intel for this error. Potential Cause: After Image is passed to the firmware, firmware is seeing issues with file read or Inappropriate Image is passed in FWUpdate Tool.
8763	Firmware delete file failure	Contact Intel for this error. Potential Cause: After Image is passed to the firmware, firmware is seeing issues with file delete or Inappropriate Image is passed in FWUpdate Tool.
8764	Partition layout not compatible	Contact Intel for this Error
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	Contact Intel for this Error.
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	

§ §