Developer's Guide
SLAU656–October 2015

# MSP Debug Stack (MSP430.dll) Developer's Guide

The MSP Debug Stack (MSP430.dll) is a dynamic link library (DLL) that provides functions for controlling and debugging TI MSP430™ ultra-low-power microcontrollers during the software development phase. The MSP Debug Stack controls the MSP430 microcontroller through the JTAG interface. The MSP Debug Stack provides device control (for example, run and stop), memory programming, and debugging functionality (for example, breakpoints).

The MSP Debug Stack supports the standard 4-wire JTAG and the low-pin-count debug interface called Spy-Bi-Wire (2-wire JTAG). All MSP430 debuggers can be used with the MSP Debug Stack.

The MSP Debug Stack simplifies control of the MSP430 microcontroller by isolating the user from the complexities of the JTAG protocol.

This document provides an overview of the MSP Debug Stack and how to use it to control MSP430 microcontrollers. Additional information is provided in the C header files that are installed with the DLL. Several sample programs and flow charts show the practical use of the MSP Debug Stack.

**NOTE:** This document assumes knowledge of C/C++, the dynamic link library mechanism, the MSP430 MCUs, and the MSP430 JTAG mechanism.

**NOTE:** Refer to the *MSP430 Hardware Tools User's Guide* (SLAU278) for information on hardware connection to the JTAG pins of the MCUs. Refer to the *MSP430 Programming Via the JTAG Interface User's Guide* (SLAU319) for details on the MSP430-specific JTAG implementation in silicon.

MSP430, LaunchPad, eZ430, eZ430 Chronos, Code Composer Studio are trademarks of Texas Instruments.
IAR Embedded Workbench is a trademark of IAR Systems.

## Contents


| | | |
|---|---|---|
| 1 | Abbreviations | 3 |
| 2 | Developer's Package Folder and File Structure | 3 |
| 3 | Using the MSP430.dll | 5 |
| 4 | MSP-FET430UIF Firmware Update Support | 16 |
| 5 | Supporting eZ430 Emulator Dongles | 19 |
| 6 | Application Examples | 20 |
| 7 | Installation of CDC for USB-FET Debuggers | 20 |
| 8 | Update MSP-FET430UIF With Hardware Revision 1.3 | 22 |
| 9 | References | 23 |


## List of Figures


| | | |
|---|---|---|
| 1 | Recommended Flow to Start an MSP430 Debug Session | 6 |
| 2 | Code Example to Start a Debug Session | 7 |
| 3 | Attach to Running target | 9 |
| 4 | Code Example for "Attach to running target" | 10 |
| 5 | Retrieve Information About Available USB-FETs or Debuggers | 12 |
| 6 | Code Example for Communication With Multiple USB-FETs or Debuggers | 13 |
| 7 | General Firmware Update Flow | 16 |
| 8 | USB-FET HID Recovery Flow | 17 |
| 9 | Update Tool | 18 |
| 10 | Start Firmware Update | 19 |
| 11 | Firmware Update Successful | 19 |
| 12 | New Hardware | 20 |
| 13 | Update Wizard | 21 |
| 14 | UIF Revision 1.3 | 22 |
| 15 | UIF Revision 1.4 | 22 |

# 1 Abbreviations

- **MSP-FET430UIF**: The TI MSP430 USB Debugging Interface (USB FET)
- **MSP-FET**: The TI MSP MCU Programmer and Debugger
- **MSP-EXP430F5529LP**: The TI MSP430F5529 USB LaunchPad™ Evaluation Kit (includes an eZ-FET lite debugger)
- **eZ430-RF2500**: The TI MSP430 Wireless Development Tool (includes an eZ430™ debugger)
- **SBW**: The Spy-Bi-Wire JTAG debug interface, which is used on low-pin-count MSP430 MCUs.
- **CDC**: Communication Device Class
- **MSP Debug Stack**: The name of the developer's package that contains the MSP430.dll and supporting material
- **USB-FET**: General term for different debuggers including MSP-FET430UIF, MSP-FET, eZ-FET, and eZ-FET lite

# 2 Developer's Package Folder and File Structure

The MSP430 DLL developer packet is composed of the following folders and files. Installing the MSP430 DLL developer package creates the following folders and files in the selected installation destination directory.

- **ApplicationExamples**: This folder contains a set of application examples on how to apply the DLL. See Section 6 for specific details on each code example.
- **Doc**: This folder contains the complete API documentation of the DLL in HTML and compressed HTML formats. This developer's guide is also in the Doc folder.
- **Driver**: This folder contains driver setup and files:
  - **CDC**: TI's CDC driver (DLL V3 only). Supports the MSP-FET430UIF, MSP-FET, and eZ-FET JTAG interfaces.
  - **VCP**: Deprecated. TI's VCP driver (DLL V2 only). Supports the MSP-FET430UIF JTAG interfaces.
  - **INF**: MS-Windows driver information file for the MSP430 Application UART that is available with the eZ430-RF2500 (eZ430 debuggers) emulator dongles (see Section 5 for details). Also see the *eZ430-RF2500 Development User's Guide* (SLAU227) for more information. This folder also contains the PreinstallCDC folder, which contains example source code that shows how to install the driver INF file on a MS-Windows PC.
  - **DriverX**: Deprecated. Teradyne's DriverX (Windows XP only). Supports MSP-FET430PIF JTAG interface. This interface is not natively supported by the V3 DLL. Only the old V2 DLL includes support for the MSP-FET430PIF JTAG interface.
- **Inc**: This folder contains all of the C header files that are needed to use the MSP430.dll in an application. These header files document all DLL functions, function prototypes, function parameters and function return values. The header files also include all required typedefs, #defines, enumerations, and data.
  - **MSP430.h**: This file is the main header file for the MSP430.dll, and it provides the function prototypes, typedefs, #defines, enumerations, and data structures for the functions of the DLL. MSP430.h is normally located in the same directory as the application source file, and it should be #included by the application source file. MSP430.h is used during compile-time (see Section 3.1).
  - **MSP430_Debug.h**: This file is a header file for the MSP430.dll, and it provides the function prototypes, typedefs, #defines, enumerations, and data structures for the debugging functions of the DLL. MSP430_Debug.h is normally located in the same directory as the application source file, and should be #included by the application source file. This file is used during compile-time (see Section 3.6).
  - **MSP430_EEM.h**: This file is a header file for the MSP430.dll, and provides the function prototypes, typedefs, #defines, enumerations, and data structures for the **enhanced** debugging functions of the DLL. MSP430_EEM.h is normally located in the same directory as the application source file, and should be #included by the application source file. This file is used during compile-time (see Section 3.7).
  - **MSP430_FET.h**: This file is a header file for the MSP430.dll, and provides the function prototypes,

typedefs, #defines, enumerations, and data structures for MSP-FET430UIF maintenance functions of the DLL. MSP430_FET.h is normally located in the same directory as the application source file, and should be #included by the application source file. This file is used during compile-time (see Section 4).

  - **HIL.h**: This file is provided as a reference for the HIL.dll (which is used only for the deprecated PIF).
  - **wtypes.h:** Required type definitions.
- **Lib**: This folder contains according library files.
  - **MSP430.lib**: This file is the library file for the MSP430.dll and is required to access functions of the DLL. MSP430.lib is normally located in the same directory as the application source file and should be added to the Linker Object/Library Modules list of the application. This file is used during link-time.
  - **HIL.lib**: This file is the library file for the HIL.dll (which is used only for the deprecated PIF).
- **MSP430.dll**: This file is the dynamic link library and contains the device control functions. This file is normally located in the same directory as the application's executable file, or in your computer system's default DLL folder. This file is used during run-time.

The MSP430.dll interfaces to the device's JTAG pins using a set of functions encapsulated in another Dynamic Link Library called the Hardware Interface Layer (HIL.dll). The HIL.dll provides a unified API for MSP430.dll which abstracts the peculiarities of the physical interface to the MSP430. The described software layer hierarchy permits the MSP430.dll to be used with any physical interface (that supports the required control of the JTAG pins) by providing a corresponding HIL.dll. A HIL.dll is supplied that controls the JTAG pins via a Texas Instruments FET Interface Module and the parallel port of a PC.

- **HIL.dll**: This file is the dynamic link library, and contains the HIL functions. This file is normally located in the same directory as the application's executable file, or in your computer system's default folder of DLLs. This file is used during run-time.
- **revisions.txt**: This file provides information about added features of dedicated versions of the DLL.

# 3 Using the MSP430.dll

## 3.1 General Application and Device Handling

The functions of the DLL are sequenced as follows:

1. Initialize the interface: MSP430_Initialize()
2. Set the device VCC: MSP430_GetExtVoltage(), MSP430_VCC(), MSP430_GetCurVCCT()
3. Configure the JTAG protocol (Spy-Bi-Wire 2-Wire JTAG, 4-wire JTAG) is optional. By default the protocol is selected automatically: MSP430_Configure()
4. Connect and identify target device: MSP430_OpenDevice()
5. Return the identified device: MSP430_GetFoundDevice()
6. Manipulate the device memory:
   - Execute erase operation: MSP430_Erase()
   - Read or write device memory: MSP430_Memory(), MSP430_ReadOutFile(), MSP430_ProgramFile()
   - Execute verify operation: MSP430_VerifyFile(), MSP430_VerifyMem(), MSP430_EraseCheck()
7. Manipulate device functionality:
   - Secure device (disable JTAG access): MSP430_Secure()
   - Execute device reset: MSP430_Reset()
   - Start device code execution: MSP430_Run()
   - Stop device code execution: MSP430_State()
8. Close device connection and CDC port: MSP430_Close()
9. Handle errors: MSP430_Error_Number(), MSP430_Error_String()

Figure 1 shows the start-up flow of an MSP430 debug session using the MSP430.dll.

Figure 2 shows the example code for starting a debug session including all needed error handling executed by the MSP430_Error_Number() and MSP430_Error_String() functions. The MSP430_DLL.chm help file offers detailed information on all DLL functions, their parameters and return values.

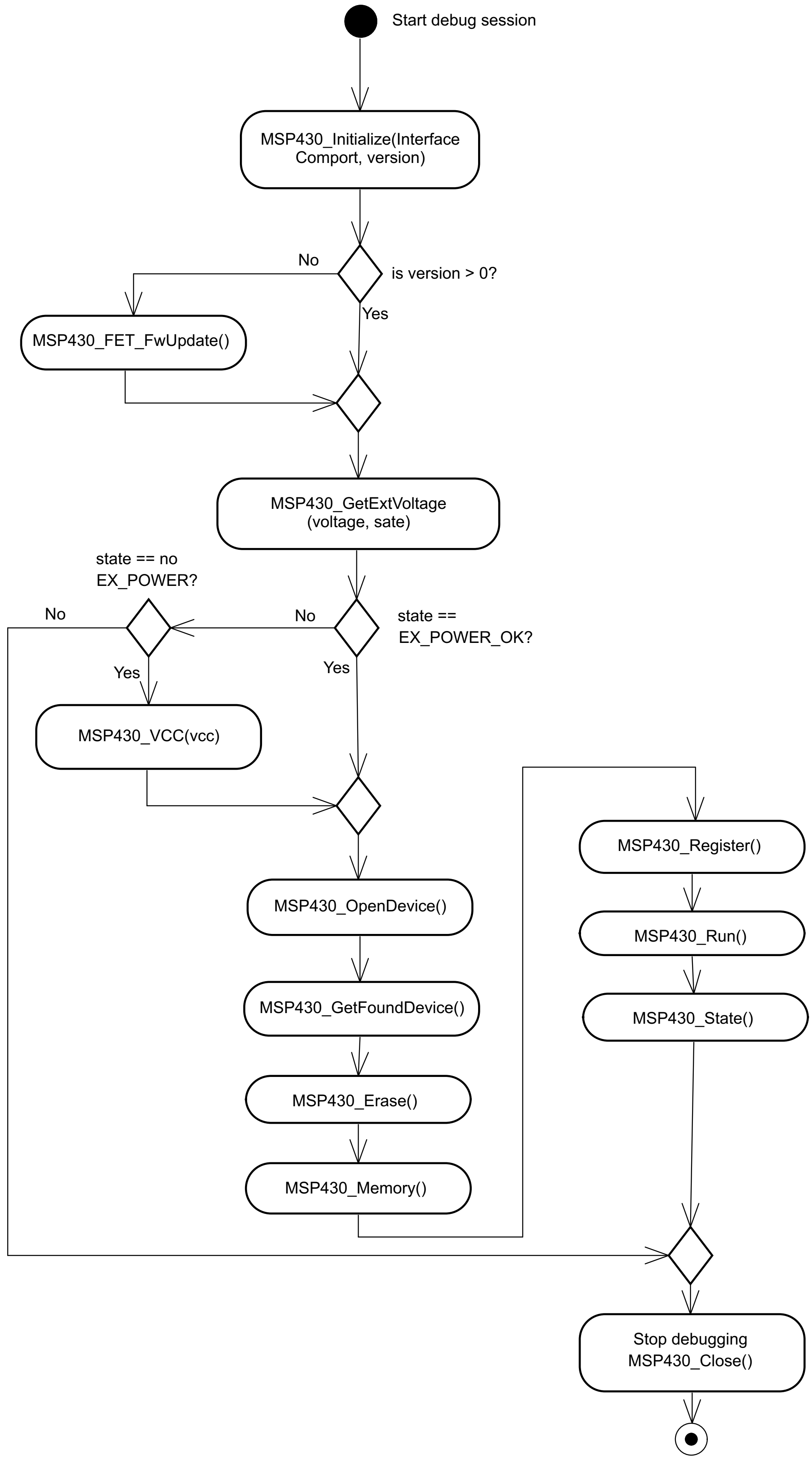


**Figure 1. Recommended Flow to Start an MSP430 Debug Session**

```
#include "stdio.h"
#include "MSP430_FET.h"
#include "MSP430_Debug.h"
#include "MSP430.h"

long lVersion;     // DLL version
long verify = 0;   // verify the filetransfer?

// init JTAG interface - TIUSB will use first connected debugger
printf("MSP430_Initialize()\n");
if(MSP430_Initialize("TIUSB", &lVersion) == STATUS_ERROR)
{
  printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number()));       // print error string
  MSP430_Close(1); // close the debug session
}
// Check firmware compatibility
if(lVersion < 0)         // firmware outdated?
{
  // perform firmware update
  printf("MSP430_FET_FwUpdate()\n");
  if(MSP430_FET_FwUpdate(NULL, NULL, NULL) == STATUS_ERROR)
  {
        printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number())); // print error string
        MSP430_Close(1); // close the debug session
  }
}
// power up the target device
printf("MSP430_VCC()\n");
if(MSP430_VCC(3000) == STATUS_ERROR)      // target VCC in millivolts
{
  printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number()));       // print error string
  MSP430_Close(1); // close the debug session
}
// configure interface - this is optional! automatic interface selection is the default
printf("MSP430_Configure()\n");
if(MSP430_Configure(INTERFACE_MODE, AUTOMATIC_IF) == STATUS_ERROR)
{
  printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number()));       // print error string
  MSP430_Close(1); // close the debug session
}
// open the device
printf("MSP430_OpenDevice()\n");
if(MSP430_OpenDevice("DEVICE_UNKNOWN","",0,0,DEVICE_UNKNOWN) == STATUS_ERROR)
{
  printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number()));       // print error string
  MSP430_Close(1); // close the debug session
}
// program .txt file into device memory (optional)
printf("MSP430_ProgramFile()\n");
if(MSP430_ProgramFile("C:\file.txt", ERASE_ALL, verify) == STATUS_ERROR)
{
  printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number()));       // print error string
  MSP430_Close(1); // close the debug session
}
/**************************** debug session is started ****************************/
```

**Figure 2. Code Example to Start a Debug Session**

## 3.2 Attach to a Running Device

The MSP430.dll can connect to a running MSP430 target device without stopping or changing the target program execution. This feature can be used for debugging an application that is already running on the target device. During this special start-up sequence, only the JTAG interface must be initialized. No reset of the target device is performed, because a reset could change the application context of the target device. The running application could contain various information of interest for the debug session (for example, error states of long run-time errors such as a stack overflow).

Establishing the physical JTAG connection to the target device is not trivial, especially when the RST signal of the target processor is connected to the JTAG header. A successful connection is subject to stable signals on the JTAG connector (a bouncing signal on the RST pin will perform a reset of the connected microcontroller).

**NOTE:** Attach to running target is only available with external power supply. Using the internal power supply that is generated by the USB-FET would reset the device during $V_{CC}$ supply start-up sequence.

Figure 3 shows the flow with the highest probability for successfully attaching to a running target. Figure 4 shows a code example that follows this flow.

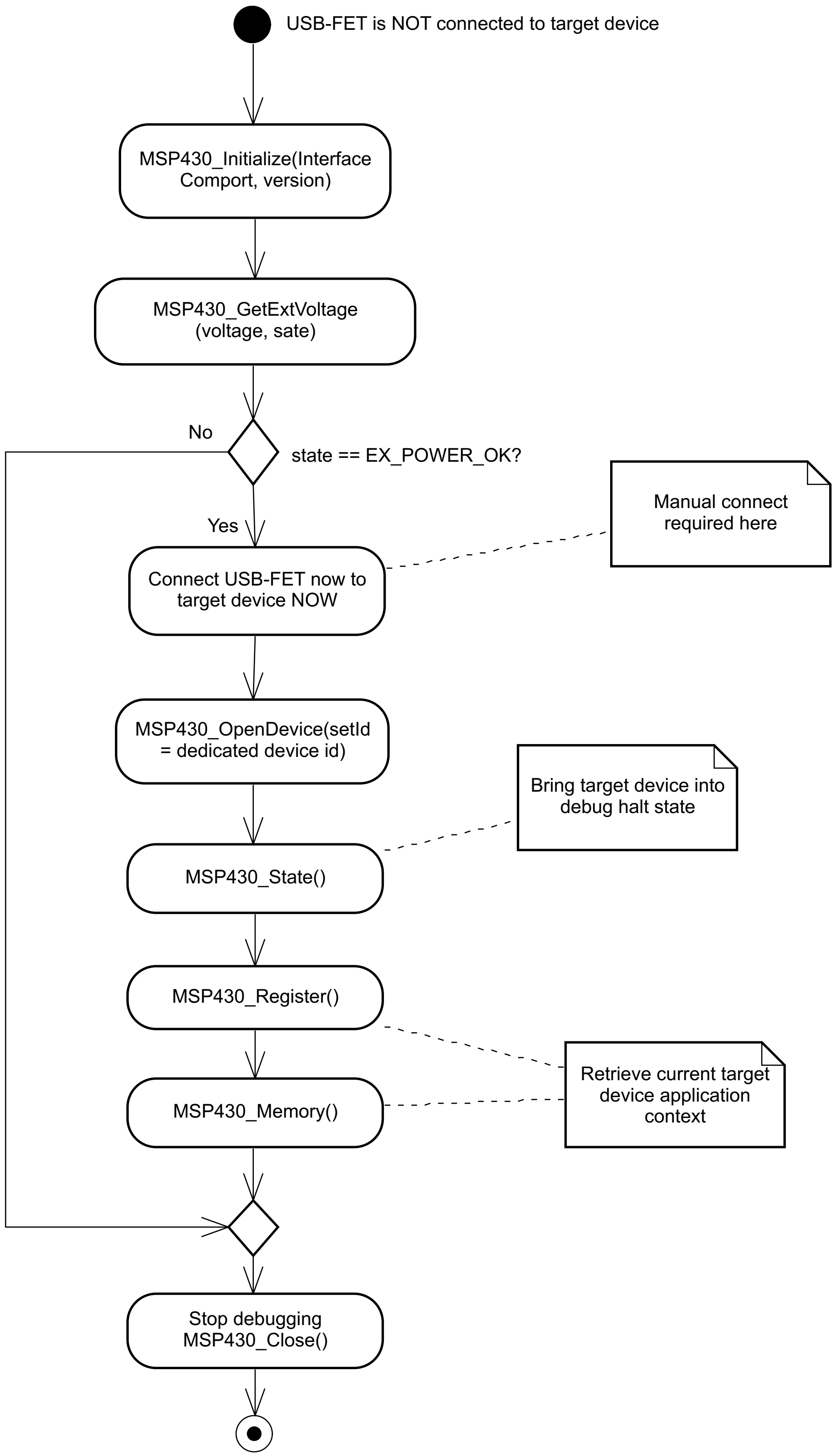


**Figure 3. Attach to Running target**

```
long lVersion, state, pCpuCycles;
DEVICE_T TargetDevice;

// get device information - determine device id
printf("MSP430_GetFoundDevice()\n");
if(MSP430_GetFoundDevice((char*)&TargetDevice, sizeof(TargetDevice.buffer)) == STATUS_ERROR)
{
  printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
  MSP430_Close(1);
}

// release the target from JTAG control
printf("MSP430_Run(FREE_RUN, release from JTAG)\n");
if(MSP430_Run(FREE_RUN, TRUE) == STATUS_ERROR)
{
  printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
  MSP430_Close(1);
}

printf("MSP430_Close(VccOff = false)\n");// close the interface connection
if(MSP430_Close(FALSE) == STATUS_ERROR)  // do NOT turn off Vcc power supply
{
  printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
  MSP430_Close(1);
}

Sleep(100);  // wait a few milliseconds

// initialize the interface again
printf("MSP430_Initialize()\n");
if(MSP430_Initialize("TIUSB", &lVersion) == STATUS_ERROR)
{
  printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
  MSP430_Close(1);
}

// attach to the running target with correct device string and/or device id
printf("MSP430_OpenDevice(DeviceNameString,…,…, TargetDevice.id)\n");
if(MSP430_OpenDevice((char*)TargetDevice.string,"", 0, 0, TargetDevice.id) == STATUS_ERROR)
{
  printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
  MSP430_Close(1);
}

// check CPU state - state should be "RUNNING"
printf("MSP430_State(...,stop = FALSE,...) -> check CPU state\n");
if(MSP430_State(&state, FALSE, &pCpuCycles) == STATUS_ERROR)
{
  printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
  MSP430_Close(1);
}
```

NOTE: Open a debug session before using this code (see Figure 2).

**Figure 4. Code Example for "Attach to running target"**

### 3.3 Supporting Multiple USB-FET Debuggers

The MSP430.dll can support multiple USB-FET debuggers connected to the computer. For this purpose, two MSP430 USB-FET support functions are available inside the MSP430.h file:

- MSP430_GetNumberOfUsbIfs()
- MSP430_GetNameOfUsbIf()

Before calling MSP430_Initialize() (to open the COM port where the USB-FET is attached) the two functions above must be executed in the correct order.

- MSP430_GetNumberOfUsbIfs()
  - First find how many USB-FETs are connected to the PC system
- MSP430_GetNameOfUsbIf()
  - Get the name (for example, COM5, or COM19) and status of the CDC COM port that is assigned to a certain USB-FET debugger.

After all information about how many and which CDC ports are available on the PC system has been retrieved, a dedicated USB-FET tool can be employed directly by passing the CDC port name to MSP430_Initialize() function [for example, MSP430_Initialize("COM5",…)].

Figure 5 shows the typical flow, which is executed to retrieve all needed information about connected USB-FET tools/debuggers.

Figure 6 shows example code for initializing multiple USB-FET debuggers one by one.

Also see Section 6.4 for information on the MultipleUifs example project, which is an application implementation proposal.

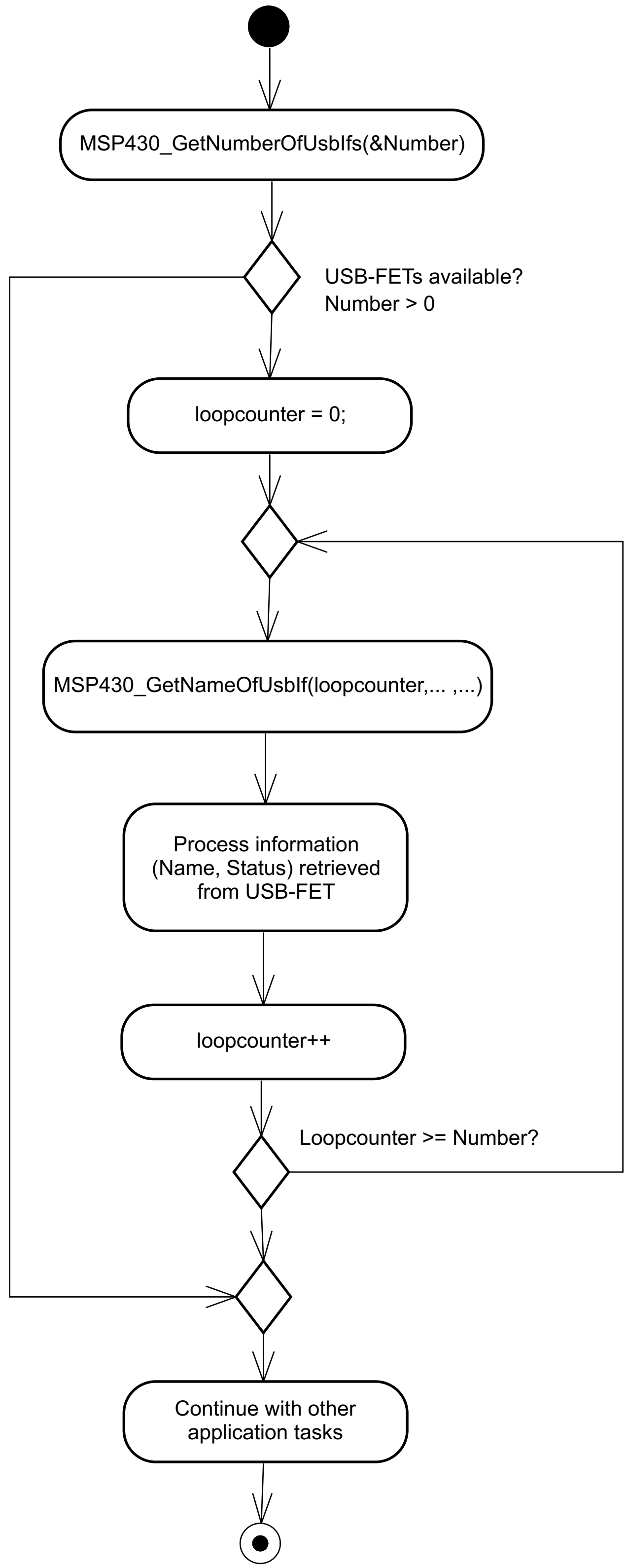


**Figure 5. Retrieve Information About Available USB-FETs or Debuggers**

```
#include "stdio.h"
#include "MSP430.h"

long number, count, status, lVersion, lErrorNumber;
char * name;

// determine the number of connected UIFs
printf("MSP430_GetNumberOfUsbIfs()\n");
if(MSP430_GetNumberOfUsbIfs(&number) == STATUS_ERROR)
{
    printf("Error: Could not determine number of UIFs!\n");
    lErrorNumber = MSP430_Error_Number();
    printf("Reason: %s\n", MSP430_Error_String(lErrorNumber));
}
else
{
    printf("Found %d UIF(s).\n", number);
    for(count = 0; count < number; count++)
    {
    // get the com port name
        printf("MSP430_GetNameOfUsbIf()\n");
        if(MSP430_GetNameOfUsbIf(count, &name, &status) == STATUS_ERROR)
        {
            printf("Error: Could not obtain com port name for UIF %d.\n", count+1);
            lErrorNumber = MSP430_Error_Number();
            printf("Reason: %s\n", MSP430_Error_String(lErrorNumber));
        }
        else
        {
            // initialize the interface
            printf("Initializing UIF @ %s.\n", name);
            printf("MSP430_Initialize(UIF %d)\n", count+1);
            if(MSP430_Initialize(name, &lVersion) == STATUS_ERROR)
            {
               lErrorNumber = MSP430_Error_Number();
               printf("Error: %s\n", MSP430_Error_String(lErrorNumber));
            }
            else
            {
               printf("Success!\n");

               // commence with debug session start here...

               // close the interface
               printf("MSP430_Close()\n");
               MSP430_Close(1);
            }
        }
    }
}
```

**Figure 6. Code Example for Communication With Multiple USB-FETs or Debuggers**

### 3.4 Configuring the JTAG Protocol

By default. the MSP430.dll is configured to perform an automatic protocol scan before starting communication with MSP430 devices. This default configuration can be overwritten manually by using the INTERFACE_MODE configuration (see MSP430.h file for details). Four different interface modes are available and can be used for debugging the connected MSP430 device.

- **JTAG_IF**: The normal standard 4-wire JTAG communication (not supported by eZ debuggers).
- **SPYBIWIRE_IF**: Spy-Bi-Wire (2-wire) JTAG protocol (not supported by deprecated PIF).
- **SPYBIWIREJTAG_IF**: Standard 4-wire JTAG communication for MSP430 devices that also support Spy-Bi-Wire (a special entry sequence is needed to switch these MSP430 derivatives into 4-wire mode which cannot be applied to any MSP430 devices) (not supported by eZ debuggers).
- **AUTOMATIC_IF**: JTAG communication protocol is selected automatically by the DLL (default).

If MSP430_Configure() is used to manually configure the JTAG protocol, it must be called before MSP430_OpenDevice() is called.

### 3.5 Speed up Flash Programming

The API routines MSP430_Erase() and MSP430_Memory() enable manipulation of the target devices Flash, RAM, and FRAM.

If flash memory is programmed by the DLL, the target device RAM is used by the DLL flash programming routines. Therefore, the RAM content of the target devices must be preserved before programming flash memory. After successfully programming, the original RAM content must be restored.

This RAM preservation mechanism allows flash memory manipulation during an active debug session without corrupting or changing any RAM content. However, the process requires noticeable time to preserve and restore RAM contents. Thus, this mechanism might not be useful under some circumstances; for example, during an initial flash programming at the beginning of a debug session.

Therefore, the RAM preserve and restore mechanism can be disabled by an additional MSP430_Configure() function call. This additional configuration mode is called RAM_PRESERVE_MODE.

The following sequence might be used, for example, for an initial flash programming sequence:

(1) MSP430_Configure(RAM_PRESERVE_MODE, DISABLE);

(2) MSP430_Erase(ERASE_ALL,..,..);

(3) MSP430_Memory(..., ..., ..., WRITE );

(4) MSP430_Memory(..., ..., ..., READ );

..... Flash programming or download finished

(n) MSP430_Configure(RAM_PRESERVE_MODE, ENABLE);

### 3.6 Controlling Device Program Execution

The MSP430.dll provides additional debugging functions to developers of third-party tools for the MSP430. The debugging functions include execution control (free run, run to breakpoints, single step, state, stop, set breakpoint), device control (read or write registers, reset, clock configuration, device configuration), and low-level access to the advanced features of the Enhanced Emulation Module (EEM) that provides such features as complex breakpoints and trace buffers. The low-level access to EEM registers (namely the read/write EEM register) is in the DLL for compatibility with the EEM API.

### 3.7 Enhanced Emulation Module (EEM) Access – EEM API

The MSP430.dll provides an enhanced debug API that allows access to the MSP430 Enhanced Emulation Module functionality. Refer to the source code of the application examples for details on how to use the EEM API.

**NOTE:** Some deprecated API functions are no longer allowed to be called when the EEM API is used. These functions are:

- MSP430_Configure() with parameter 'mode' set to CLK_CNTRL_MODE
- MSP430_Configure() with parameter 'mode' set to MCLK_CNTRL_MODE
- MSP430_State() with parameter 'stop' set to FALSE
- MSP430_EEM_Open()
- MSP430_EEM_Read_Register()
- MSP430_EEM_Read_Register_Test()
- MSP430_EEM_Write_Register()
- MSP430_EEM_Close()

Refer to the detailed documentation in MSP430_EEM.h.

### 3.8 Error Handling

Most functions of the MSP430.dll return an indication of success (STATUS_OK) or failure (STATUS_ERROR). If STATUS_ERROR is returned, MSP430_Error_Number() can be used to obtain a detailed error code. MSP430.h contains an enumeration of all error codes, and lists the error codes returned by each DLL function. MSP430_Error_String() returns the string corresponding to the error code parameter.

STATUS_ERROR is returned at the first error condition. The DLL typically does not attempt to retry or recover from the error condition. It is the responsibility of the application to retry the failed operation and possibly to implement a recovery mechanism.

### 3.9 Miscellaneous

The MSP430.dll is a partially intrusive tool–accessing the device through JTAG can affect the device (for example, the clocking of the watchdog mechanism). However, steps are taken within the DLL to minimize the effects upon the device caused by JTAG.

**CAUTION**

**Do not unplug the JTAG cable during an active debug session!** This might cause unknown device behavior!

## 4 MSP-FET430UIF Firmware Update Support

With every new version of MSP430.dll, the firmware of the connected USB JTAG interface needs to be updated. The MSP430.dll includes a binary image of the corresponding USB-FET debugger firmware. Calling MSP430_FET_FwUpdate() as described in Figure 7 assures consistency between USB-FET firmware and MSP430.dll.

With this release of the DLLv3, the firmware has changed and now consists of different independent parts (USB communication core, JTAG stack, low-level debugger hardware access, $V_{CC}$ generation, and UART backchannel) that can be updated independently. Therefore, it was necessary to extend the firmware update mechanism to execute DLLv2 to DLLv3 updates. Figure 7 shows that MSP430_Initialize() returns either -3, -2 , -1 or the actual DLL firmware version.

If MSP430_Initialize() returns -3, a major firmware version update (DLLv2 to DLLv3) is required. Afterward this update is complete, call MSP430_FET_FwUpdate() again to update the firmware with the DLL internal binary image. In this special update case, a given update file is ignored.

If MSP430_Initialize() returns -1, the USB-FET firmware has been already updated to DLLv3 firmware. In this case, the communication core, JTAG stack, or HIL module does not match the DLL version. By calling the MSP430_FET_FwUpdate() function, the internal DLL binary images are used for the USB-FET update.

If MSP430_Initialize() returns -2, the USB-FET firmware needs recovery because of major system corruption. A corrupted USB-FET always enumerates as HID-FET. The MSP Debug Stack detects the HID-FET and raises a message that a connected USB-FET needs recovery. See Figure 8 for details on the USB-FET HID recovery flow.

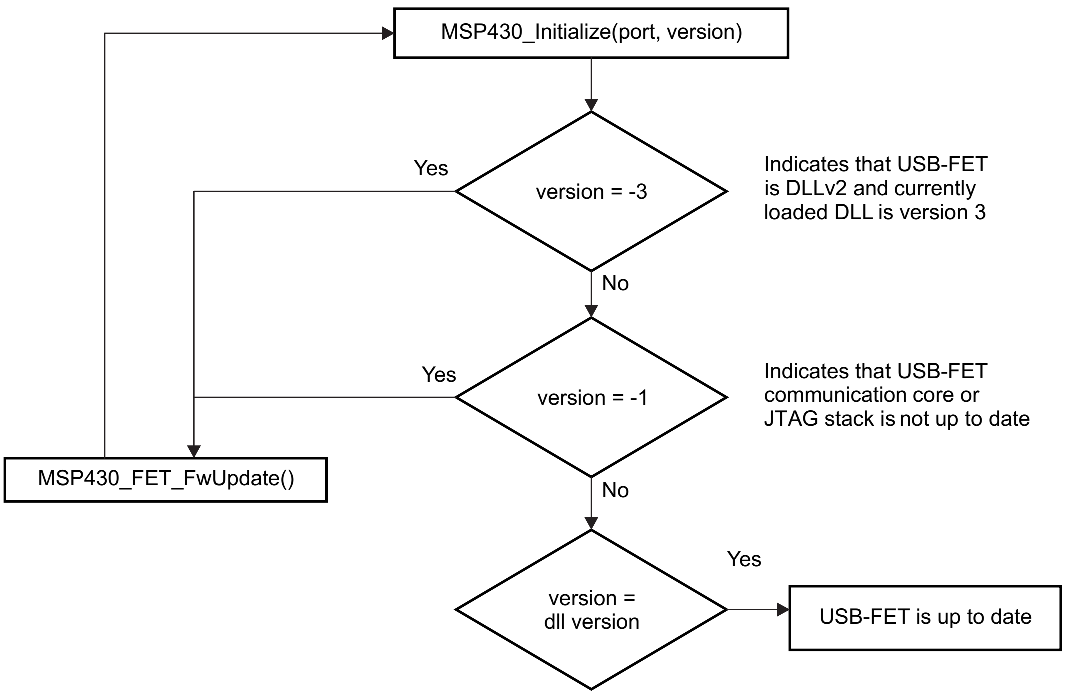


**Figure 7. General Firmware Update Flow**

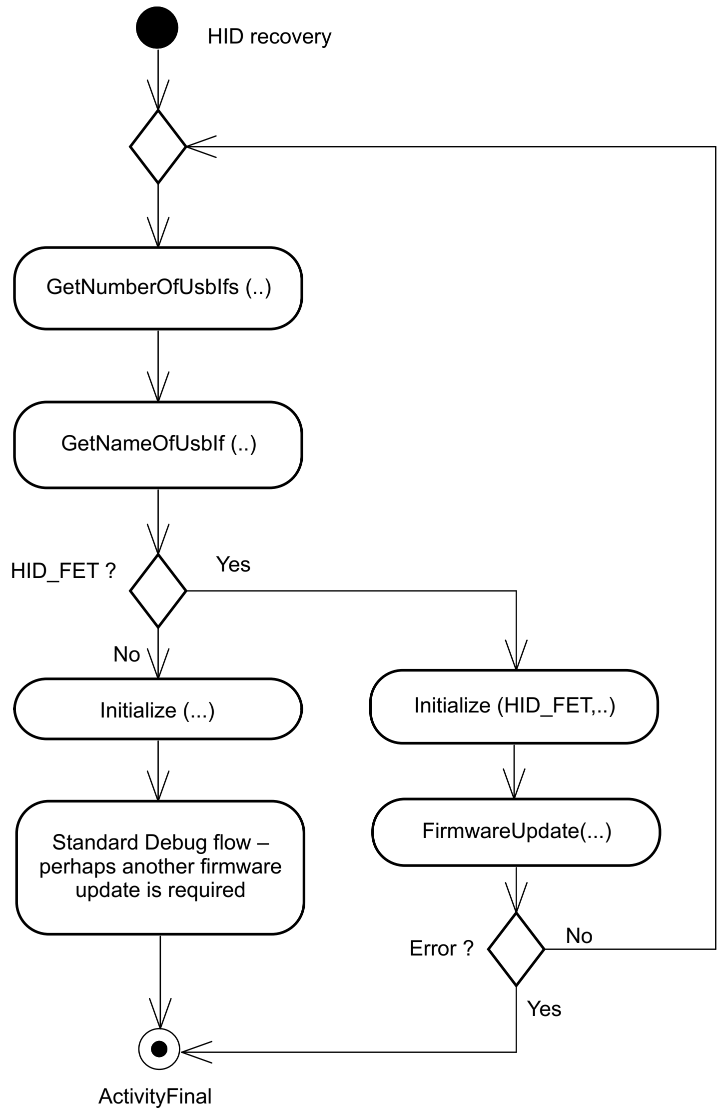


**Figure 8. USB-FET HID Recovery Flow**

### 4.1 Firmware Update With Update Tool

An update of the MSP-FET430UIF firmware without an IDE can be executed using the command line based Update Tool.

The Update Tool can also upgrade or downgrade the firmware between major firmware versions.

For detailed information how to update the MSP-FET430UIF, refer to http://processors.wiki.ti.com/index.php/MSPDS_Debugger_Up-_and_Downgrade

---

**NOTE:** See Section 8 for details on how to determine if you are using MSP-FET430UIF revision 1.3, because it requires additional update steps (see Section 4.2).

---

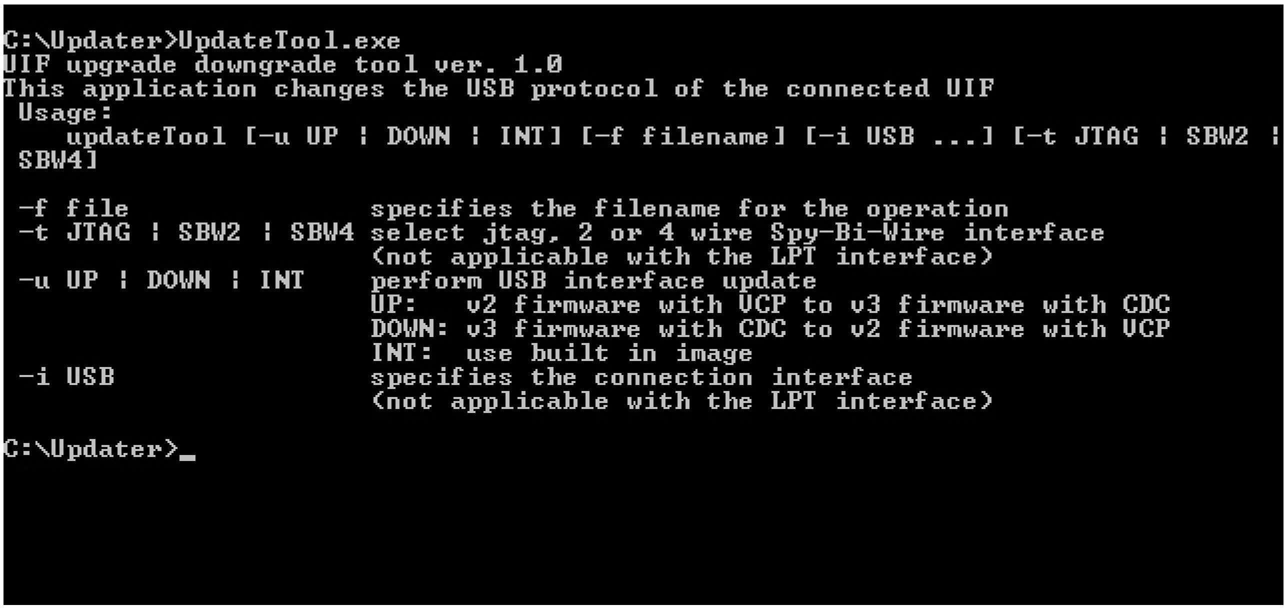


**Figure 9. Update Tool**

#### Available Commands

updateTool –u **UP**: Updates the UIF major firmware version (for example, version 2 to 3).

updateTool –u **DOWN**: Downgrades the UIF major firmware version through the binary image stored in Uifv3Downgrader.txt.

updateTool –u **INT**: Updates the UIF with the DLL internal firmware image.

---

**NOTE:** Make sure that the CDC driver is already installed before performing a major firmware update. Also, a file named CDC.log with the content "True" must be in the same folder as the DLL to indicate that the CDC driver was installed successfully. If this file is not found or does not contain this content, the update process returns an update error.

---

### 4.2 Additional Update Step for MSP-FET430UIF Hardware Revision 1.3

After calling updateTool –u UP, the update process starts. Figure 10 shows the command line window that is displayed.

```
C:\Updater>UpdateTool -u UP
    Initialize: done
    MSP430_FET_GetFwVersion()
    Firmware Version: 20409001
Status: Starting firmware update with built in image!

    Initializing bootloader...
    Erasing interrupt vectors...
    Erasing firmware...
    Programming new firmware...
    100 percent done

    Finishing...
```

**Figure 10. Start Firmware Update**

When update is complete, the TUSB3410 is typically reset, and the UIF shows as a CDC device. However, as described in Section 8, the TUSB3410 cannot be reset in MSP-FET430UIF revision 1.3, so it is necessary to disconnect the MSP-FET430UIF and reconnect it. After doing so, the update process continues (see Figure 11).

```
    Initializing bootloader...
    Erasing firmware...
    Programming new firmware...
    100 percent done

    Finishing...
    Update complete.


    Update complete.


 Status: Firmware update performed successfully
```

**Figure 11. Firmware Update Successful**

## 5 Supporting eZ430 Emulator Dongles

Several different versions of the eZ430 emulator are available.

- eZ430-RF2500: The dongle enumerates as a Human Interface Device (HID). The HID class driver is part of the Windows operation system, thus the enumeration does not require any user interaction. The HID interface is used for the JTAG communication to the target device.

  In addition to the HID channel, the dongle also tries to enumerate a Virtual COM Port (which is called MSP430 Application UART). This driver is based on the Communication Device Class (CDC) interface. This CDC driver class is also part of the Windows operating system but it requires an INF file for installation. The provided INF file (430CDC.inf, can be found in folder Driver/Inf) is certified for MS-Windows operating systems XP32, XP64, Vista32, Vista64, Win7-32, and Win7-64.

  The folder Driver/Inf contains a subfolder PreinstallCDC. This subfolder contains an example source code that shows how to install the INF file on a MS-Windows PC. TI recommends installing the INF file as described in the example. If the install is not done, the Windows Hardware Wizard opens as soon as the user connects the tools to the PC. The user must manually point the Wizard to the correct location of the INF file when using the wizard.

- Other supported eZ430 tools that make use of the HID interface are the eZ430 Chronos™ Wireless development tool in a watch, the MSP430 LaunchPad Value Line Development Kit, and the MSP-EXP430FR5739 FRAM experimenter board.

# 6 Application Examples

The DLL developer's package features a series of example projects that demonstrate the use of different DLL functions. After the rebuild, the executables can be found in the folder ApplicationExample/Executables. See the source code for details on how to call DLL functions and correctly pass parameters to those functions.

## *6.1 Example*

Example is a simple example project that demonstrates how the basic functions of the DLL are called to initialize the interface, identify and configure the device, manipulate the device memory (erase, program, verify, read), secure the device, reset the device, close the interface, and handle error conditions. Refer to the source file Example.c.

## *6.2 ExampleDebug*

ExampleDebug is an example project that demonstrates how the functions of the DLL are called to initialize the interface, identify and configure the device, manipulate the device memory (erase, program, verify, read), read the device registers, set device breakpoints, run the device (free, with breakpoints, single step), reset the device, close the interface, and handle error conditions. Refer to the source file Example Debug.c.

## *6.3 UifUpdate*

UifUpdate is an example project that demonstrates how to perform an USB-FET firmware update by calling MSP430_FET_FwUpdate() including handling of the notify callback mechanism during the update process. Refer to the DLL API documentation of MSP430_FET_FwUpdate() for details.

## *6.4 MultipleUifs*

MultipleUifs is an example project that demonstrates how to support multiple MSP-FET430UIF tools connected to one PC system. The example project comes with a GUI that shows a possible support implementation.

# 7 Installation of CDC for USB-FET Debuggers

The UIF tries to enumerate a Virtual COM Port, which is based on Communication Device Class (CDC) driver. This CDC driver class is part of the Windows operating system, but it requires an INF file for installation.

After plugging in the USB-FET, Windows recognizes a new hardware called MSP-FET430UIF or MSP Debug interface (see Figure 12).


**Figure 12. New Hardware**

Next, the hardware wizard opens a new dialog (see Figure 13).

- If Code Composer Studio™ (CCS) IDE version 5 or 6 or IAR Embedded Workbench™ IDE is already installed, select "Install the software automatically".
- If no IDE has been installed, select the msp430tools.inf, which is part of the developer's packet (MSP430_DLL_Developer_Package_Rev_x_x_x_x\Driver\CDC) and install the driver manually.

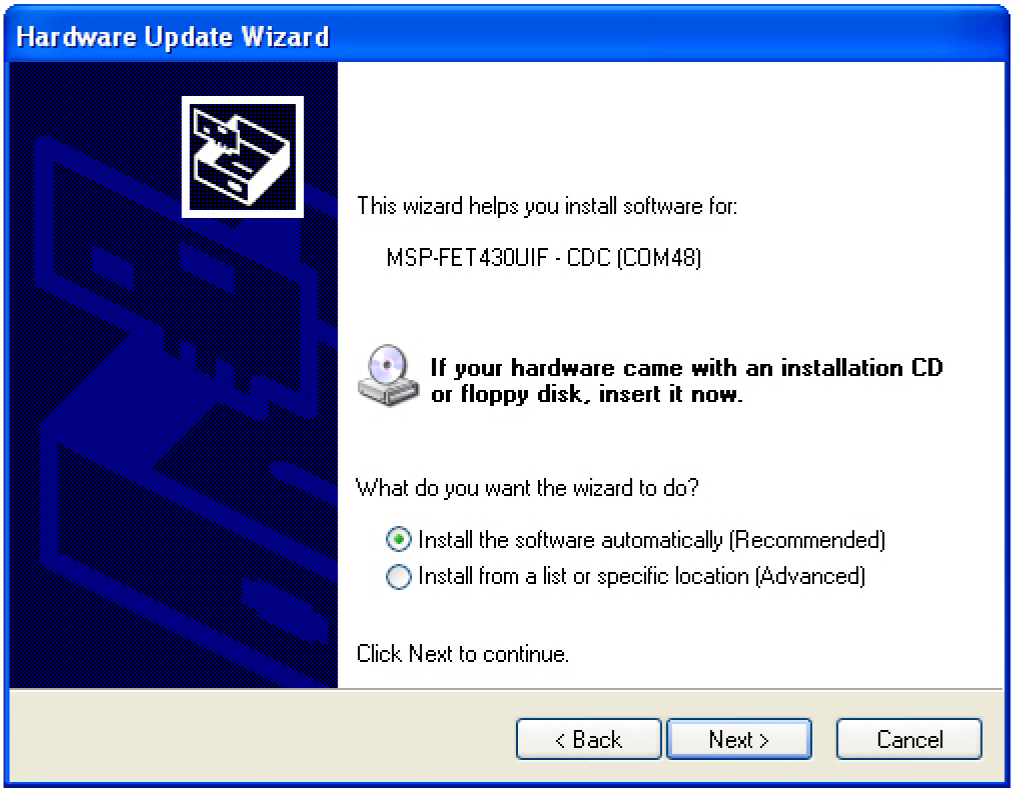


**Figure 13. Update Wizard**

## 8 Update MSP-FET430UIF With Hardware Revision 1.3

If you are using a MSP-FET430UIF with hardware revision 1.3, the update process includes one additional step, because it is not possible to reset the TUSB3410 USB port controller during firmware update.

Without a reset, the TUSB3410 cannot change the VCP protocol to CDC and then install the new communication core and JTAG stack. Therefore, it is necessary to reset the device manually by disconnecting the MSP-FET430UIF and connect it to the PC again.

For IDE-specific information on how to update an MSP-FET430UIF revision 1.3, See the MSP-FET430UIF Debug FAQ (CCS v5.1 or later and IAR EW v5.40 or later).

See Figure 14 and Figure 15 to determine if you are using MSP-FET430UIF with hardware revision 1.3. Figure 14 shows that revision 1.3 has a CE logo on the front and no label with a revision number on the back. Figure 15 shows that revision 1.4 does not have the CE logo on the front and does have a label with the version number on the back.


**Figure 14. UIF Revision 1.3**


**Figure 15. UIF Revision 1.4**

# 9 References

1. TUSB3410 RS232/IrDA Serial-to-USB Converter (http://www.ti.com/product/tusb3410)
2. MSP Debug Stack (http://www.ti.com/tool/MSPDS)

## IMPORTANT NOTICE



| **Products** | | **Applications** | |
|---|---|---|---|
| Audio | www.ti.com/audio | Automotive and Transportation | www.ti.com/automotive |
| Amplifiers | amplifier.ti.com | Communications and Telecom | www.ti.com/communications |
| Data Converters | dataconverter.ti.com | Computers and Peripherals | www.ti.com/computers |
| DLP® Products | www.dlp.com | Consumer Electronics | www.ti.com/consumer-apps |
| DSP | dsp.ti.com | Energy and Lighting | www.ti.com/energy |
| Clocks and Timers | www.ti.com/clocks | Industrial | www.ti.com/industrial |
| Interface | interface.ti.com | Medical | www.ti.com/medical |
| Logic | logic.ti.com | Security | www.ti.com/security |
| Power Mgmt | power.ti.com | Space, Avionics and Defense | www.ti.com/space-avionics-defense |
| Microcontrollers | microcontroller.ti.com | Video and Imaging | www.ti.com/video |
| RFID | www.ti-rfid.com | | |
| OMAP Applications Processors | www.ti.com/omap | **TI E2E Community** | e2e.ti.com |
| Wireless Connectivity | www.ti.com/wirelessconnectivity | | |


Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265