

Bootloader (BSL) Scripter

The BSL Scripter is a command line tool to communicate with the bootloader (BSL) on an MSP430™ or SimpleLink™ MSP432™ microcontroller.

Contents

1	Introduction	3
2	Script Mode	4
2.1	Scripter File Format	4
2.2	BSL Scripter Usage	4
2.3	BSL Command Convention	5
2.4	BSL Scripting Language	5
2.5	Generate Firmware Image in TI TXT and Intel Hex Format	25
3	Command Line Interface Mode.....	28
3.1	Comparison Between Script Mode and Command Line Interface Mode	28
4	Using BSL Scripter With Communication Bridge Tools	35
4.1	Invoke Sequence	35
4.2	Example	36
5	Frequently Asked Questions (FAQ).....	39

List of Figures

1	BSL Scripter as Part of BSL Ecosystem.....	3
2	Ethernet Communication	7
3	CCS – Project Properties	25
4	CCS – Enable MSP430 Hex Utility	25
5	CCS – Select Output Format	26
6	CCS – Output File.....	26
7	IAR – Project Options.....	27
8	IAR – Select Output Format	27
9	IAR – Output File.....	27
10	Connection of MSP-FET and MSP430FR5739 LaunchPad™ Development Kit Using Jumper Wires	36
11	Device Manager	37
12	Open MSP Tools Driver Properties	37
13	MSP Tools Driver Properties	37
14	Connection of MSP-BSL and MSP430FR5739 LaunchPad™ Development Kit Using Jumper Wires	38
15	High-Level View of the BSL System	39
16	BSL Application.....	40
17	Two Different Blink Applications.....	40

List of Tables

1	BSL Commands Supported by Device Family.....	5
2	Comparison of Script Mode and Command Line Mode Commands	28
3	Connection of MSP-FET and MSP430FR5739 LaunchPad™ Development Kit	36
4	Connection of MSP-BSL and MSP430FR5739 LaunchPad™ Development Kit	38
5	JTAG/SBW Lock on F5xx and F6xx Devices.....	41

6	JTAG/SBW Lock on FR5xx and FR6xx Devices.....	41
---	---	----

Trademarks

MSP430, SimpleLink, MSP432, LaunchPad, Code Composer Studio are trademarks of Texas Instruments.
OS X is a registered trademark of Apple Inc.
Ubuntu is a registered trademark of Canonical Limited.
IAR Embedded Workbench is a registered trademark of IAR Systems.
Linux is a registered trademark of Linus Torvalds.
Windows is a registered trademark of Microsoft Corporation.
All other trademarks are the property of their respective owners.

1 Introduction

The BSL Scripter is a command line tool to communicate with the bootloader (BSL) on an MSP430 or MSP432 microcontroller. By using BSL-Scripter, the user can communicate with the BSL application that is programmed on the MSP430 or MSP432 target device to program the memory or update the firmware. As shown in [Figure 1](#), the BSL Scripter is a PC host programmer that transforms a firmware image into a certain package and deploys it to the communication bridge, which then programs the MSP target device memory.

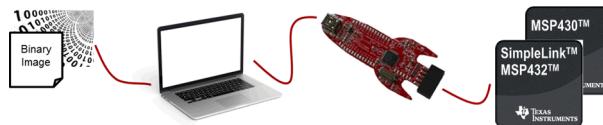


Figure 1. BSL Scripter as Part of BSL Ecosystem

For a more detailed overview of the MSP BSL Ecosystem, see these online videos training provided by TI:

- [MSP BSL overview](#)
- [MSP BSL options](#)

The [BSL Scripter package](#) provides the BSL Scripter application for Windows® (starting from Windows 7), Ubuntu® 32 bit, Ubuntu 64 bit (starting from Ubuntu 14), and OS X® (starting from OS X 10.9) operating systems. The source code is also provided for custom development by the user, if required. Example cases for the different families that the BSL-Scripter supports are also provided to show how to use the BSL Scripter.

NOTE: The BSL Scripter does not support the BSL of the MSP430 1xx, 2xx, and 4xx device families. To communicate with these devices, use the BSLDEMO2.exe command line tool found in the Deprecated folder.

The latest release of BSL Scripter supports:

- MSP430 flash family: MSP430F5xx and MSP430F6xx devices
- MSP430F543x family: MSP430F543x devices
- MSP430 FRAM family: MSP430FRxx devices
- Crypto-Bootloader (Crypto-BSL) (see [Crypto-Bootloader \(CryptoBSL\) for MSP430FR59xx and MSP430FR69xx MCUs](#))
- MSP432 family: MSP432P4xx and MSP432E4 devices

The BSL-Scripter version 3.0.0 and higher is tested using the following communication bridges:

- MSP-BSL Rocket (see [MSP-BSL Bootloader \(BSL\) Programmer for MSP430 and MSP432 MCUs](#))
- MSP-FET (see [MSP Debuggers](#))
- UART BSL interface on MSP430 LaunchPad™ development kits (see [LaunchPad-Based MSP430 UART BSL Interface](#))
- XDS110 backchannel UART on the [MSP432P401R LaunchPad development kit](#)

The BSL Scripter also supports the communication with USB BSL on MSP430 flash devices (F5xx and F6xx).

The application serves as a device programmer, a starting point for a custom BSL application (source code is included), and as a reference on how to use the BSL protocol (sent and received data can be observed when using the verbose mode).

Starting with BSL Scripter version 3.2.0, the BSL Scripter supports commands from a script file and also a command line interface.

2 Script Mode

2.1 Scripter File Format

The script file is a ASCII text file. The BSL Scripter reads this text file to parse the BSL commands. Each line must contain a single BSL command with no preceding spaces. Empty lines are allowed. Some commands may have optional parameters and some may have mandatory parameters. See [Section 2.4](#) for details on the BSL scripting language. In the description of each command, parameters that are shown in curly brackets must be substituted by the corresponding value. Lines in the script file can be commented out using C-style '//' as the first two characters on the line.

2.2 BSL Scripter Usage

The BSL Scripter can be started from the command line by typing the application name then typing the name of the script file to read. The folder of the script file is specified relative to the BSL-Scripter.exe folder.

Folder paths can be specified by any of these four methods:

- Same folder: The file to read is in the same folder as the script file
- Subfolder: The file to read is in a folder one or more levels down relative to the folder of the script file
- Up folder: The file to write is in a folder one or more levels up relative to the folder of the script file
- Absolute folder: The file to read is in a folder that is specified by an absolute path

If the `script_file.txt` is not specified, the program prompts for the file name.

Examples (Windows)

- Same folder

```
BSL-Scripter.exe .\script_file.txt  
BSL-Scripter.exe script_file.txt
```

- Subfolder

```
BSL-Scripter.exe SubFolder\script_file.txt
```

- Up folder

```
BSL-Scripter.exe ..\script_file.txt
```

- Absolute folder

```
BSL-Scripter.exe C:\Data\ScriptFolder\script_file.txt
```

Examples (Linux® and OS X)

- Same folder

```
./BSL-Scripter.exe script_file.txt
```

- Subfolder

```
./BSL-Scripter.exe SubFolder/script_file.txt
```

- Up folder

```
./BSL-Scripter.exe ../script_file.txt
```

- Absolute folder

```
./BSL-Scripter /home/Document/ScriptFolder/script_file.txt
```

2.3 BSL Command Convention

The BSL command format is:

<Command_Name> <Parameter_1> <Parameter_2> ...

There are two types of parameter, the mandatory and the optional parameters. In this user's guide, the mandatory parameters is written in bold font and optional parameters is written with italic font.

<Command_Name> {**Parameter_1**} {Parameter_2}

Example

MODE {CRYPTO} {**FAMILY**} {PROTOCOL} {BAUDRATE} {COM} {PARITY}

2.4 BSL Scripting Language

Different device families support different commands. [Table 1](#) lists the commands supported by each device family. See the following sections for a description of each command.

Table 1. BSL Commands Supported by Device Family

Command	MSP Device Family					Section
	F5xx, F6xx	FRxx	P4xx	E4xx	CryptoBSL ⁽¹⁾	
MODE	✓	✓	✓	✓	✓	Section 2.4.1
CHANGE_BAUD_RATE	✓	✓	✓		✓	Section 2.4.2
CRC_CHECK	✓	✓	✓		✓	Section 2.4.3
CRC_CHECK_32			✓			Section 2.4.4
DELAY	✓	✓	✓	✓	✓	Section 2.4.5
ERASE_SEGMENT	✓		✓			Section 2.4.6
ERASE_SEGMENT_32			✓			Section 2.4.7
ERASE_BLOCK				✓ ⁽²⁾		Section 2.4.8
SET_PC	✓	✓	✓		✓	Section 2.4.9
SET_PC_32			✓	✓		Section 2.4.10
MASS_ERASE	✓	✓	✓	✓ ⁽²⁾	✓	Section 2.4.11
REBOOT_RESET			✓	✓	✓	Section 2.4.12
RX_DATA_BLOCK	✓	✓	✓		✓	Section 2.4.13
RX_DATA_BLOCK_32			✓	✓		Section 2.4.14
RX_DATA_BLOCK_FAST	✓	✓	✓	✓ ⁽²⁾	✓	Section 2.4.15
RX_SECURE_DATA_BLOCK					✓	Section 2.4.16
RX_PASSWORD	✓	✓			✓	Section 2.4.17
RX_PASSWORD_32			✓			Section 2.4.18
TOGGLE_INFO	✓					Section 2.4.19
TX_DATA_BLOCK	✓	✓	✓		✓	Section 2.4.20
TX_DATA_BLOCK_32			✓	✓ ⁽²⁾		Section 2.4.21
TX_BSL_VERSION	✓	✓			✓	Section 2.4.22
TX_BSL_VERSION_32			✓			Section 2.4.23
TX_BUFFER_SIZE	✓	✓				Section 2.4.24
LOG	✓	✓	✓	✓	✓	Section 2.4.25
VERBOSE	✓	✓	✓	✓	✓	Section 2.4.26
FACTORY_RESET			✓			Section 2.4.27

⁽¹⁾ The CryptoBSL is a custom bootloader application for MSP430FR59xx and MSP430FR69xx devices. For more information, see [Crypto-Bootloader \(CryptoBSL\) for MSP430FR59xx and MSP430FR69xx MCUs](#).

⁽²⁾ This command is only supported by MSP432E4 devices with USB DFU communication.

2.4.1 MODE

Command

Minimum configuration for serial (UART, I²C, and SPI) bootloader:

```
MODE {CRYPTO} {FAMILY} {PROTOCOL} {COM}
```

Other properties for serial bootloader:

```
{BAUDRATE} {PARITY} {DISABLE_AUTO_BAUD} {I2CAddr=<0xslaveAddr>}
```

Minimum configuration for Ethernet bootloader:

```
MODE {FAMILY} {PROTOCOL} {MAC=<macAddr>} {IPClient=<ipAddr>} {IPServer=<ipAddr>}
```

Description

Initializes the selected communication channel for a BSL session and invokes the BSL. This command also tells the PC application which communication protocol to use. If an MSP432Pxx device is used, the initial UART baud rate can be specified. For all other devices, use the CHANGE BAUD RATE command to select a different UART baud rate.

Parameters

CRYPTO[optional]

This new feature is added to support communication with CryptoBSL devices.

FAMILY [mandatory]

- 543x family or msp430f543x: Indicates communication with one of the following devices:
MSP430F5418, MSP430F5419, MSP430F5435, MSP430F5436, MSP430F5437, MSP430F5438

NOTE: 54xxA devices are included in the '5xx' family.

- 5xx or msp430f5xx: Indicates communication with all other 5xx devices
- 6xx or msp430f6xx: Identical to '5xx' and can be used interchangeably
- FRxx or msp430frxx: Indicates communication with FRAM devices
- P4xx or msp432p4xx: Indicates communication with MSP432P4xx devices
- E4xx or msp432e4xx: Indicates communication with MSP432E4 devices

PROTOCOL [mandatory]

- UART: standard communication on most MSP430 devices, default baud rate: 9600 baud
- I²C: default baud rate: 100000 bit/s
- SPI: default baud rate: 125000 Hz
- USB
- Ethernet

BAUDRATE [optional]

- UART [optional, MSP432P4xx and MSP432E4 only]
 - 9600
 - 19200
 - 38400
 - 57600
 - 115200
- I²C
 - 100000
 - 400000

- SPI
 - 125000
 - 250000
 - 500000
 - 1000000

COM [mandatory]

- COM{x}: Indicates the PC COM port to use (for example, COM42)

PARITY [optional]

This mode selects the parity of UART communication. The default setting in the BSL Scripter is without parity. When the PARITY is set in the MODE command, the BSL Scripter runs the UART communication with even-parity format. Most of the BSL devices communicate with the even-parity format. Only the E4xx family BSL communicates with the no-parity format.

With the presence of MSP-BSL (Rocket) or MSP-FET, the even-parity format is generated by these bridge devices. When using the MSP432 LaunchPad development kit that has a backchannel UART as communication bridge, the even-parity should be generated by the Scripter. BSL Scripter v3.1.0.x does not support communication with MSP430 BSL devices without an MSP-BSL (Rocket) or MSP-FET.

Ethernet Configuration

BSL Scripter starts from version 3.3.0 support the Ethernet communication for MSP432E4 family. The protocol follows the Bootstrap Protocol (BOOTP) with Trivial File Transfer Protocol (TFTP) data transmission format.

The mandatory parameters are the MAC address of BSL target and IP Client Address that will be assigned into BSL target. If the PC has more than one local IP address (for example, having eth0 and wlan0 at the same time), the IP Server parameter could be configured to give specific local IP address that will be used as the host local IP address.

[Figure 2](#) shows the communication between host and the BSL target.

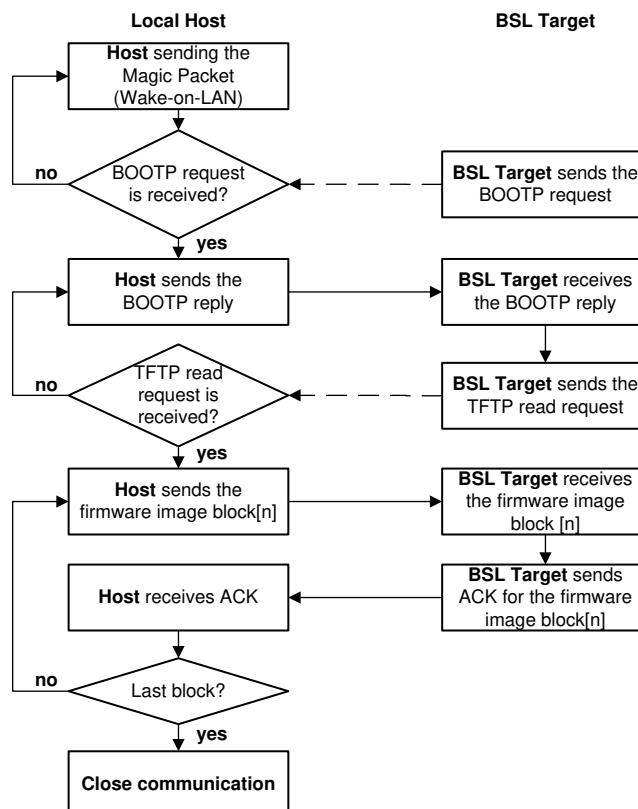


Figure 2. Ethernet Communication

Examples (Windows)

```
MODE 543x_family COM42
MODE msp430f543x COM42
MODE 5xx UART COM42
MODE msp430f5xx UART COM42
MODE 5xx USB
MODE msp430f5xx USB MODE FRxx I2C COM42
MODE msp430frxx I2C COM42
MODE P4xx SPI 250000 COM42
MODE msp432p4xx SPI 250000 COM42
MODE P4xx UART 115200 COM42
MODE msp432p4xx UART 115200 COM42
MODE Crypto FRxx UART COM15
MODE E4xx Ethernet MAC=08:00:28:5a:8b:7f IPClient=137.167.191.147
IPServer=137.167.191.146
```

The Scripter supports upper and lower case alphabets. Use the proper serial port name in Linux and OS X.

Examples (Linux and OS X)

```
MODE 543x_family /dev/ttyACM0
MODE msp430f543x /dev/ttyACM0
MODE P4xx I2C 100000 /dev/ttys0.usbmodem1451
MODE Crypto FRxx UART 9600 /dev/ttys0.usbmodem1451
MODE E4xx Ethernet MAC=08:00:28:5a:8b:7f IPClient=137.167.191.147
IPServer=137.167.191.146
```

2.4.2 CHANGE_BAUD_RATE

Command

```
CHANGE_BAUD_RATE {SPEED}
```

Description

Changes the baud rate of the UART communication.

Parameters

SPEED [*mandatory*]

Specifies a new baud rate, from one of the following values.

- UART
 - 9600
 - 19200
 - 38400
 - 57600
 - 115200
- I²C
 - 100000
 - 400000
- SPI
 - 125000
 - 250000
 - 500000
 - 1000000

NOTE: See the individual BSL descriptions in the device family user guides for information on supported baud rates and protocols.

Examples

```
CHANGE_BAUD_RATE 115200
CHANGE_BAUD_RATE 9600
```

2.4.3 CRC_CHECK

Command

CRC_CHECK {ADDRESS} {LENGTH} {EXPECTED}

Description

Performs a CRC starting at the given address over the number of bytes given by length. This command can either output the result of the CRC operation or compare the result to a supplied value and report whether there is a match or mismatch.

Parameters

ADDRESS [*mandatory*]

The start address for the CRC (hex format)

LENGTH [*mandatory*]

The number of bytes to include in the CRC (hex format)

EXPECTED [*optional*]

The expected value to verify the CRC result against

Examples

```
CRC_CHECK 0x8000 0x1000 0xCFB8
```

```
CRC_CHECK 0x8000 0x10
```

2.4.4 CRC_CHECK_32

Command

CRC_CHECK_32 {ADDRESS} {LENGTH} {EXPECTED}

Description

Identical to CRC_CHECK except that the memory is 32-byte addressed. This command supports MSP432P4xx devices only.

Parameters

ADDRESS [*mandatory*]

The start address for the CRC (hex format)

LENGTH [*mandatory*]

The number of bytes to include in the CRC (hex format)

EXPECTED [*optional*]

The expected value to verify the CRC result

Examples

```
CRC_CHECK_32 0x8000 0x1000 0xCFB8
```

```
CRC_CHECK_32 0x12345678 0x1010 0xFF31
```

2.4.5 DELAY

Command

DELAY {MS}

Description

Delay for the specified number of milliseconds.

Parameters

MS [*mandatory*]

The number of milliseconds to wait before proceeding

Examples

DELAY 1000

2.4.6 ERASE_SEGMENT

Command

ERASE_SEGMENT {ADDRESS}

Description

Causes the BSL to erase the segment containing the supplied address.

Parameters

ADDRESS [*mandatory*]

An address in hex format within MSP430 or MSP432 flash. The segment that contains this address is erased.

Examples

ERASE_SEGMENT 0x10000

ERASE_SEGMENT 0x8000

2.4.7 ERASE_SEGMENT_32

Command

ERASE_SEGMENT_32 {ADDRESS}

Description

Identical to ERASE_SEGMENT except that the memory is 32-byte addressed. This command supports only MSP432 MCUs.

Parameters

ADDRESS [*mandatory*]

An address in hex format within MSP432 flash. The segment that contains this address is erased.

Examples

ERASE_SEGMENT_32 0x10000

ERASE_SEGMENT_32 0x8000

2.4.8 ERASE_BLOCK

Command

ERASE_BLOCK {ADDRESS} {NUMBER_OF_BLOCKS}

Description

Similar functionality to ERASE_SEGMENT_32, but the parameter of number of blocks (segments) must be given. This command is supported by MSP432E4 MCUs with USB protocol.

Parameters

ADDRESS [*mandatory*]

An address in hex format within MSP432E4 main memory flash. BSL Scripter divides the memory by 1024 bytes for the block address alignment.

NUMBER_OF_BLOCKS [*mandatory*]

The number in hex format of flash blocks to erase. In MSP432E4 MCUs, each flash block is 16KB.

Examples

ERASE_BLOCK 0x4000 0x1

2.4.9 SET_PC

Command

SET_PC {ADDRESS}

Description

Sets the program counter to the given address and starts program execution at that address.

NOTE: SET_PC performs a function call to this address, so it can be returned from the call through a BSL action function.

Parameters

ADDRESS [*mandatory*]

The address to which the Program Counter of the MSP430 or MSP432 MCU is set.

Examples

SET_PC 0x2504

2.4.10 SET_PC_32

Command

SET_PC_32 {ADDRESS}

Description

Identical to SET_PC except that the address is 32-byte addressed. This command supports only MSP432 MCUs.

Parameters

ADDRESS [*mandatory*]

The address to which the Program Counter of the MSP432 MCU is set.

Examples

SET_PC_32 0x1FF16540

2.4.11 MASS_ERASE

Command

MASS_ERASE

Description

Performs a mass erase.

NOTE: For MSP430 flash and MSP432P4 devices, this command erases only the main memory of the device, not the information memory. To erase the information memory, use the [ERASE_SEGMENT](#) or [ERASE_SEGMENT_32](#) command.

Parameters

None

Examples

MASS_ERASE

2.4.12 REBOOT_RESET

Command

REBOOT_RESET

Description

Performs a reboot reset. This command is supported by MSP432 MCUs and the Crypto-Bootloader.

Parameters

None

Examples

REBOOT_RESET

2.4.13 RX_DATA_BLOCK

Command

```
RX_DATA_BLOCK {FILENAME}
```

Description

Reads the supplied TI TXT file or Intel Hex file, and downloads all data contained in this file to the MSP430 or MSP432 MCU.

NOTE: The memory writing is applicable only for flash and FRAM main and information memory sections. For flash memory, erase the memory before writing if the memory was already programmed. Writing to ROM is not supported.

Parameters

FILENAME [*mandatory*]

The name of the TI TXT or Intel Hex file to read. The path of the file to read is relative to the location of the script file:

- Same folder: The file to read is in the same folder as the main script file
- Subfolder: The file to read is in a folder one or more levels down relative to the folder that contains the script file
- Up folder: The file to read is in a folder one or more levels up relative to the folder that contains the script file
- Absolute folder: The file to read is in a folder that is specified by an absolute path

Example (Windows)

- Same folder

```
RX_DATA_BLOCK Big_File.txt  
RX_DATA_BLOCK .\Big_File.txt
```

- Subfolder

```
RX_DATA_BLOCK SubFolder\Big_File.txt
```

- Up folder

```
RX_DATA_BLOCK ..\Big_File.txt
```

- Absolute folder

```
RX_DATA_BLOCK C:\Data\ScriptFolder\Big_File.txt
```

Example (Linux or OS X)

- Same folder

```
RX_DATA_BLOCK Big_File.txt
```

- Subfolder

```
RX_DATA_BLOCK SubFolder/Big_File.txt
```

- Up folder

```
RX_DATA_BLOCK ../Big_File.txt
```

- Absolute folder

```
RX_DATA_BLOCK /home/Data/ScriptFolder/Big_File.txt
```

2.4.14 RX_DATA_BLOCK_32

Command

```
RX_DATA_BLOCK_32 {FILENAME}
```

Description

Reads the supplied TI TXT file or Intel Hex file, and downloads all data contained in this file to the MSP432 MCU. The command supports full 32-bit address space and supports only MSP432 MCUs.

NOTE: The flash memory is not rewritable, so the firmware upgrade requires proper erase memory.

Specific for the MSP432E4 serial bootloader, the mechanism of sending the data is based on segment size. If the firmware image has several blocks in one specific segment, it is padded with 0xFFh to be one packet.

Specific for MSP432E4 Ethernet bootloader, the firmware image is transformed into one complete block of data padded with 0xFFh for any gap between addresses.

Writing ROM is not supported with the RX_DATA_BLOCK_32 command.

Parameters

FILENAME [*mandatory*]

The name of the TI TXT or Intel Hex file to read. The path of the file to read is relative to the location of the script file:

- Same folder: The file to read is in the same folder as the main script file
- Subfolder: The file to read is in a folder one or more levels down relative to the folder that contains the script file
- Up folder: The file to read is in a folder one or more levels up relative to the folder that contains the script file
- Absolute folder: The file to read is in a folder that is specified by an absolute path

Example (Windows)

- Same folder

```
RX_DATA_BLOCK_32 Big_File.txt  
RX_DATA_BLOCK_32 .\Big_File.txt
```

- Subfolder

```
RX_DATA_BLOCK_32 SubFolder\Big_File.txt
```

- Up folder

```
RX_DATA_BLOCK_32 ..\Big_File.txt
```

- Absolute folder

```
RX_DATA_BLOCK_32 C:\Data\ScriptFolder\Big_File.txt
```

Example (Linux or OS X)

- Same folder

```
RX_DATA_BLOCK_32 Big_File.txt
```

- Subfolder

```
RX_DATA_BLOCK_32 SubFolder/Big_File.txt
```

- Up folder

```
RX_DATA_BLOCK_32 ../Big_File.txt
```

- Absolute folder

```
RX_DATA_BLOCK_32 /home/Data/ScriptFolder/Big_File.txt
```

2.4.15 RX_DATA_BLOCK_FAST

Command

```
RX_DATA_BLOCK_FAST {FILENAME}
```

Description

For MSP430 and MSP432P4 MCUs, the command is identical to RX_DATA_BLOCK except that no verification of programming is returned from the BSL. This is useful for USB programming only, and the BSL Scripter can only confirm that the file was sent. This command works only with 24-bit address space. For MSP432E4 MCUs, RX_DATA_BLOCK_FAST is based on the USB DFU standard protocol. Therefore, the the MSP432E4 responds to data transmission from BSL Scripter with specific USB DFU success and error messages.

NOTE: This command supports writing only to flash and FRAM main and information memory sections. For flash memory, erase the memory before writing if the memory was already programmed. Writing to ROM is not supported.

Parameters

FILENAME [*mandatory*]

The name of the TI TXT or Intel Hex file to read. The path of the file to read is relative to the location of the script file:

- Same folder: The file to read is in the same folder as the main script file
- Subfolder: The file to read is in a folder one or more levels down relative to the folder that contains the script file
- Up folder: The file to read is in a folder one or more levels up relative to the folder that contains the script file
- Absolute folder: The file to read is in a folder that is specified by an absolute path

Example (Windows)

- Same folder

```
RX_DATA_BLOCK_FAST Big_File.txt
RX_DATA_BLOCK_FAST .\Big_File.txt
```

- Subfolder

```
RX_DATA_BLOCK_FAST SubFolder\Big_File.txt
```

- Up folder

```
RX_DATA_BLOCK_FAST ..\Big_File.txt
```

- Absolute folder

```
RX_DATA_BLOCK_FAST C:\Data\ScriptFolder\Big_File.txt
```

Example (Linux or OS X)

- Same folder

```
RX_DATA_BLOCK_FAST Big_File.txt
```

- Subfolder

```
RX_DATA_BLOCK_FAST SubFolder/Big_File.txt
```

- Up folder

```
RX_DATA_BLOCK_FAST ../Big_File.txt
```

- Absolute folder

```
RX_DATA_BLOCK_FAST /home/Data/ScriptFolder/Big_File.txt
```

2.4.16 RX_SECURE_DATA_BLOCK

Command

```
RX_SECURE_DATA_BLOCK {FILENAME}
```

Description

Identical to RX_DATA_BLOCK, the RX_SECURE_DATA_BLOCK sends data to the device and receives back the verification of programming from the BSL. The difference from RX_DATA_BLOCK is that RX_SECURE_DATA_BLOCK contains the encrypted data or encrypted keys.

For additional information regarding the generation of an encrypted image file, see [Crypto-Bootloader \(CryptoBSL\) for MSP430FR59xx and MSP430FR69xx MCUs](#).

The BSL Scripter does not contain an encryption algorithm. It works as the transporter the encrypted data from PC to the CryptoBSL devices.

Parameters

FILENAME [*mandatory*]

The name of the encrypted TI TXT or Intel Hex file to read. The path of the file to read is relative to the location of the script file:

- Same folder: The file to read is in the same folder as the main script file.
- Subfolder: The file to read is in a folder one or more levels down relative to the folder that contains the script file.
- Up folder: The file to read is in a folder one or more levels up relative to the folder that contains the script file.
- Absolute folder: The file to read is in a folder that is specified by an absolute path.

Example (Windows)

- Same folder

```
RX_SECURE_DATA_BLOCK Encrypted_File.txt
RX_SECURE_DATA_BLOCK .\Encrypted_File.txt
```

- Subfolder

```
RX_SECURE_DATA_BLOCK SubFolder\Encrypted_File.txt
```

- Up folder

```
RX_SECURE_DATA_BLOCK ..\Encrypted_File.txt
```

- Absolute folder

```
RX_SECURE_DATA_BLOCK C:\Data\ScriptFolder\Encrypted_File.txt
```

Example (Linux or OS X)

- Same folder

```
RX_SECURE_DATA_BLOCK Encrypted_File.txt
```

- Subfolder

```
RX_SECURE_DATA_BLOCK SubFolder/Encrypted_File.txt
```

- Up folder

```
RX_SECURE_DATA_BLOCK ../Encrypted_File.txt
```

- Absolute folder

```
RX_SECURE_DATA_BLOCK /home/Data/ScriptFolder/Encrypted_File.txt
```

2.4.17 RX_PASSWORD

Command

```
RX_PASSWORD {FILENAME}
```

Description

Reads the supplied TI TXT file or Intel Hex file, and submits this data to the BSL as a password to unlock the device if the password is correct. If the password is wrong, a mass erase is performed.

NOTE: Although the same command is used to supply the password for the F543x family and other F5xx devices, the password file needs to be handled differently for these devices due to smaller password size in the F543x family. For details, see [MSP430 Programming With the Bootloader \(BSL\)](#).

NOTE: For USB BSL (on the device, not the full-featured RAM BSL for USB) without built in MASS_ERASE command, this command can be used with an incorrect password to perform a mass erase.

Parameters

FILENAME [*optional*]

The name of the TI TXT or Intel Hex file to read. When the input file is not inserted, the default password will be used by the Scripter.

The path of the file to read is relative to the location of the script file:

- Same folder: The file to read is in the same folder as the main script file.
- Subfolder: The file to read is in a folder one or more levels down relative to the folder that contains the script file.
- Up folder: The file to read is in a folder one or more levels up relative to the folder that contains the script file.
- Absolute folder: The file to read is in a folder that is specified by an absolute path.

Example (Windows)

- Same folder

```
RX_PASSWORD app_pass.txt
RX_PASSWORD .\app_pass.txt
```

- Subfolder

```
RX_PASSWORD SubFolder\app_pass.txt
```

- Up folder

```
RX_PASSWORD ..\app_pass.txt
```

- Absolute folder

```
RX_PASSWORD C:\Data\ScriptFolder\app_pass.txt
```

Example (Linux or OS X)

- Same folder

```
RX_PASSWORD app_pass.txt
```

- Subfolder

```
RX_PASSWORD SubFolder/app_pass.txt
```

- Up folder

```
RX_PASSWORD ../app_pass.txt
```

- Absolute folder

```
RX_PASSWORD /home/Data/ScriptFolder/app_pass.txt
```

2.4.18 RX_PASSWORD_32

Command

```
RX_PASSWORD_32 {FILENAME}
```

Description

Identical to RX_PASSWORD except that the default password has a length of 256 byte and can only be used for MSP432Pxx MCUs. The memory is 32-byte addressed. When the input file is not inserted, the default password will be used by the Scripter.

Parameters

FILENAME [optional]

The name of the TI TXT or Intel Hex file to read. The path of the file to read is relative to the location of the script file:

- Same folder: The file to read is in the same folder as the main script file.
- Subfolder: The file to read is in a folder one or more levels down relative to the folder that contains the script file.
- Up folder: The file to read is in a folder one or more levels up relative to the folder that contains the script file.
- Absolute folder: The file to read is in a folder that is specified by an absolute path.

Example (Windows)

- Same folder

```
RX_PASSWORD_32 app_pass.txt  
RX_PASSWORD_32 .\app_pass.txt
```

- Subfolder

```
RX_PASSWORD_32 SubFolder\app_pass.txt
```

- Up folder

```
RX_PASSWORD_32 ..\app_pass.txt
```

- Absolute folder

```
RX_PASSWORD_32 C:\Data\ScriptFolder\app_pass.txt
```

Example (Linux or OS X)

- Same folder

```
RX_PASSWORD_32 app_pass.txt
```

- Subfolder

```
RX_PASSWORD_32 SubFolder/app_pass.txt
```

- Up folder

```
RX_PASSWORD_32 ../app_pass.txt
```

- Absolute folder

```
RX_PASSWORD_32 /home/Data/ScriptFolder/app_pass.txt
```

2.4.19 TOGGLE_INFO**Command**

TOGGLE_INFO

Description

Toggles the INFO A lock to either protect or unlock the INFO A segment on MSP430F5xx and MSP430F6xx MCUs. For details on this lock, see the [MSP430x5xx and MSP430x6xx Family User's Guide](#).

Parameters

None

Examples

TOGGLE_INFO

2.4.20 TX_DATA_BLOCK

Command

```
TX_DATA_BLOCK {ADDRESS} {LENGTH} {FILENAME}
```

Description

Reads a block of memory from the device and writes the data to the specified file in TI TXT or Intel Hex format.

Parameters

ADDRESS [*mandatory*]

The address at which the read should begin (hex format).

LENGTH [*mandatory*]

The number of bytes to read (hex format).

FILENAME [*mandatory*]

The name of the TI TXT or Intel Hex file to write. The path of the file is relative to the location of the script file:

- Same folder: The file to write is in the same folder as the main script file.
- Subfolder: The file to write is in a folder one or more levels down relative to the folder that contains the script file.
- Up folder: The file to write is in a folder one or more levels up relative to the folder that contains the script file.
- Absolute folder: The file to write is in a folder that is specified by an absolute path.

Example (Windows)

- Same folder

```
TX_DATA_BLOCK 0x8000 0x1000 Data_Read.txt  
TX_DATA_BLOCK 0x8000 0x1000 .\Data_Read.txt
```

- Subfolder

```
TX_DATA_BLOCK 0x8000 0x1000 SubFolder\Data_Read.txt
```

- Up folder

```
TX_DATA_BLOCK 0x8000 0x1000 ..\Data_Read.txt
```

- Absolute folder

```
TX_DATA_BLOCK 0x8000 0x1000 C:\Data\ScriptFolder\Data_Read.txt
```

Example (Linux or OS X)

- Same folder

```
TX_DATA_BLOCK 0x8000 0x1000 Data_Read.txt
```

- Subfolder

```
TX_DATA_BLOCK 0x8000 0x1000 SubFolder/Data_Read.txt
```

- Up folder

```
TX_DATA_BLOCK 0x8000 0x1000 ../Data_Read.txt
```

- Absolute folder

```
TX_DATA_BLOCK 0x8000 0x1000 /home/Data/ScriptFolder/Data_Read.txt
```

2.4.21 TX_DATA_BLOCK_32

Command

```
TX_DATA_BLOCK_32 {ADDRESS} {LENGTH} {FILENAME}
```

Description

Identical to TX_DATA_BLOCK except that the memory is accessed by a 32-byte address. This command supports only MSP432P4 and MSP432E4 with USB DFU protocol.

NOTE: When the SPI communication is set to the highest speed of 1 MHz, there is a known issue due to a speed limitation in the serial library. To verify the memory, the application can use CRC_CHECK_32, or the application can use a lower speed to read the memory.

Parameters

ADDRESS [mandatory]

The address to start reading (hex format).

LENGTH [mandatory]

The number of bytes to read (hex format).

FILENAME [mandatory]

The name of the file to write. The path of the file is relative to the location of the script file:

- Same folder: The file to write is in the same folder as the main script file.
- Subfolder: The file to write is in a folder one or more levels down relative to the folder that contains the script file.
- Up folder: The file to write is in a folder one or more levels up relative to the folder that contains the script file.
- Absolute folder: The file to write is in a folder that is specified by an absolute path.

Example (Windows)

- Same folder

```
TX_DATA_BLOCK_32 0x8000 0x1000 Data_Read.txt
TX_DATA_BLOCK_32 0x8000 0x1000 .\Data_Read.txt
```

- Subfolder

```
TX_DATA_BLOCK_32 0x8000 0x1000 SubFolder\Data_Read.txt
```

- Up folder

```
TX_DATA_BLOCK_32 0x8000 0x1000 ..\Data_Read.txt
```

- Absolute folder

```
TX_DATA_BLOCK_32 0x8000 0x1000 C:\Data\ScriptFolder\Data_Read.txt
```

Example (Linux or OS X)

- Same folder

```
TX_DATA_BLOCK_32 0x8000 0x1000 Data_Read.txt
```

- Subfolder

```
TX_DATA_BLOCK_32 0x8000 0x1000 SubFolder/Data_Read.txt
```

- Up folder

```
TX_DATA_BLOCK_32 0x8000 0x1000 ../Data_Read.txt
```

- Absolute folder

```
TX_DATA_BLOCK_32 0x8000 0x1000 /home/Data/ScriptFolder/Data_Read.txt
```

2.4.22 TX_BSL_VERSION

Command

TX_BSL_VERSION

Description

Returns the BSL version of the MSP430 device.

Vendor : [xx], CI : [xx], API : [xx], PI : [xx]

Parameters

None

Examples

TX_BSL_VERSION

2.4.23 TX_BSL_VERSION_32

Command

TX_BSL_VERSION_32

Description

Returns the BSL version of the MSP432 device.

Vendor : [xxxx], CI : [xxxx], API : [xxxx], PI : [xxxx], BuildID : [xxxx]

Parameters

None

Examples

TX_BSL_VERSION_32

2.4.24 TX_BUFFER_SIZE

Command

TX_BUFFER_SIZE

Description

Tells the BSL to transmit the number of bytes available in the BSL data buffer for sending or receiving BSL core data packets.

NOTE: This command is supported for only some devices in the flash family.

Parameters

None

Examples

TX_BUFFER_SIZE

2.4.25 LOG

Command

LOG

Description

In the beginning of the script, LOG can be called to generate a log file of the script execution. The LOG command must be used before the MODE command.

The format name of generated log file is: <script_name>_<data>_<time>.log. The log is generated in the same directory as the script file.

2.4.26 VERBOSE

Command

VERBOSE

Description

Causes the PC application to toggle output for all transmitted and received bytes on or off.

Parameters

None

Examples

VERBOSE

2.4.27 FACTORY_RESET

Command

FACTORY_RESET {PASSWORD_0} {PASSWORD_1} {PASSWORD_2} {PASSWORD_3}

Description

This command is applicable only for the MSP432 MCUs. It writes the flash-boot override mailbox to perform the factory reset with password. If no password was configured before, this command sets the password to the default value 0xFFFFFFFF. See the *Device Security* chapter in the [MSP432P4xx SimpleLink™ Microcontrollers Technical Reference Manual](#). After the write to the flash-boot override mailbox, the BSL on the MSP432 MCU triggers the reboot reset.

Parameters

PASSWORD_0, PASSWORD_1, PASSWORD_2, PASSWORD_3 [mandatory]

 Password to perform the factory reset (hex format)

Examples

FACTORY_RESET 0x01234567 0x89ABCDEF 0x00112233 0x44556677

NOTE: The MSP432 MCUs have 32-bit memory architecture. The placing of the byte starts based on little endianness. Assume the first password is as given in the example. In the memory view, the bytes are placed as 0x67, 0x45, 0x23, 0x01, 0xEF, 0xCD, 0xAB, 0x89, 0x33, 0x22, 0x11, 0x00, 0x77, 0x66, 0x55, and 0x44, respectively. This information is delivered to avoid confusion when the verbose mode is executed for the FACTORY_RESET command.

2.5 Generate Firmware Image in TI TXT and Intel Hex Format

As mentioned in [Section 1](#), the BSL Scripter receive a firmware image to be transformed and then deployed to program the target device memory. BSL Scripter supports the TI TXT and Intel Hex formats. The following sections describe the generation of the firmware image using Code Composer Studio™ IDE (CCS) and IAR Embedded Workbench® for MSP430 IDE (IAR).

2.5.1 Code Composer Studio™ IDE (CCS)

1. In the CCS project ([Figure 3](#) shows the Blink LED project example), right click the Project and select **Properties**.

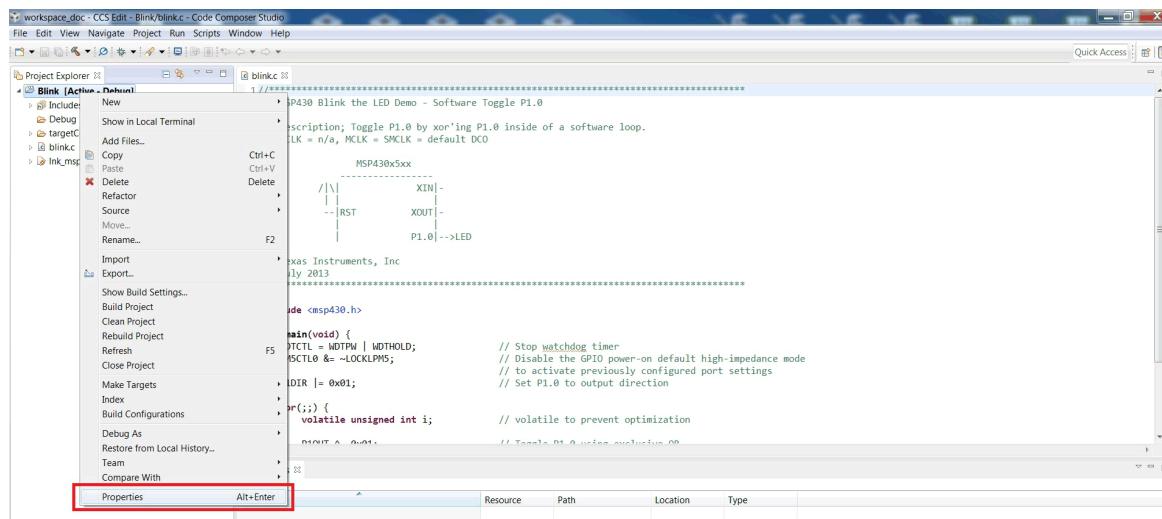


Figure 3. CCS – Project Properties

2. Click **Build → MSP430 Hex Utility**, and then select **Enable MSP430 Hex Utility** (see [Figure 4](#)).

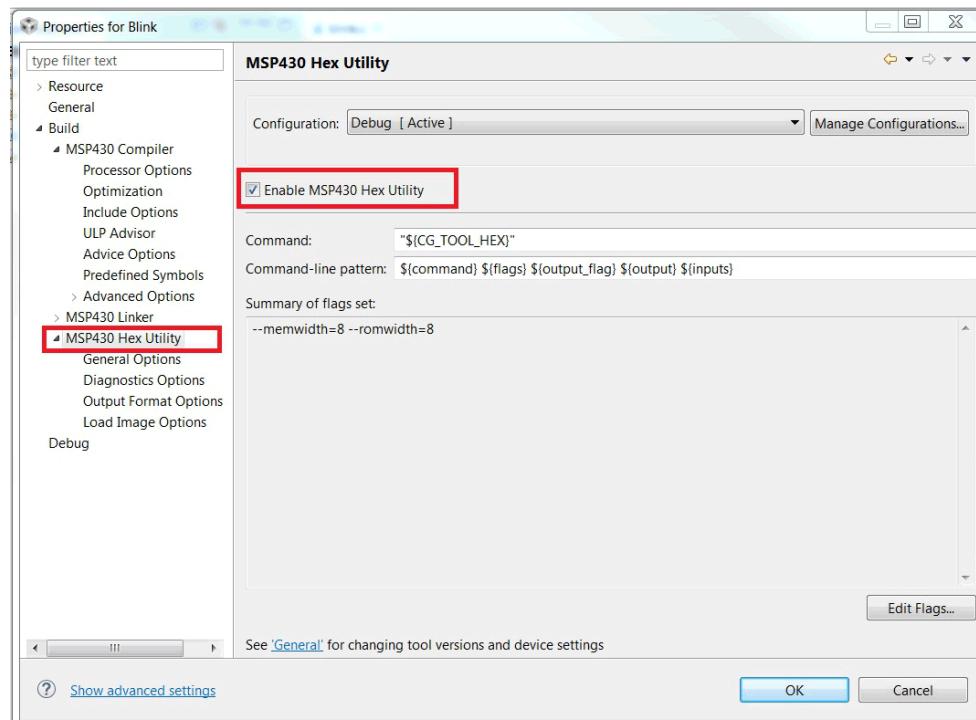


Figure 4. CCS – Enable MSP430 Hex Utility

3. Under **MSP430 Hex Utility**, select **Output Format Options**. The available output formats are listed in the window to the right. Select either **Output TI-TXT hex format (--ti_txt)** or **Output Intel hex format (--intel, -i)**.
4. Click OK.

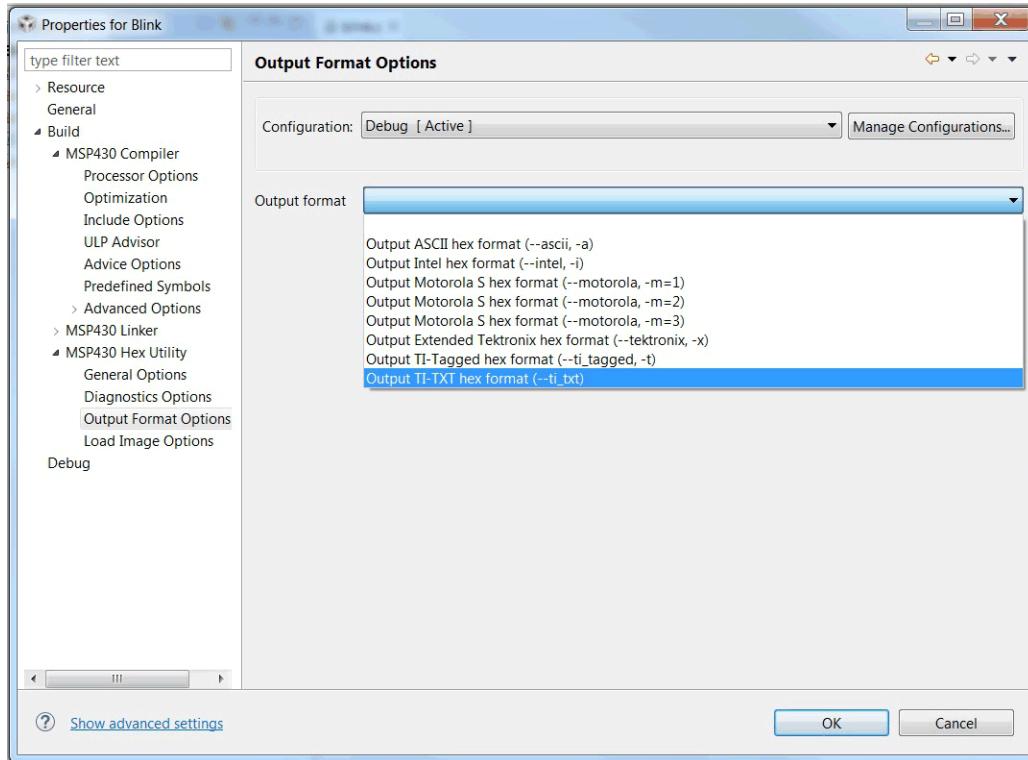


Figure 5. CCS – Select Output Format

5. Build the project. The file **Blink.txt** is generated in the **Debug** or **Release** folder, depending on the build option that been selected (see [Figure 6](#)).

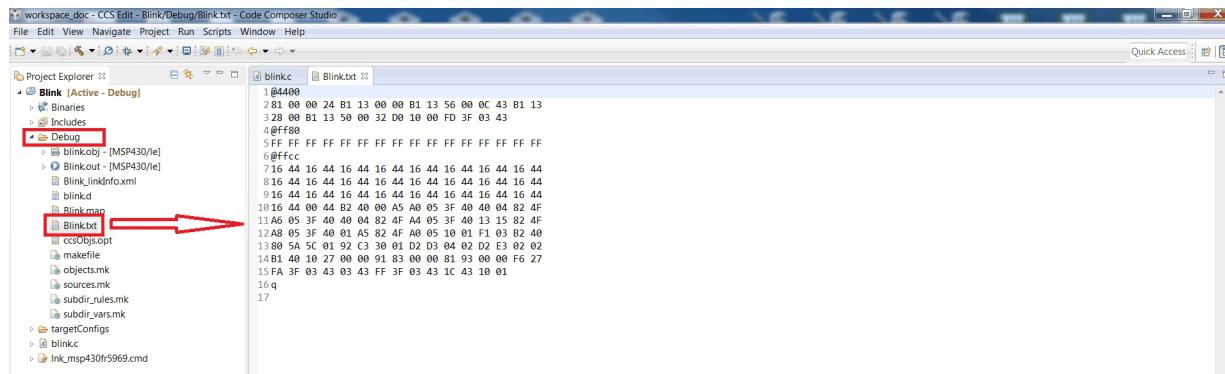


Figure 6. CCS – Output File

2.5.2 IAR Embedded Workbench® IDE (IAR)

1. Right click on the Project (the Blink example code is used in [Figure 7](#)), and select Options.

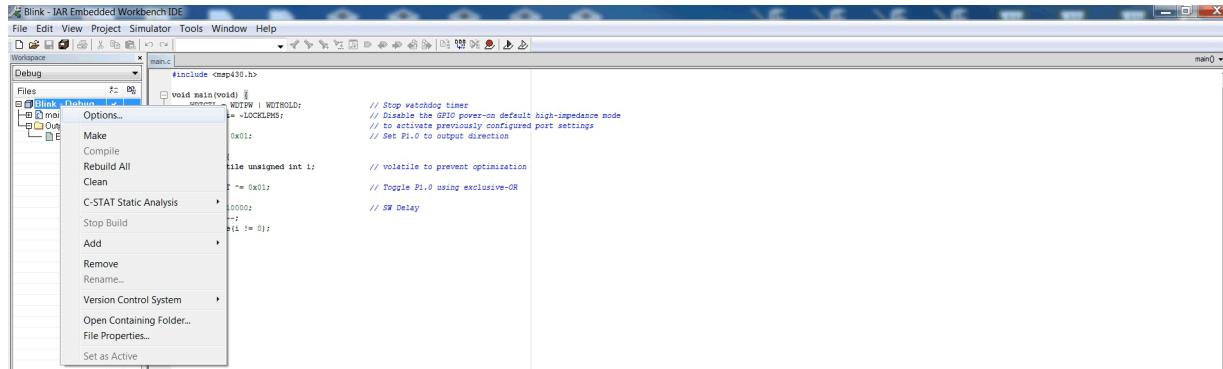


Figure 7. IAR – Project Options

2. Under **Linker Category**, select the **Output** tab. In the **Format** section, select **Other**, then select the **msp430-txt** or **intel-standard** output format (see [Figure 8](#)).

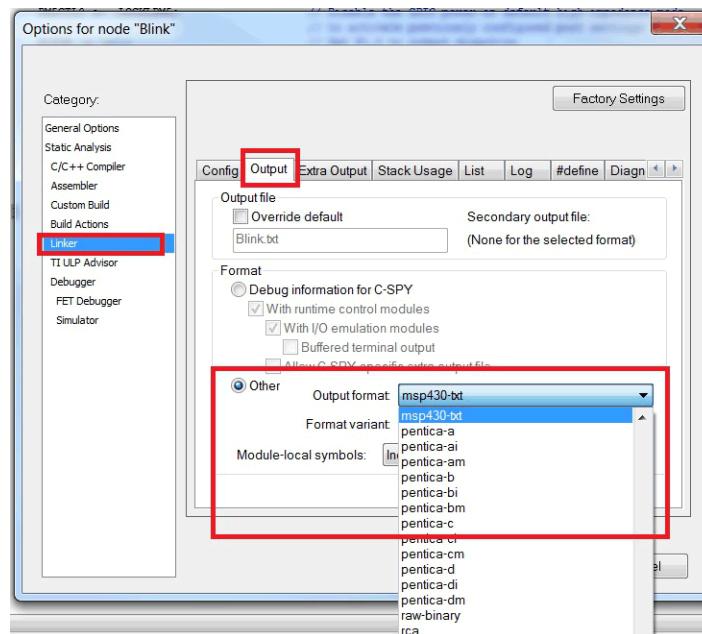


Figure 8. IAR – Select Output Format

3. Build or make the project. The **Blink.txt** or **Blink.hex** file (see [Figure 9](#)) is generated in the **Output** folder.

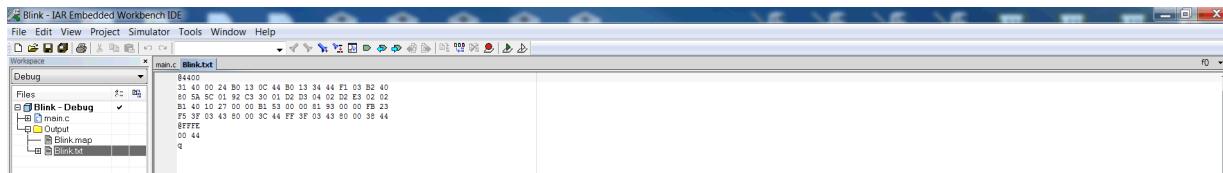


Figure 9. IAR – Output File

3 Command Line Interface Mode

Starting with version 3.2.0, the BSL Scripter supports a command line interface. From a software perspective, the command line interface uses the script mode command described in [Section 2](#). This section describes the commands and parameters available in the command line mode.

3.1 Comparison Between Script Mode and Command Line Interface Mode

The command line interface in the BSL Scripter reuses the script commands. In addition, some new commands have been added in the command line mode. [Table 2](#) compares the commands of the script mode and command line mode.

Table 2. Comparison of Script Mode and Command Line Mode Commands

No.	Commands in Script Mode	Commands in Command Line Mode	Functionality of the Commands in Command Line Mode	Section
1	N/A	--help -h --?	Display usage information for command line interface mode	Section 3.1.1
2	N/A	--exitInfo -e	Display exit specification	Section 3.1.2
3	LOG	--log -g	Set the logging mode	Section 3.1.3
4	N/A	--quiet -q	Set the quiet mode	Section 3.1.4
5	VERBOSE	--debug -d	Set the debug (also called verbose) mode	Section 3.1.5
6	MODE	--initComm -i	Initialize the communication parameters	Section 3.1.6
7	MODE	--device -n	Initialize the family parameters	Section 3.1.7
8	RX_PASSWORD RX_PASSWORD_32	--bslPwd -b	Send the password to unlock BSL	Section 3.1.8
9	CHANGE_BAUD_RATE	--speed -j	Change communication speed setting	Section 3.1.9
10	TOGGLE_INFO	--unlockInfo -u	Unlock information memory	Section 3.1.10
11	MASS_ERASE ERASE_SEGMENT	--erase -e	Erase memory	Section 3.1.11
12	RX_DATA_BLOCK RX_DATA_BLOCK_32 RX_SECURE_DATA_BLOCK RX_DATA_BLOCK_FAST	--write -w	Write memory	Section 3.1.12
13	TX_DATA_BLOCK RX_DATA_BLOCK_32	--read -r	Read memory	Section 3.1.13
14	CRC_CHECK CRC_CHECK_32	--verify -v	Verify memory	Section 3.1.14
15	N/A	--lockJtag -l	Lock the JTAG by writing the JTAG signature	Section 3.1.15
16	N/A	--pwdLockJtag -p	Set the JTAG lock password	Section 3.1.16
17	SET_PC SET_PC_32 REBOOT_RESET	--exit -z	Exit the BSL by setting PC register or applying a reboot reset	Section 3.1.17

3.1.1 Help Command

Syntax

```
--help  
-h  
--?
```

Description

Display usage information on the console.

3.1.2 Exit Info Command

Syntax

```
--exitInfo  
-e
```

Description

Display exit specification on the console.

3.1.3 Log Command

Syntax

```
--log  
-g
```

Description

Disables the logging mode, which is enabled by default. Without giving the log command, the log file is automatically generated. With giving the log command, the log file is not generated.

3.1.4 Quiet Command

Syntax

```
--quiet  
-q
```

Description

Enables the quiet mode, which is the default. In quiet mode, the execution status is not shown on the console, and only the failed status is shown.

3.1.5 Debug Command

Syntax

```
--debug  
-d
```

Description

Enables the debug (or verbose) mode, so that all bytes on the transmission line are shown on the console.

3.1.6 Initialize Communication Command

Syntax

```
--initComm [ComPort,Protocol,Speed]  
-i [ComPort,Protocol,Speed]
```

Description

This command has part of the function of the [MODE](#) command to set up the communication.

When Protocol and Speed are not given, the default is assigned, using UART protocol with baud rate of 9600.

Parameters

ComPort [*mandatory*]

It is mandatory for the serial communication protocol (UART, I²C, and SPI), to recognize on which comm port the BSL is connected with the PC. The USB protocol does not require this parameter.

Protocol [*optional*]

USB protocol must be declared if chosen. Other available protocols are:

- UART
- SPI
- I²C

When the Protocol parameter is not given, the default UART protocol is used for the initialization.

Speed [*optional*]

USB protocol ignores the speed parameter.

UART protocol speed is adjustable only for the MSP432 MCU. For MSP430, the baud rate must be 9600.

I²C and SPI protocol speed is adjustable (see [Section 2.4.1](#) for details).

Example

```
--initComm [COM45,UART,9600]  
--initComm [/dev/ttyACM0,I2C,100000]  
-i [USB]
```

3.1.7 Initialize Device and Family Command

Syntax

```
--device family  
-n family
```

Description

This command has part of the function of the [MODE](#) command to set up the family of the device.

Parameters

See [Section 2.4.1](#) for the available family definitions.

Example

```
--device 5xx  
-n MSP432P4xx
```

3.1.8 BSL Password Command

Syntax

```
--bslPwd filename  
-b filename
```

Description

This command executes:

- RX_PASSWORD for all families other than the P4xx family
- RX_PASSWORD_32 for the P4xx family

Parameters

filename [optional]

Filename is the file that contains the BSL password. When the filename parameter is not given, the default password is sent (for details, see [Section 2.4.17](#) and [Section 2.4.18](#)).

Example

```
--bslPwd  
-b pass.txt
```

3.1.9 Speed Command

Syntax

```
--speed FAST/MEDIUM/SLOW  
-j FAST/MEDIUM/SLOW
```

Description

This command executes the [CHANGE_BAUD_RATE](#) command, which is applicable only for UART communication.

Parameters

The argument is mandatory. The options are:

- FAST: Set baud rate to 115200 baud
- MEDIUM: Set baud rate to 57600 baud
- SLOW: Set baud rate to 9600 baud

3.1.10 Unlock Information Memory Command

Syntax

```
--unlockInfo  
-u
```

Description

This command executes the [TOGGLE_INFO](#) command to unlock the information memory on the device (see [Section 2.4.19](#) for details).

3.1.11 Erase Command

Syntax

```
--erase ERASE_ALL/ERASE_MAIN/ERASE_SEGMENT  
-e ERASE_ALL/ERASE_MAIN/ERASE_SEGMENT
```

Description

This command performs the memory erase. See the parameters for details of the difference between the parameter options.

Parameters

The argument is optional. When no argument is given, the ERASE_SEGMENT parameter is the default. The argument options are:

- ERASE_ALL: Erases all main memory.
- ERASE_MAIN: Erases all main memory.
- ERASE_SEGMENT: Must be used when the [Write](#) command is also called. Before the write is executed, the BSL Scripter erases the segment that will be programmed based on the firmware image given in the Write command.

The erase segment for MSP432 family executes the [ERASE_SEGMENT_32](#) command and assumes that the segment size is 4KB. For the other families, the erase segment executes the [ERASE_SEGMENT](#) command and assumes that the segment size is 512 bytes.

3.1.12 Write Command

Syntax

```
--write filename  
-w filename
```

Description

This command performs the programming to the memory based on the firmware image given as the parameters. The execution depends on the initialization family and communication:

- USB protocol: RX_DATA_BLOCK_FAST
- Crypto family : RX_SECURE_DATA_BLOCK
- P4xx family: RX_DATA_BLOCK_32
- Others: RX_DATA_BLOCK

The Write command is also related with the [Erase](#) command. When no Erase command is given, an ERASE_ALL is automatically performed.

Parameters

filename [*mandatory*]

For a description of the relative path syntax, see [Section 2.4.13](#) to [Section 2.4.16](#).

3.1.13 Read Command

Syntax

```
--read [filename, start_address-end_address]  
-r [filename, start_address-end_address]
```

Description

This command performs the reading of the specific address of the memory. The execution depends on the initialization of the family:

- P4xx family: TX_DATA_BLOCK_32
- Others: TX_DATA_BLOCK

Parameters

filename [*mandatory*]

For a description of the relative path syntax, see [Section 2.4.20](#) to [Section 2.4.21](#).

start_address, end_address *mandatory*

The start and end address parameters must be given in hexadecimal format. The start address is the location of the memory where the read should start, and the read continues to the end address.

Example

```
--read [readBlinkLED.txt, 0x8000-0xFFFF]  
-r [readBlinkLED.hex, 0x8000-0xFFFF]
```

3.1.14 Verify Command

Syntax

```
--verify filename  
--v filename
```

Description

This command performs the verification of the memory based on the firmware image which is given. It reuses the CRC Check command that the BSL protocol already has. The execution depends on the initialization of the family:

- P4xx family: CRC_CHECK_32
- Others: CRC_CHECK

Parameters

filename [*optional*]

When the verify command is called along with the write command, the filename is not necessary to be given. The verify command verifies the memory based on what the firmware image in the write command has. When the command is called and the write command is not called, the filename is a mandatory parameter.

Example

```
--write blinkLED.txt --verify  
-v blinkLED.txt
```

3.1.15 Lock JTAG Command

Syntax

```
--lockJtag  
-l
```

Description

This command locks JTAG by writing the JTAG signature for flash and FRAM families. For the flash family, the JTAG lock signature is located at 0xFF80, and for FRAM family the JTAG lock signature is located at 0x17FC. This location is written with 0x55, 0x55, 0x55, and 0x55 respectively. The command reuses the simple RX_DATA_BLOCK with the address and signature value defined internally in the BSL Scripter, therefore no additional file is required.

NOTE: For more information about code protection in MSP430 families, see [MSP Code Protection Features](#).

Parameters

None

3.1.16 JTAG Password Command

Syntax

```
--pwdLockJtag [numPwd,Pwd(0),Pwd(1),...,Pwd(numPwd-1)]  
-p [numPwd,Pwd(0),Pwd(1),...,Pwd(numPwd-1)]
```

Description

This command must be used with the lock JTAG command and is applicable for only the FRAM family. FRAM family security provides a password mechanism to lock and unlock the JTAG. The password is located starting on 0xFF88, and user can customize the length of the password.

Parameters

numOfPwd *[mandatory]*

Number bytes in the password that is supplied. This value must in hexadecimal format.

Pwd(0),Pwd(1),...,Pwd(numOfPwd-1) *[mandatory]*

Password in bytes.

Example

```
--pwdLockJtag [0x04,0x1A,0x2B,0x3C,0x4D]  
-p [0x04,0x1A,0x2B,0x3C,0x4D]
```

3.1.17 Exit Command

Syntax

```
--exit [RESET]/[SET_PC,address]/[SET_PC_32,address]  
-z
```

Description

This command exits the BSL Scripter and specifies the operation for the MCU to perform next:

- Reset: Reuses the REBOOT_RESET command as described in [Section 2.4.12](#).
- Set PC: Reuses the SET_PC or SET_PC_32 command as described in [Section 2.4.9](#) or [Section 2.4.10](#), respectively.
- Normal exit: Closes the communication port and exits the BSL Scripter application.

Parameters

[RESET]

Implements the REBOOT_RESET command.

[SET_PC,address]

Implements the SET_PC command and jumps to the specified address. Address must be in hexadecimal format.

[SET_PC_32,address]

Implements the SET_PC_32 command and jumps to the specified address. Address must be in hexadecimal format.

Example

```
--exit [RESET]  
-z [SET_PC,0x8000]  
-z [SET_PC_32,0x1000]  
-z
```

4 Using BSL Scripter With Communication Bridge Tools

The BSL Scripter version 3 and higher is tested with the following communication bridge tools:

- MSP-BSL Rocket
- MSP-FET
- XDS110 backchannel UART on MSP432 LaunchPad development kit

This section describes the invoke sequence and how to use these tools.

4.1 Invoke Sequence

The BSL Scripter invokes the BSL application that resides on the target device when the invoke sequence is applied. Other than the P4xx family, the invoke sequence is performed by giving the special sequence on the Reset and Test pins (see detailed information in the *Standard Reset and BSL Entry Sequence* section of [MSP430 Programming With the Bootloader \(BSL\)](#)). For the P4xx and E4xx families, the invocation does not use the Reset and Test pins (see detailed information in the *BSL Invocation* section of the [MSP432™ SimpleLink™ Microcontrollers Bootloader \(BSL\) User's Guide](#)).

4.2 Example

This example demonstrates how to connect the MSP-FET with MSP430 target devices. Connection to the following pins is required for the UART BSL in the MSP430 target device:

- TEST
- RESET
- BSL-RX (communication direction: data from PC to the target device)
- BSL-TX (communication direction: data from target device to the PC)
- V_{CC}
- Ground

[Table 3](#) and [Figure 10](#) show the connection of the MSP-FET and the MSP430FR5739 LaunchPad development kit.

Table 3. Connection of MSP-FET and MSP430FR5739 LaunchPad™ Development Kit

Pin	MSP-FET	MSP430FR5739 LaunchPad™ Development Kit	Jumper Wire Color
TEST	Pin 8	TEST	White
RESET	Pin 11	RST	Brown
BSL-RX	Pin 14	P2.1/RXD	Orange
BSL-TX	Pin 12	P2.0/TXD	Yellow
V_{CC}	Pin 2	VCC	Red
GND	Pin 9	GND	Black

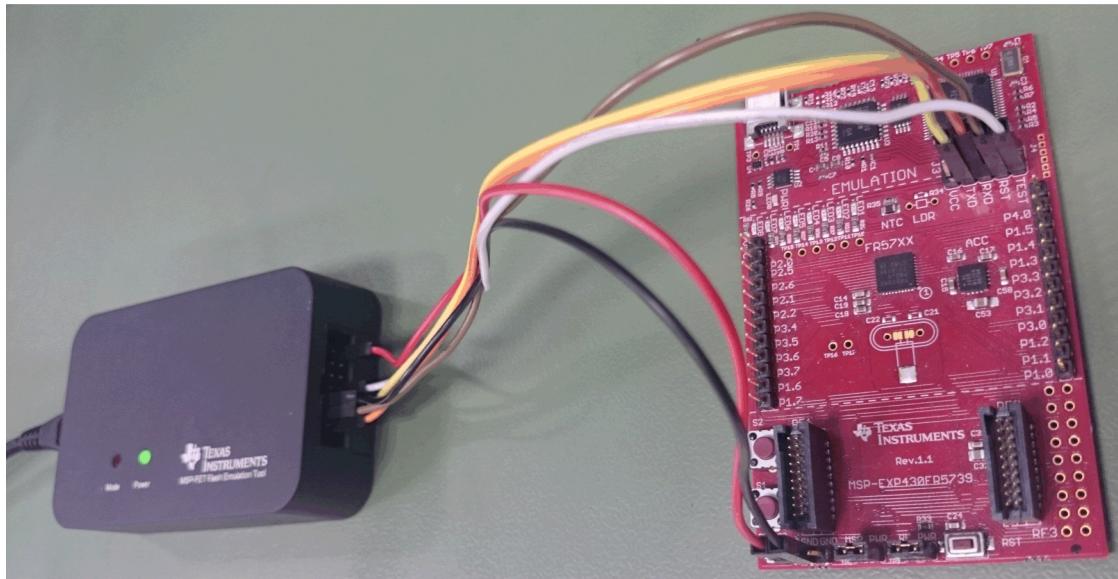


Figure 10. Connection of MSP-FET and MSP430FR5739 LaunchPad™ Development Kit Using Jumper Wires

Select the correct COM port for the MSP-FET.

1. Open the Device Manager in Windows (see [Figure 11](#)).

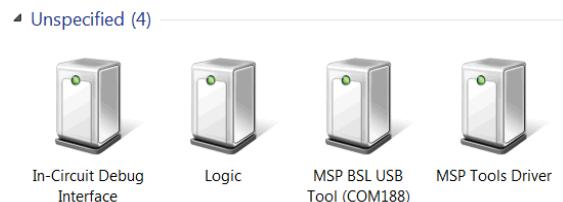


Figure 11. Device Manager

2. Right click on the *MSP Tools Driver* that belongs to the MSP-FET, and click Properties (see Figure 12).

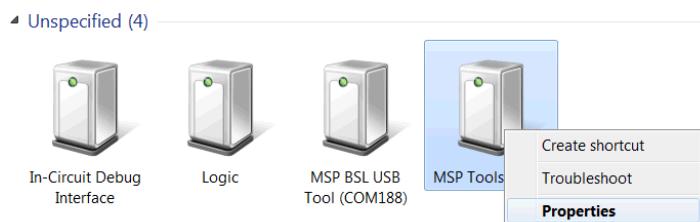


Figure 12. Open MSP Tools Driver Properties

- Two COM ports are available in the Properties windows. BSL communication using MSP-FET is supported by MSP Application UART1. In the example in [Figure 13](#), COM54 should be used to communicate with the BSL.

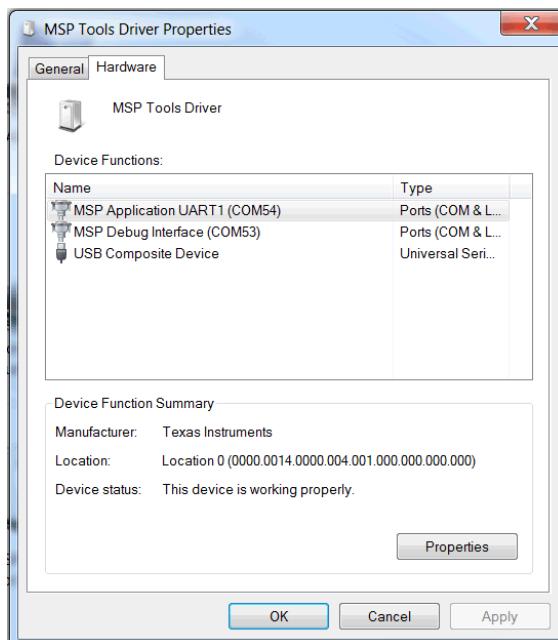


Figure 13. MSP Tools Driver Properties

Table 4 and [Figure 14](#) show the connection of the MSP-BSL Rocket and the MSP430FR5739 LaunchPad development kit.

Table 4. Connection of MSP-BSL and MSP430FR5739 LaunchPad™ Development Kit

Pin	MSP-BSL Rocket	MSP430FR5739 LaunchPad™ Development Kit	Jumper Wire Color
TEST	Pin 7	TEST	White
RESET	Pin 4	RST	Brown
BSL-RX	Pin 3	P2.1/RXD	Orange
BSL-TX	Pin 1	P2.0/TXD	Yellow
V _{cc}	Pin 6	VCC	Red
GND	Pin 5	GND	Black

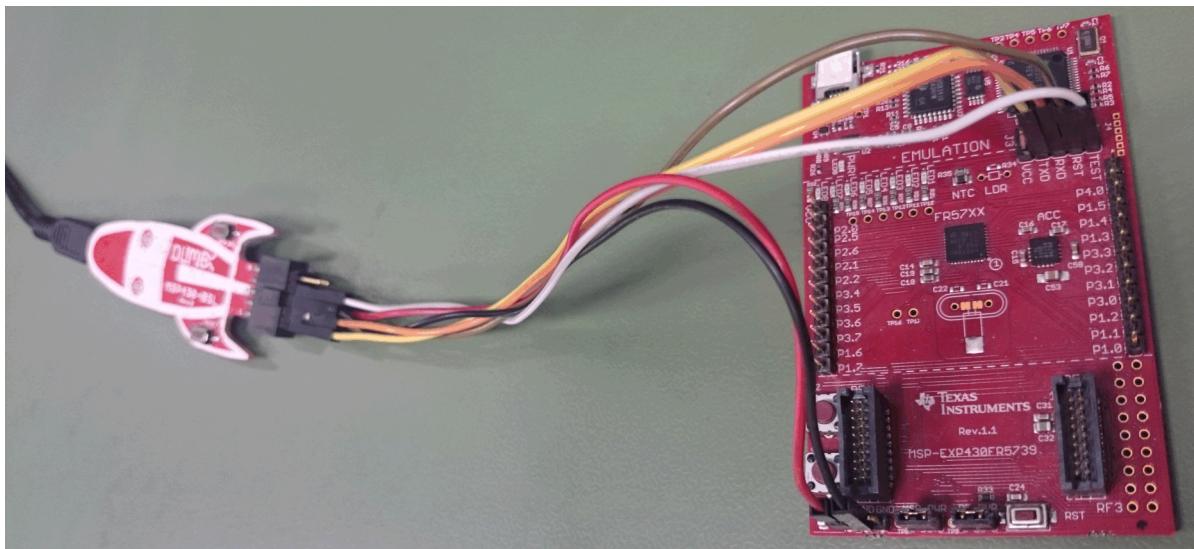


Figure 14. Connection of MSP-BSL and MSP430FR5739 LaunchPad™ Development Kit Using Jumper Wires

5 Frequently Asked Questions (FAQ)

1. How is the BSL invocation generated by BSL Scripter?

The MSP430 and MSP432 MCUs have different methods of BSL invocation.

- **MSP430 MCUs**

This explanation applies to only the MSP430F5xx, MSP430F6xx, and MSP430FRxx MCUs. In addition, BSL Scripter is executed using MSP-FET or MSP-BSL Rocket. The BSL in these MSP430 devices is invoked through the RST and TST pins. Figure 2 of [MSP430 Programming With the Bootloader \(BSL\)](#) shows how the RST and TST signals must be configured to enter the BSL mode. The MSP-BSL Rocket and MSP-FET generate these signals.

The BSL Scripter sets the device initialization in a script (MODE command) or on the command line from the user (--device/-n command). When the family is selected as Flash or FRAM, the BSL Scripter generates a special baud rate in the beginning, 9601. The MSP-BSL Rocket or MSP-FET receives this special baud rate and generates the invoke sequence in the RST and TST pins.

- **MSP432 MCUs**

For the MSP432 BSL UART, BSL invocation on a blank device does not use the RST and TST pins. Instead, the BSL Scripter starts with the selected baud rate (for example, 9600 or 19200) and sends "0xFF" to the BSL. For these devices, the invoke byte (instead of the invoke sequence) invokes the BSL. The MSP-BSL Rocket and MSP-FET do not generate the invoke sequence as in the MSP430. If the BSL in the MSP432 device is invoked, then the device returns "0x00" as succeed status. If "0x00" is not returned, BSL Scripter shows "Initialization failed" on the screen.

The invocation with "0xFF" cannot succeed when the device is already programmed (@0x0 to @0x100 already filled). In this case, the BSL must be invoked using hardware invocation. For details about configuring the BSL by hardware invocation, see the *Bootloader (BSL) Configuration* section of the [SimpleLink™ MSP432™ Security and Update Tool User's Guide](#).

2. Why do I see different behavior when running an application by SET_PC compared to a device reset?

In some cases, calling the downloaded or user application by SET_PC instead of with a device reset causes unexpected behavior.

As an example, consider a blink application that should run with clock of 1 MHz and blink the LED with a period of 1 second. First, the BSL is invoked on the empty device, and then the blink application is downloaded. Next, the SET_PC command is called to run the blink application. The LED blinks but with a faster frequency than expected based on the setup in the blink application itself. When the device is reset, the blink application now runs as expected.

The BSL on the device is an application (see [Figure 15](#)). This application is programmed from the factory, and the user can call it by using a specific invoke sequence as described in the previous question.

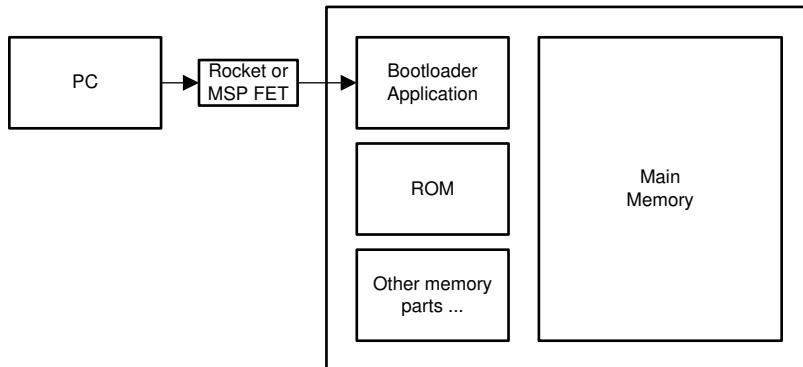


Figure 15. High-Level View of the BSL System

The BSL application itself contains certain functions. For example, it configures the clock, peripherals, and memory access. The BSL application then enters a while loop to receive commands from BSL scripter through MSP-BSL Rocket or MSP-FET so that it can answer the command (see [Figure 16](#)).

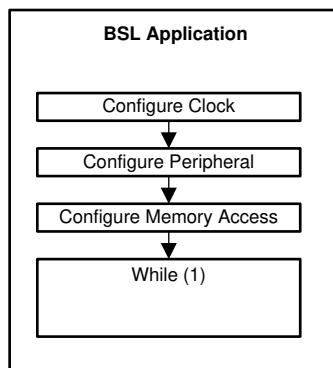


Figure 16. BSL Application

Consider two different blink applications. The first is called BlinkLED_F6459 and the second is taken from the example in CCS, ucs_ex1_DCO12MHz. Looking at the flow of both applications in [Figure 17](#), the differences can be seen. The blinking mechanism is the same for both, but the BlinkLED_F6459 has no configuration for clock, while ucs_ex1_DCO12MHz does configure the clock.

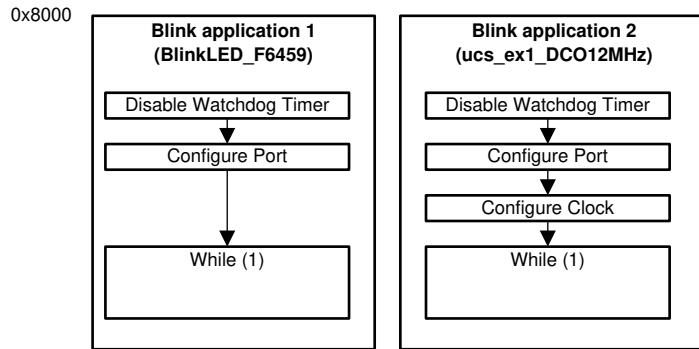


Figure 17. Two Different Blink Applications

When the BSL application jumps to the blink application using the SET_PC command, the blink application is executed. There is no other initialization done, so what was configured in the BSL application stays. Therefore, BlinkLED_F6459 continues to use the clock configuration from the BSL, but ucs_ex1_DCO12MHz configures the clock for its intended blink frequency. This explains why the first blink LED runs faster than expected, because BSL application runs with a higher frequency than intended for BlinkLED_F6459.

3. How do I enable or disable the JTAG/SBW lock using the BSL?

In MSP430 devices, JTAG/SBW is locked by changing the JTAG/SBW signature that resides in memory. When the JTAG/SBW is locked on a device, the bootloader can be executed to rewrite the JTAG/SBW signature and unlock the device. The location of the JTAG/SBW signature differs between flash and FRAM devices.

MSP430 Flash Devices

The JTAG/SBW signature resides in the BSL flash area (see [Table 5](#)). Even though BSL programming can rewrite this area, when the JTAG signature is programmed with 0000_0000h, it cannot be reprogrammed. To change the value back to FFFF_FFFFh, the segment must be erased, which erases all of the BSL area.

A script example is provided in the following BSL Scripter programmer package:
MSPBSL_Scripter_{OS}\Example\Flash_OpenJtag

Table 5. JTAG/SBW Lock on F5xx and F6xx Devices

Name	Address	Value	Device Security
JTAG/SBW Signature	17FCh to 17FFh	FFFF_FFFFh	JTAG/SBW is unlocked.
		0000_0000h	
		Any other value	JTAG/SBW is locked.

MSP430 FRAM Devices

The FRAM can be written with any value without the need to erase the memory, which differs from the flash-based devices. If the location of the JTAG/SBW signature (see [Table 6](#)) is protected by the IPE module, this protection prevents writing by the BSL.

A script example is provided in the following BSL Scripter programmer package:
MSPBSL_Scripter_{OS}\Example\FRAM_OpenJtag

For more information regarding securing the JTAG interface in the MSP430 family, see [MSP Code Protection Features](#).

Table 6. JTAG/SBW Lock on FR5xx and FR6xx Devices

Name	Addresses	Value	Device Security
JTAG/SBW Signature	FF80h to FF83h	5555h_5555h	JTAG/SBW is locked without password.
		@FF80h = AAAAh @FF82h = password length in words	JTAG/SBW is locked with password.
		Any other value (default value: FFFF_FFFFh)	JTAG/SBW is unlocked.

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from May 8, 2018 to February 28, 2020

Page

- | | |
|--|---|
| • Added the earliest supported operating system versions to the paragraph that begins "The BSL Scripter package provides..." in Section 1 Introduction | 3 |
|--|---|

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2020, Texas Instruments Incorporated