SEGGER Flash Loader

Contents

SEGGER Flash Loader
- Minimum supported J-Link software version

Supported CPU architectures

Supported flash types

Entry functions overview
- SEGGER_FL_Prepare
- SEGGER_FL_Restore
- SEGGER_FL_Program
- SEGGER_FL_Erase
- SEGGER_FL_EraseChip
- SEGGER_FL_CheckBlank
- SEGGER_FL_Verify
- SEGGER_FL_Read
- SEGGER_FL_CalcCRC
- SEGGER_FL_Start
- SEGGER_FL_GetFlashInfo

SEGGER_FL_Flags
- SEGGER_FL_FLAG_SUPPORT_AUTO_ERASE
- SEGGER_FL_FLAG_GO_INT_EN

SEGGER_FL_MaxBlocksizeErase

FlashDevice struct

Integrate SFL in J-Link software

Stack usage

Section layout

Templates

Troubleshooting

SFL FAQ

SEGGER Flash Loader

The SEGGER Flash Loader (SFL) is part of the J-Link Device Support Kit (DSK) and allows users to add flash programming support for a new device on their own.

Minimum supported J-Link software version

If not stated otherwise for a specific function / feature / etc., the minimum supported J-Link software version is V7.80.

Supported CPU architectures

The SEGGER Flash Loader (SFL) is available for the following architectures:

- ARMv4/v5
  - ARM720T, ARM7TDMI, ARM7TDMI-S
  - ARM920T, ARM922T, ARM926EJ-S, ARM946E-S, ARM966E-S
- ARMv6-M
  - Cortex-M0, Cortex-M0+, Cortex-M1
- ARMv7-M
  - Cortex-M3, Cortex-M4, Cortex-M7
- ARMv8-M
  - Cortex-M23, Cortex-M33, Cortex-M55, Cortex-M85
- ARMv7-A
  - Cortex-A5, Cortex-A7, Cortex-A8, Cortex-A9, Cortex-A12, Cortex-A15, Cortex-A17
- ARMv7-R
  - Cortex-R4, Cortex-R5, Cortex-R7, Cortex-R8
- ARMv8-R
  - Cortex-R52
- RISC-V RV32

> **Note:**
>
> Architectures / cores not listed here must be assumed as being not supported.
> While they may work by chance, no support is provided for problems in regards to getting SEGGER Flash Loaders running on non-listed architectures.
>
> In case of doubt, please feel free to contact SEGGER.

Supported flash types

Because a SEGGER Flash Loader programs the flash via the MCU it runs on, basically any flash (or other non-volatile memories) can be supported.
This applies to flash that is memory mapped accessible by the MCU (internal flash, external QSPI NOR flash, ...) as well as to non-memory mapped flashes (external I2C EEPROM, external NAND flash, ...)

For non-memory-mapped flashes, the following additional entry functions must be implemented in an SFL:

- SEGGER_FL_CheckBlank()
- SEGGER_FL_Verify()
- SEGGER_FL_Read()

- SEGGER_FL_CalcCRC()

Entry functions overview

The following table gives an overview about the mandatory and optional entry functions, of a SEGGER Flash Loader:

Entry function	Type
SEGGER_FL_Prepare()	Mandatory
SEGGER_FL_Restore()	Mandatory
SEGGER_FL_Program()	Mandatory
SEGGER_FL_Erase()	Mandatory
SEGGER_FL_EraseChip()	Optional
SEGGER_FL_CheckBlank()	Optional
SEGGER_FL_Verify()	Optional
SEGGER_FL_Read()	Optional
SEGGER_FL_CalcCRC()	Optional
SEGGER_FL_Start()	Optional
SEGGER_FL_GetFlashInfo()	Optional

SEGGER_FL_Prepare

Prototype

```
int SEGGER_FL_Prepare(U32 PreparePara0, U32 PreparePara1, U32 PreparePara2);
```

Function description
Mandatory function. Must be present to make SFL detected as valid.
Prepares SFL for further usage. Very first function of SFL to be called.
Can rely on not being called multiple times in a row without a SEGGER_FL_Restore() in between.
This function may be used for example to initialize the PLL of the device before flash programming starts, to improve flash programming performance.
The following is a valid series of calls:

```
SEGGER_FL_Prepare()
SEGGER_FL_Erase()
SEGGER_FL_Program()
SEGGER_FL_Restore()
SEGGER_FL_Prepare()
SEGGER_FL_Erase()
SEGGER_FL_Restore()
```

Parameters

Parameter	Description
PreparePara0	Reserved for future use. Always == 0 for now

PreparePara1	Reserved for future use. Always == 0 for now
PreparePara2	Reserved for future use. Always == 0 for now

Return value

Value	Meaning
>= 0	OK
== 0	OK, Clock speed of core not changed
> 0	New clock speed in [Hz](!) of core
< 0	Error

SEGGER_FL_Restore

Prototype

```
int SEGGER_FL_Restore(U32 RestorePara0, U32 RestorePara1, U32 RestorePara2);
```

Function description
Mandatory function. Must be present to make SFL detected as valid.
Restores SFL after work is done. Very last function of SFL to be called.
This function may be used for example to restore the PLL settings of the device after flash programming is done.
This is for example needed when using the unlimited number of breakpoints in flash (https://www.segger.com/products/debug-probes/j-link/technology/flash-breakpoints/) feature of J-Link.

Parameters

Parameter	Description
RestorePara0	Reserved for future use. Always == 0 for now
RestorePara1	Reserved for future use. Always == 0 for now
RestorePara2	Reserved for future use. Always == 0 for now

Return value

Value	Meaning
>= 0	OK
< 0	Error

Notes

> **Note:**
>
> Can rely on not being called multiple times in a row without a SEGGER_FL_Prepare() in between.
> The following is a valid (but unlikely) series of calls:
>
> - SEGGER_FL_Prepare()

> - SEGGER_FL_Erase()
> - SEGGER_FL_Program()
> - SEGGER_FL_Restore()
> - SEGGER_FL_Prepare()
> - SEGGER_FL_Erase()
> - SEGGER_FL_Restore()

SEGGER_FL_Program

Prototype

```
int SEGGER_FL_Program(U32 DestAddr, U32 NumBytes, U8 *pSrcBuff);
```

Function description

Mandatory function. Must be present to make SFL detected as valid.
Programs flash. The block passed to this function is always a multiple of what is indicated as page size by FlashDevice.PageSize.
This function can rely on only being called with destination addresses and NumBytes that are aligned to FlashDevice.PageSize.

Parameters

Parameter	Description
DestAddr	Destination address. Always aligned to FlashDevice.PageSize
NumBytes	Number of bytes to be programmed (always a multiple of FlashDevice.PageSize)
pSrcBuff	Pointer to the data to be programmed

Return value

Value	Meaning
== 0	O.K.
== 1	Error
== 2	O.K., already performed implicit verify of block that was just programmed

SEGGER_FL_Erase

Prototype

```
int SEGGER_FL_Erase(U32 SectorAddr, U32 SectorIndex, U32 NumSectors);
```

Function description

Mandatory function. Must be present to make SFL detected as valid.
Erases flash. For one-time programmable memories (OTP), this function may check if the passed sector is still erased/not programmed and throw an error in case it already is programmed.

Parameters

Parameter	Description
SectorAddr	Address of the first sector to be erased.
SectorIndex	Index of the start sector to be erased. Counting starts at 0 (first sector reported by FlashDevice.SectorInfo[0])
NumSectors	Number of sectors to be erased. It is the responsibility of this function to track the sector size in case a device provides multiple sector sizes.

Return value

Value	Meaning
== 0	O.K.
== 1	Error

SEGGER_FL_EraseChip

Prototype

```
int SEGGER_FL_EraseChip(void);
```

Function description
Optional function. May be present to speed up erase procedure under special circumstances

Parameters

Parameter	Description
None	

Return value

Value	Meaning
== 0	O.K.
== 1	Error

SEGGER_FL_CheckBlank

Prototype

```
int SEGGER_FL_CheckBlank(U32 Addr, U32 NumBytes, U8 BlankValue);
```

Function description
Optional function. May not be present.
Checks if a memory region is blank / erased / unprogrammed.
Mainly used to speed up flash programming as it allows J-Link to skip calls to SEGGER_FL_Erase() before calling SEGGER_FL_Program(), in case a sector is already erased.

Parameters

Parameter	Description
Addr	Address where checking for blank / erased data should start
NumBytes	Number of bytes to be checked
BlankValue	Blank (erased) value of flash (Most flashes have 0xFF, some have 0x00, some do not have a defined erased value)

Return value

Value	Meaning
>= 0	OK
== 0	OK, complete region is blank
== 1	OK, not the complete region is blank
< 0	Error

SEGGER_FL_Verify

Prototype

```
U32 SEGGER_FL_Verify(U32 Addr, U32 NumBytes, U8* pData);
```

Function description

Optional function. May not be present. Not present for most memory-mapped accessible flash memories.
Verifies a given range by comparing the provided data versus the data in the flash.
Mainly used for non-memory mapped flashes in case flash is not simply read-accissble for the core at a certain address.

Parameters

Parameter	Description
Addr	Address where verify should start
NumBytes	Number of bytes to verify
pBuff	Pointer to data to compare flash contents to

Return value

Value	Meaning
xxx	End address of verification. ReturnVal == Addr+NumBytes indicates successful verification ReturnVal != Addr+NumBytes indicates an error and gives the address where verification failed

SEGGER_FL_Read

Prototype

```
int SEGGER_FL_Read(U32 Addr, U32 NumBytes, U8 *pDestBuff);
```

Function description

Optional function. May not be present.
Not needed for memory-mapped flashes that can be just read by the MCU core as RAM or other memories. Mainly used for non-memory mapped flashes where a virtual addres has been assigned to make J-Link flash download work.
If this function is present, J-Link will call it to read flash memory instead of reading it directly via the core.

Parameters

Parameter	Description
Addr	Address to start reading flash at
NumBytes	Number of bytes to read
pBuff	Pointer to buffer where to store read data to

Return value

Value	Meaning
>= 0	OK, number of bytes read (usually == NumBytes)
< 0	Error

SEGGER_FL_CalcCRC

Prototype

```
U32 SEGGER_FL_CalcCRC(U32 CRC, U32 Addr, U32 NumBytes, U32 Polynom);
```

Function description

Optional function, may not be present.
If present, J-Link may call this function to speed up the verification because not all data must be read + transferred back to the host but instead only the CRC (calculated by the SFL) is transferred to the host.

Parameters

Parameter	Description
CRC	Start value of CRC
Addr	Address where calculation of CRC starts
NumBytes	Number of bytes to calculate CRC on
Polynom	Polynom to use for CRC calculation

Return value

Value	Meaning

xxx	Calculated CRC

Notes

> **Note:**
>
> 1. If an SFL implements this function, it is recommended to simply call SEGGER_FL_Lib_CalcCRC() from the SEGGER_SFL_Lib.a library and pass its return value to the caller.

SEGGER_FL_Start

Prototype

```
void SEGGER_FL_Start(volatile struct SEGGER_FL_CMD_INFO* pInfo);
```

Function description

Optional function, may not be present.
If present and target architecture supports turbo mode, J-Link will use this entry function to start turbo mode which will lead to a big performance gain in flash programming.
If an SFL implements this function, it should simply call SEGGER_FL_Lib_StartTurbo() from the SEGGER_SFL_Lib.a library. The exact operation of SEGGER_FL_Lib_StartTurbo() is not disclosed.

Parameters

Parameter	Description
pInfo	Pointer to command info for SEGGER_FL_Lib_StartTurbo()

Return value

Value	Meaning
None	

SEGGER_FL_GetFlashInfo

Prototype

```
int SEGGER_FL_GetFlashInfo(SEGGER_FL_FLASH_INFO* pInfo, U32 InfoAreaSize);
```

Function description

Optional function. May not be present.
Only needed for flash loaders where the exact sectorization is determined at runtime.
This is for example the case for some loaders for external QSPI NOR flash, where different users may have different QSPI flashes connected to their hardware.
This allows to have only 1 loader to serve many different setups.

Parameters

Parameter	Description

pInfo	Points to info area which is used to store the flash info.
InfoAreaSize	Indicates how big the area pointed to by SEGGER_FL_FLASH_INFO really is. May be used to extend pInfo->aRangeInfo[] in case more elements are needed for the detected flash memory.

Return value

Value	Meaning
>= 0	OK
< 0	Error

Notes

Note:

1. If present, this function is called before any erase, program, verify operation but after Prepare().
2. If present, this function overrides the information provided by FlashDevice.TotalSize and FlashDevice.SectorInfo[]

SEGGER_FL_Flags

SEGGER_FL_Flags is an optional constant which can be specified in the flash algorithm. It can be used to influence the behavior of how the flash loader is called.

```
const U32 SEGGER_FL_Flags __attribute__((used)) = (SEGGER_FL_FLAG_XXX);
```

The following table gives an overview about the supported flags:

Name	Bit position
SEGGER_FL_FLAG_SUPPORT_AUTO_ERASE	[0:0]
SEGGER_FL_FLAG_GO_INT_EN	[1:1]
Reserved for future use	[31:2]

SEGGER_FL_FLAG_SUPPORT_AUTO_ERASE

Description
If set, the implicit erase before program will be skipped. This can be useful for memory types which can be programmed without being erased before.

Definition

```
#define SEGGER_FL_FLAG_SHIFT_SUPPORT_AUTO_ERASE     (0)
#define SEGGER_FL_FLAG_SUPPORT_AUTO_ERASE           (1 << SEGGER_FL_FLAG_SHIFT_SUPPORT_AUTO_ERASE)
```

Product	Supported
J-Link DLL	✓

J-Flash	✓
Flasher Stand-Alone mode	✓

SEGGER_FL_FLAG_GO_INT_EN

Description
If set, flash loader will be executed with interrupts enabled. This is required if the flash loader make use interrupts.

Definition

```
#define SEGGER_FL_FLAG_SHIFT_GO_INT_EN              (1)
#define SEGGER_FL_FLAG_GO_INT_EN                    (1 << SEGGER_FL_FLAG_SHIFT_GO_INT_EN)
```

Product	Supported
J-Link DLL	✓
J-Flash	✓
Flasher Stand-Alone mode	✓

SEGGER_FL_MaxBlocksizeErase

SEGGER_FL_MaxBlocksizeErase is an optional constant which can be specified in the flash algorithm. It can be used to influence the maximum erase block size with which the flash loader is called. E.g. SEGGER_FL_MaxBlocksizeErase = 0x10000 (64 KB) will limit the the block size to 64 KB. So e.g. with a flash with sector size 0x2000 (8 KB) the erase call will have 64 KB / 8 KB = 8 sectors as NumSectors parameter as a maximum. If SEGGER_FL_MaxBlocksizeErase is not specified the default value is 128 KB.

```
const U32 SEGGER_FL_MaxBlocksizeErase __attribute__((used)) = (0x00010000); // Overrides maximum block size to 64
KB
```

Product	Supported
J-Link DLL	✓
J-Flash	✓
Flasher Stand-Alone mode	✗

FlashDevice struct

The FlashDevice structure variable contains all the static information about the flash algorithm like sectorization of the flash, programming chunk size, ...
In the following, the structure elements are explained in detail to give a good idea about what needs to be filled in for a new flash loader.

```
struct FlashDevice  {
  U16 AlgoVer;
  U8  Name[128];
  U16 Type;
  U32 BaseAddr;
  U32 TotalSize;
  U32 PageSize;
  U32 Reserved;
  U8  ErasedVal;
  U32 TimeoutProg;
  U32 TimeoutErase;
  struct SECTOR_INFO SectorInfo[MAX_NUM_SECTORS];
};
```

Member	Explanation
AlgoVer	Set to 0x0101. Do not set to anything else!
Name	Name of the flash bank this flash loader handles. (E.g. "internal flash", "QSPI", ...). Must not exceed 127 characters (last character reserved for string termination). NEVER change the size of this array!
Type	Flash device type. Currently ignored. Set to 1 to get max. compatibility.
BaseAddr	Flash base address. It is recommended to always use the real address of the flash here, even if the flash is also available at other addresses (via an alias / remap), depending on the current settings of the device.
TotalSize	Total flash device size in bytes. This describes the total size of the flash that is achieved when summing up all sector info from <SectorInfo>. If the flash is 512 KB in size, <TotalSize> must be 0x80000 (524,288 => 512 KB)
PageSize	This field describes in what chunks J-Link feeds the flash loader. The SEGGER_FL_Program() function will be called with a data chunk of multiple of <PageSize> bytes. For example if the flash requires to be programmed in multiple of 128 bytes, <PageSize> should be set to 128. If the flash can be programmed 4 byte wise, <PageSize> should be set to 4.
Reserved	Set this element to 0
ErasedVal	Most flashes have an erased value of 0xFF (set this element to 0xFF in such cases). However, some flashes have different erased value like for example 0x00 (set element to 0x00 for such cases)
TimeoutProg	Timeout in milliseconds (ms) to program one chunk of <PageSize>. Since SEGGER_FL_Program can handle multiple pages at once, the J-Link software calculates how many pages should be programmed and accordingly calculates the total timeout for one call of SEGGER_FL_Program. Therefore the total programming timeout for one call of SEGGER_FL_Program is NumPagesAtOnce * TimeoutProg.
TimeoutErase	Timeout in milliseconds (ms) to erase one sector. Since SEGGER_FL_Erase can handle multiple sectors at once, the J-Link software calculates how many sectors should be erased and accordingly calculates the total timeout for one call of SEGGER_FL_Erase. Therefore the total erase timeout for one call of SEGGER_FL_Erase is NumSectorsAtOnce * TimeoutErase.
SectorInfo	This element is actually a list of **different sector sizes** present on target flash. Having a flash with uniform sectors will result in only SectorInfo[0] being used for sectorization information. In case the initial number of [4] regions is not sufficient, the array may be expanded up to [8] by the user when writing the flash algorithm. For more information, see the example below.

Example for SectorInfo[] member

```
// Example device with 256 KB divided into:
//   4 *  16 KB sectors
//   1 *  64 KB sectors
//   1 * 128 KB sectors
//
// FlashDevice.TotalSize:
//   0x40000
//
// FlashDevice.SectorInfo[]:
//   {
//       SectSize    StartAddr
//     { 0x00004000, 0x00000000 },   // 4 *  16 KB =  64 KB
//     { 0x00010000, 0x00010000 },   // 1 *  64 KB =  64 KB
//     { 0x00020000, 0x00020000 },   // 1 * 128 KB = 128 KB
//     { 0xFFFFFFFF, 0xFFFFFFFF }    // Indicates the end of the flash sector layout. Must be present.
//   }
//
//
// Example device with 256 KB divided into:
// 256 *   1 KB sectors
//
// FlashDevice.TotalSize:
//   0x40000
//
// FlashDevice.SectorInfo[]:
//   {
//       SectSize    StartAddr
//     { 0x00000400, 0x00000000 },   // 256 *  1 KB =  256 KB
//     { 0xFFFFFFFF, 0xFFFFFFFF }    // Indicates the end of the flash sector layout. Must be present.
//   }
//
```

Integrate SFL in J-Link software

For more information about how to actually integrate the resulting SFL binary into the J-Link software and connect it with support for your new device, please refer to the J-Link DSK knowledge base article

Stack usage

J-Link reserves 512 bytes stack for the SEGGER Flash Loader. This value is fixed and cannot be changed. If 512 bytes cannot be allocated (e.g. for devices with small RAM areas), 256 bytes are used as fallback.

Section layout

The J-Link software expects a special layout when it comes to RO code, RO data, RW data placement in the SFL binary.
The reference algorithms and templates provided with the J-Link DSK already take care of that layout, so there is nothing special to be considered by the user.

The section layout is as expected:

```
section PrgCode                    // Marks the start of the SFL. Must be the very first section
sections .text, .rodata, ...       // In any order
section PrgData                    // Marks the end of the code + rodata region (functions, const data, ...) and
the start of the data region (static + global variables)
sections .textrw, .data, .fast, ... // In any order
```

```
section DevDscr                    // Marks the location of the <FlashDevice> structure variable and also the end
of the loader. Must(!!!) be the very last section
```

Templates

There are templates as well as functioning reference loaders available in source. These are part of the J-Link Device Support Kit (DSK).

Troubleshooting

General considerations

1. Get the latest version of the template project
2. Follow the Step-By-Step instructions except of 1.2 FlashPrg.c --> Functions should not contain any code that accesses any SFRs
3. Build the flash loader using the release configuration
4. Perform a flash download using J-Link Commander. Flash download should report an error during verify
5. Implement SEGGER_FL_Erase() and retry the flash download test. J-Link Commander should still report verify failed but effected flash memory region should be empty. If not, check SEGGER_FL_Erase().
6. Implement SEGGER_FL_Program() and retry the flash download test. Expected result: Test reports O.K. --> Programmed successfully. If not, check SEGGER_FL_Program() code.

PC has unexpected value after flash download

This error may have different root causes:

- Watchdog is enabled but not fed in the flash loader functions. This may result in a watchdog timeout pops up during RAMCode execution. The behavior is different but usually a reset will be triggered.
- Accessing not enabled / clocked special function registers / peripherals
- Accessing invalid memory regions (reserved)

External flash pin init

The SEGGER_FL_Prepare() code has to make sure that the (QSPI) pins as well as the (QSPI) controller are configured so that the flash is memory-mapped read-accessible.
This is necessary as the J-Link software by default compares flash contents before starting the flash programming, to save time in case large portions of the flash are already identical to the programming data.
This can be validated by setting the compare method in J-Link Commander to "skip" (exec SetCompareMode 0). Now start flash download.
J-Link Commander should report a verify error but the flash should be memmory mapped accessible from now. If not, check the Init() code.

I get build errors in release build config

If you get build errors when switching from "Debug" to "Release" build in the template project the main cause is that third party libraries are used.
The SEGGER Flash Loader interface expects all program parts to be linked to sections PrgCode and PrgData.
Third party libraries often use statically linked program parts which will not be put in the aforementioned sections which will cause a build or linker error.
Generally we recommend not using any third party libraries when creating Flash loaders as they will increase the Flash loader size drastically while usually slowing down the maximum possible

Flash loader speed.
If you happen to have to use an external library in your project it is user responsibility to make sure this external library is linked to the aforementioned sections for all application parts.

SFL FAQ

Q: Do I need a commercial Embedded Studio license when using the SEGGER Flash Loader?
A: Basically, any usage of ES in a commercial scope requires a commercial use license. However, the SEGGER Flash Loader is an exception. The SEGGER's Friendly License suffice; a commercial use license is not required.

Q: How is the *.elf file that is the result from building the SFL executed?
A: The *.elf file is loaded into the RAM of the target device and executed on its processor.

Q: How are the functions like "SEGGER_FL_Program()" called and executed on the target?
A: The J-Link DLL analyzes the ELF file to determine the function entry points and then calls those functions.

Q: What does the "SEGGER_FL_Lib_<CORE>_LE.a" file contain and how are they used by SEGGER flash loader?
A: The file contains generic (not device specific) logic which interacts with the custom Flashloader functions SEGGER_FL_*. It needs to be included in any build.

Q: There are two files "SEGGER_FL_Lib_<CORE>_**LE**.a" and "SEGGER_FL_Lib_<CORE>_**BE**.a". Which one do I have to use?
A: "SEGGER_FL_Lib_<CORE>_**LE**.a" is used for **L**ittle **E**ndian targets while "SEGGER_FL_Lib_<CORE>_**BE**.a" is used for **B**ig **E**ndian targets.

Q: My flash loader uses interrupts. It works when debugging, but does not work when integrated with J-Link software. How can this be solved?
A: Before calling the flash loader functions, J-Link will disable interrupts on the device. If you require interrupts, you will need to reenable those e.g. in SEGGER_FL_Prepare().

Retrieved from "https://kb.segger.com/index.php?title=SEGGER_Flash_Loader&oldid=27269"

This page was last edited on 8 December 2025, at 18:08.