Application Programmer's Interface

for

MSP430 Flash Bootloader (BLMSPF)

18 July, 2002

From

SoftBaugh

Custom Software, Firmware, Hardware,
and Project Management Consulting

5040 Oakmont Bend Drive
Alpharetta, GA 30004 USA

Table of Contents

Table of Tables

Table of Figures

1. Overview

The MSP430 Flash Bootloader (BLMSPF) consists of the following components:

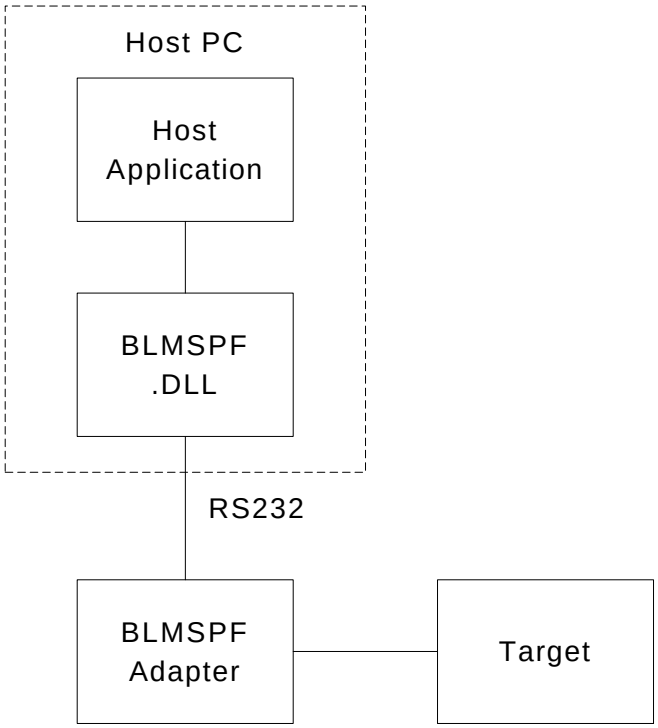

Figure 1-1 BLMSPF Components

The *Host Application* uses the *BLMSPF.DLL* to access the *BLMSPF Adapter* over a serial port. The *BLMSPF Adapter* interacts with the *Target* to perform operations commanded by the *Host Application*.

This SDK consists of the BLMSPF.DLL, the header file BLMSPF.h, and the import library BLMSPF.lib. The remainder of this document provides instructions on controlling the BLMSPF hardware adapter from your application.

Refer to documentation available from Texas Instruments for further detail of the implementation of the bootloader ROM in the various flash MSP430 versions. Please pay particular attention to the sequence of operations required for various types of access.

2. Sample Applications

The Visual C++ 6.0 sample applications included with this SDK provide the source code for useful examples of how to interact with the target.

When using the samples, or your own application, it is important to remember to place BLMSPF.DLL and Patch.txt in the same folder as your application's executable.

2.1 ProgressWriter

The ProgressWriter sample is a simpler version of the BootWrite application. It demonstrates basic write/verify functionality of the DLL, including how to process text and progress messages.

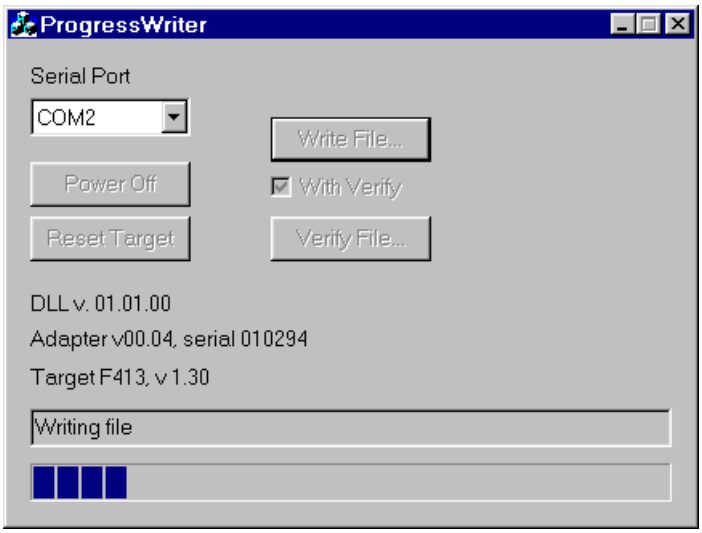

Figure 2-1 Progress Writer Screen Shot

2.2 Twiddler

The Twiddler sample application uses the LoadPC feature of the bootloader to execute code stubs in flash. Using this technique, it becomes straightforward to implement manufacturing tests or calibration procedures, even after the JTAG fuse is blown.

The Twiddler places the following code stub, found in TwiddleP13.asm, into flash:

```
; This code fragment demonstrates BLMSPF LoadPC
; The linker is configured to load the .text segment at FFC0
      .include "MSP430F41X.h"

; This project will generate a misaligned PC warning in the
; simulator because of no interrupt vectors.
; Since this fragment will not be simulated, ignore this warning.

; Export the function stubs to view addresses in map file
      .global BLMSPF_SetupP13
      .global BLMSPF_ClearP13
      .global BLMSPF_SetP13

; Also configure the assembler to load at FFC0 to
; make viewing the listing file meaningful.
      .text 0FFC0h

BLMSPF_SetupP13
      bis.b #008h,&P1DIR
      ; Intentional fall-through...
BLMSPF_ClearP13
      bic.b #008h,&P1OUT
      br &00C00h

BLMSPF_SetP13
      bis.b #008h,&P1OUT
      br &00C00h

      .end
```

The branch statements implement a "return" to the bootload ROM. After this return, it is necessary to again supply the password.

For simplicity, the Twiddler demo uses blank flash, but there is no reason why the stub cannot be programmed into a blank area of an already programmed flash part. The bold reader may wish to verify this for themselves using the BootFlash utility.

The project file for this code, TwiddleP13.prj, uses a .text segment beginning at FFC0h to locate the code stub at this address. The TwiddleP13.map file contains:

```
Addr  Name
----  ----
FFC0  BLMSPF_SetupP13
```

```
FFC4   BLMSPF_ClearP13
FFCC   BLMSPF_SetP13
```

By using the LoadPC function, these code fragments implement a P1.3 configuration stub accessed at FFC0h, a P1.3 bit clear stub accessed at FFC4h, and a P1.3 bit set capability accessed at FFCCh. These fragments could in general be expanded to perform a wide variety of functionality.

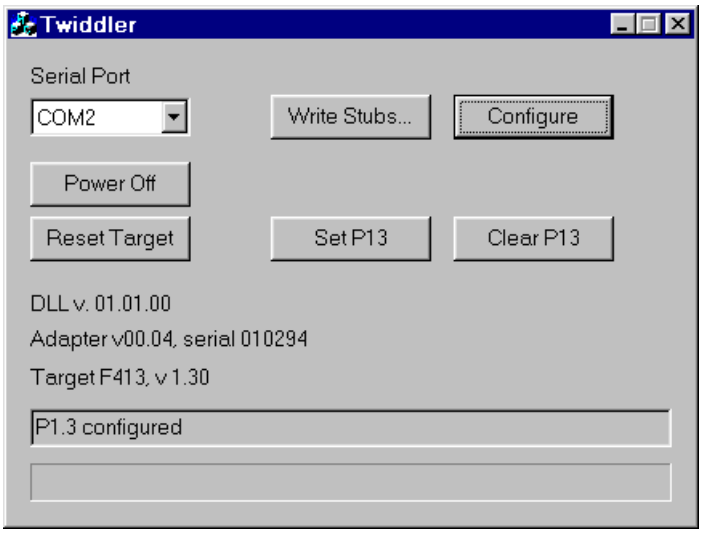

Figure 2-2 Twiddler Screen Shot

To use Twiddler, first use the Write Stubs... feature to burn the TwiddleP13.txt code into flash. Then, use the Configure... button to call into the configuration stub. Then, the SetP13 and ClearP13 will manipulate P1.3.

3. DLL Reference

The BLMSPF.DLL contains the exported functions described in this section.

3.1 Close_BLMSPF

Closes the BLMSPF interface. This function can be called repeatedly without harm.

Prototype

void Close_BLMSPF(void);

Parameters

None.

Return Value

None.

3.2 GetAdapterInfo_BLMSPF

Retrieves the adapter firmware version and adapter serial number.

Prototype

BOOL GetAdapterInfo_BLMSPF(LPWORD pwVersion, LPDWORD pdwSerNum);

Parameters

LPWORD pwVersion: Pointer to DWORD to receive the firmware version.

LPDWORD pdwSerNum: Pointer to DWORD to receive the adapter serial number.

Return Value

BOOL: If TRUE, the adapter information was successfully retrieved. If FALSE, the query failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.3 GetBootloaderVersion_BLMSPF

Retrieves the bootloader ROM version of the target. Meaningful only after a successful call to *SetPasswordData_BLMSPF* or *SetPasswordFile_BLMSPF*.

Prototype

BOOL GetBootloaderVersion_BLMSPF(LPBYTE pMajorVersion, LPBYTE pMinorVersion);

Parameters

LPBYTE pMajorVersion: Pointer to a BYTE to receive the major version.
LPBYTE pMinorVersion: Pointer to a BYTE to receive the minor version.

Return Value

BOOL: If TRUE, the version number was successfully copied. If FALSE, the function failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.4 GetLastError_BLMSPF

Retrieves a description of the previous error encountered with the BLMSPF interface.

Prototype

DWORD GetLastError_BLMSPF(LPTSTR szBuffer, DWORD dwBufferSize);

Parameters

LPTSTR szBuffer: Character buffer to receive the error message.

DWORD dwBufferSize: Size of the character buffer. If the message is longer than the buffer, it is truncated.

Return Value

DWORD: Returns the number of characters copied to the buffer.

3.5 GetPasswordData_BLMSPF

Retrieves the password data previously supplied to the interface by the host application.

Prototype

BOOL GetPasswordData_BLMSPF(LPBYTE pBuffer, DWORD dwBufferSize, LPDWORD pdwBytesRead);

Parameters

LPBYTE pBuffer: Address of an array of bytes to receive the password data.

DWORD dwBufferSize: Size of *pBuffer* in bytes.

LPDWORD pdwBytesRead: Address of DWORD to receive the number of bytes supplied in *pBuffer*.

Return Value

BOOL: If TRUE, the password bytes were written into the buffer. If FALSE, the function failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.6 GetProcessorType_BLMSPF

Returns a string identifying the target processor type. Meaningful only after a successful call to *SetPasswordData_BLMSPF* or *SetPasswordFile_BLMSPF*.

Prototype

LPCTSTR GetProcessorType_BLMSPF();

Parameters

None

Return Value

LPCTSTR: Null-terminated string identifying the target processor.

3.7 GetVersion_BLMSPF

Retrieves the current DLL version.

Prototype

LPCTSTR GetVersion_BLMSPF(void);

Parameters

None.

Return Value

LPCTSTR: Null-terminated string containing the DLL version in the format "XX.YY.ZZ".

3.8 InitializeAdapter_BLMSPF

Prepares the BLMSPF interface by establishing communication with the BLMSPF adapter.

Prototype

BOOL InitializeAdapter_BLMSPF();

Parameters

None.

Return Value

BOOL: If TRUE, the adapter was successfully initialized. If FALSE, the initialization failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.9 InitializeTarget_BLMSPF

Initializes communication with the target by performing a bootloader initialization sequence with the target.

Prototype

BOOL InitializeTarget_BLMSPF(BOOL bForcePower);

Parameters

BOOL bForcePower: If TRUE, forces a target power cycle. If FALSE, bootloader sequence is entered without a power cycle.

Return Value

BOOL: If TRUE, the target was successfully initialized. If FALSE, the initialization failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.10 IsOpen_BLMSPF

Determines whether the interface is open.

Prototype

BOOL IsOpen_BLMSPF();

Parameters

None

Return Value

BOOL: If TRUE, the API was previously successfully initialized and communication established with the hardware adapter. If FALSE, the interface is currently closed.

3.11 IsPatchRequired_BLMSPF

Determines whether the target ROM version requires a patch. Meaningful only after a successful call to *SetPasswordData_BLMSPF* or *SetPasswordFile_BLMSPF*.

Prototype

BOOL IsPatchRequired_BLMSPF();

Parameters

None.

Return Value

BOOL: If TRUE, a patch is required, FALSE if not.

3.12 LoadPC_BLMSPF

Performs a PC load and run operation.

Prototype

BOOL LoadPC_BLMSPF(DWORD dwAddress);

Parameters

DWORD dwAddress: Address to be executed.

Return Value

BOOL: If TRUE, the run operation succeeded. If FALSE, the operation failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.13 MassErase_BLMSPF

Performs a mass-erase operation on the target. After this operation, the password consisting of 32 FFh bytes can unlock the target by using the *SetPasswordData_BLMSPF* function.

Prototype

BOOL MassErase_BLMSPF();

Parameters

None.

Return Value

BOOL: If TRUE, the mass-erase operation succeeded. If FALSE, the operation failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.14 Open_BLMSPF

Opens the single allowed instance of the API on the indicated port at the indicated baud rate. If the interface was previously open, it is first closed and then reopened. After this function successfully executes, it is necessary then call *InitializeAdapter_BLMSPF* to prepare the adapter and the target for a bootloader session.

Prototype

BOOL Open_BLMSPF(LPCTSTR szPort, DWORD dwBaud);

Parameters

LPCTSTR szPort: Null-terminated string identifying the COM port to open. Ex: "COM1" opens a connection to the hardware adapter on COM1.

DWORD dwBaud: Baud rate for the connection to the hardware adapter. The current version of the DLL only supports 9600 baud.

Return Value

BOOL: If TRUE, the API was successfully initialized. If FALSE, the initialization failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

WARNING: Once the API has been opened, be sure to close communication with a later call to *Close_BLMSPF*.

3.15 PowerOff_BLMSPF

Removes power from the target.

Prototype

BOOL PowerOff_BLMSPF();

Parameters

None

Return Value

BOOL: If TRUE, power was successfully removed from the target. If FALSE, the function failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.16 PowerOn_BLMSPF

Applies power to the target without a corresponding reset. It is highly recommended to use the ResetTarget_BLMSPF function instead.

Prototype

BOOL PowerOn_BLMSPF();

Parameters

None.

Return Value

BOOL: If TRUE, power was successfully applied to the target. If FALSE, the function failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.17 ReadFile_BLMSPF

Performs a high-level read of the target into the indicated file.

Prototype

BOOL ReadFile_BLMSPF(LPCTSTR szFileName, int nFileType, DWORD dwAddress, DWORD dwSize);

Parameters

LPCTSTR szFileName: Null-terminated string containing the filename to be read from the target.

int nFileType: integer identifying the file type. Currently the following file types are supported:

BLMSPF_FILE_TYPE_TI_TEXT: File is in the Texas Instruments text format.
BLMSPF_FILE_TYPE_INTEL_HEX: File is in the Intel hex format.

DWORD dwAddress: Address in the target at which the read will begin.

DWORD dwSize: Size of data block to be read from the target in bytes.

Return Value

BOOL: If TRUE, the read operation succeeded. If FALSE, the operation failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.18 RegisterHWND_BLMSPF

Registers a HWND window handle to receive callback messages about key BLMSPF events. Only one HWND can be registered at a time.

Prototype

void RegisterHWND_BLMSPF(HWND hWnd, UINT nMsgID, DWORD dwMask);

Parameters

HWND hWnd: Window handle to which events are reported. Call with hWnd set to NULL to de-register a window.

UINT nMsgID: Message ID for the callback message. A typical value may be WM_USER+100.

DWORD dwMask: Mask of message types to be sent to the HWND. To be notified with all messages, use BLMSPF_MSG_TYPE_ALL.

Return Value

None

3.19 ResetTarget_BLMSPF

Applies power to and resets the target to allow it to execute any user program currently in flash.

Prototype

BOOL ResetTarget_BLMSPF();

Parameters

None.

Return Value

BOOL: If TRUE, the target was successfully reset. If FALSE, the reset failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.20 RxData_BLMSPF

Performs a low-level read of the target.

Prototype

BOOL RxData_BLMSPF(DWORD dwAddress, LPBYTE pBuffer, DWORD dwCount, LPDWORD pdwBytesRead);

Parameters

DWORD dwAddress: Target address at which the read is to begin.

LPBYTE pBuffer: Pointer to a buffer to receive the bytes read.

DWORD dwCount: Count of bytes to read (must be even).

LPDWORD pdwBytesRead: Address of a DWORD to receive the count of bytes actually read.

Return Value

BOOL: If TRUE, the read operation succeeded. If FALSE, the operation failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.21 SectorErase_BLMSPF

Performs a sector erase operation.

Prototype

BOOL SectorErase_BLMSPF(DWORD dwSectorAddress);

Parameters

DWORD dwSectorAddress: Address of the sector to be erased.

Return Value

BOOL: If TRUE, the sector was erased. If FALSE, the operation failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.22 SetPasswordData_BLMSPF

Sets the target password based on a memory buffer.

Prototype

BOOL SetPasswordData_BLMSPF(LPBYTE abyPassword);

Parameters

LPBYTE abyPassword: Address of an array of BLMSPF_PASSWORD_LENGTH bytes containing the password to be applied to the target.

Return Value

BOOL: If TRUE, the password was successfully recognized by the target, unlocking the flash for random access. If FALSE, the initialization failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.23 SetPasswordFile_BLMSPF

Sets the target password based on a file.

Prototype

BOOL SetPasswordFile_BLMSPF(LPCTSTR szFileName, int nFileType);

Parameters

LPCTSTR szFileName: Null-terminated string containing the filename to be verified against the target.

int nFileType: integer identifying the file type. Currently the following file types are supported:

BLMSPF_FILE_TYPE_TI_TEXT: File is in the Texas Instruments text format.
BLMSPF_FILE_TYPE_INTEL_HEX: File is in the Intel hex format.

Return Value

BOOL: If TRUE, the password was successfully recognized by the target, unlocking the flash for random access. If FALSE, the initialization failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.24 SetPatchFile_BLMSPF

Defines the patch file required for certain ROM versions. Texas Instruments may release new versions of the patch.

Prototype

BOOL SetPatchFile_BLMSPF(LPCTSTR szPatchFileName);

Parameters

LPCTSTR szPatchFileName: Null-terminated string identifying the patch file to be used. This value is stored for later use during operations requiring the patch file.

Return Value

BOOL: If TRUE, the patch file path was successfully stored. If FALSE, the function failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.25 TxData_BLMSPF

Performs a low-level write of the target.

Prototype

BOOL TxData_BLMSPF(DWORD dwAddress, LPBYTE pBuffer, DWORD dwCount, BOOL bWithVerify);

Parameters

DWORD dwAddress: Target address of the first byte to be written

LPBYTE pBuffer: Pointer to a buffer of bytes to be written.

DWORD dwCount: Count of bytes to be written (must be even).

BOOL bWithVerify: If TRUE, the write operation will be verified.

Return Value

BOOL: If TRUE, the write operation succeeded. If FALSE, the operation failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.26 VerifyData_BLMSPF

Low-level verification of the target against a user-supplied buffer.

Prototype

BOOL VerifyData_BLMSPF(DWORD dwAddress, LPBYTE pBuffer, DWORD dwCount);

Parameters

DWORD dwAddress: Target address of the first byte to be verified.

LPBYTE pBuffer: Pointer to a buffer of bytes to be verified.

DWORD dwCount: Count of bytes to be verified (must be even).

Return Value

BOOL: If TRUE, the verify operation succeeded. If FALSE, the operation failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.27 VerifyFile_BLMSPF

Performs a high-level verify of the target.

Prototype

BOOL VerifyFile_BLMSPF(LPCTSTR szFileName, int nFileType);

Parameters

LPCTSTR szFileName: Null-terminated string containing the filename to be verified against the target.

int nFileType: integer identifying the file type. Currently the following file types are supported:

BLMSPF_FILE_TYPE_TI_TEXT: File is in the Texas Instruments text format.
BLMSPF_FILE_TYPE_INTEL_HEX: File is in the Intel hex format.

Return Value

BOOL: If TRUE, the verify operation succeeded. If FALSE, the operation failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

3.28 WriteFile_BLMSPF

Performs a high-level write of a file to the target.

Prototype

BOOL WriteFile_BLMSPF(LPCTSTR szFileName, int nFileType, BOOL bWithVerify);

Parameters

LPCTSTR szFileName: Null-terminated string containing the filename to be written to the target.

int nFileType: integer identifying the file type. Currently the following file types are supported:

BLMSPF_FILE_TYPE_TI_TEXT: File is in the Texas Instruments text format.

BLMSPF_FILE_TYPE_INTEL_HEX: File is in the Intel hex format.

BOOL bWithVerify: If TRUE, the written data is verified. If FALSE, the written data is not verified.

Return Value

BOOL: If TRUE, the write operation succeeded. If FALSE, the operation failed. The cause of the failure can be identified by calling *GetLastError_BLMSPF*.

4. HWND Callbacks

Through use of the RegisterHWND_BLMSPF function, internal operation of the DLL can be reported to the host application. This section discusses in detail the callback mechanism.

Note: The BLMSPF DLL is single-threaded; all callbacks are implemented as SendMessage calls.

Four event classes are reported to the host application via the callback mechanism:

Table 4-1 Message Event Classes

Event Class	WPARAM Value
Text messages	BLMSPF_MSG_TYPE_TEXT
Named text messages	BLMSPF_MSG_TYPE_NAMED_TEXT
Buffer messages	BLMSPF_MSG_TYPE_BUFFER
Progress messages	BLMSPF_MSG_TYPE_PROGRESS

Each of these event classes has a corresponding structure providing additional information via the LPARAM.

4.1 Text Messages

A text message is a simple prompt passed via the BLMSPF_MSG_TEXT structure:

```
typedef struct tagBLMSPF_MSG_TEXT
{     BOOL bCRLF;
      LPCTSTR szText;
} BLMSPF_MSG_TEXT;
```

Cast LPARAM to `(BLMSPF_MSG_TEXT*)` to access this structure.

bCRLF is a flag suggesting whether a CRLF should be appended to the string for convenient display in a log file.

szText is a null-terminated string containing the prompt, suitable for use in a message box or single-line edit control.

4.2 Named Text Messages

A named text message is a prompt relevant to the host or the target, passed via the BLMSPF_MSG_NAMEDTEXT structure:

```
typedef struct tagBLMSPF_MSG_NAMEDTEXT
{     int nID;
      LPCTSTR szText;
} BLMSPF_MSG_NAMEDTEXT;
```

Cast LPARAM to (`BLMSPF_MSG_NAMEDTEXT*`) to access this structure.

nID indicates whether the message is relevant to the host, the target, or neither based on the following constants:

BLMSPF_HOST_ID: Message is relevant to the host.
BLMSPF_REMOTE_ID: Message is relevant to the target.
BLMSPF_NULL_ID: Message is of global scope.

szText is a null-terminated string containing the prompt, suitable for use in a message box or single-line edit control.

4.3 Buffer Messages

Buffer messages are only used for development of the BLMSPF API, and are not documented for public use.

4.4 Progress Messages

Progress messages report interim status of lengthy operations suitable for use with progress controls using parameters passed by the BLMSPF_MSG_PROGRESS structure:

```
typedef struct tagBLMSPF_MSG_PROGRESS
{     int nProgressType;
      int nProgress;
      int nMax;
} BLMSPF_MSG_PROGRESS;
```

Cast LPARAM to (`BLMSPF_MSG_PROGRESS*`) to access this structure.

nProgressType: Five operations are subject to progress messages:

Table 4-2 Progress Operations

Operation	BLMSPF_MSG_PROGRESS.nProgressType
Target initialization	BLMSPF_PROGRESS_TYPE_INIT_TARGET
Target reset	BLMSPF_PROGRESS_TYPE_RESET_TARGET
Write file	BLMSPF_PROGRESS_TYPE_WRITE_FILE
Read file	BLMSPF_PROGRESS_TYPE_READ_FILE
Verify file	BLMSPF_PROGRESS_TYPE_VERIFY_FILE

nProgress: The current progress value in the range 0 to *nMax*.

nMax: The maximum value of the progress operation.