BINLOG

Manual

Version 1.12 of 2022-06-21

Status	Completed
Publisher	Vector Informatik GmbH

Change History

Date	Changes (Author)
2020-02-12	Imported manual into new template
2020-02-20	Mark functions not supported under Linux
2020-10-23	Added functions BLSetCommentAttributeString, BLGetCommentAttributeString
2020-11-23	Added functions BLGetNumCommentAttributes, BLGetCommentAttributeName
2021-04-30	Added function BLPeekTimestamp
2021-09-16	Removed personal information from exported pdf
2021-10-27	Added function BLCreateFileEx3W

Contents

1 Introduction 5

2 BL Functions 6

2.1 Overview 6
2.2 BLCreateFile 9
2.3 BLCreateFileW 9
2.4 BLCreateFileEx 10
2.5 BLCreateFileExW 11
2.6 BLCreateFileEx2 12
2.7 BLCreateFileEx2W 13
2.8 BLCreateFileEx3W 14
2.9 BLCloseHandle 15
2.10 BLWriteObject 15
2.11 BLPeekObject 16
2.12 BLSkipObject 16
2.13 BLReadObject (Obsolete) 17
2.14 BLReadObjectSecure 17
2.15 BLFreeObject 18
2.16 BLSeekTime 19
2.17 BLSetApplication 20
2.18 BLSetWriteOptions 21
2.19 BLSetMeasurementStartTime 21
2.20 BLGetFileStatistics 22
2.21 BLGetFileStatisticsEx 22
2.22 BLFlushFileBuffers 23
2.23 BLSetCommentAttributeString 23
2.24 BLGetNumCommentAttributes 24
2.25 BLGetCommentAttributeName 24
2.26 BLGetCommentAttributeString 25
2.27 BLPeekTimestamp 26

3 BL structures 28

3.1 Overview 28
3.2 VBLObjectHeaderBase 28
3.3 VBLFileStatistics 28
3.4 VBLFileStatisticsEx 29

4 Additional informations 30

4.1 Extended CAN identifiers 30

5 License 31
5.1 Acknowledgement 31
5.2 The zlib Software License 31

1 Introduction

This document describes the usage of the binlog library provided with CANoe/CANalyzer.

The BL package is installed in the folder `Programming\BLF_Logging` of the CANoe/CANalyzer installation.

Besides the binlog header file in the `Include` folder, a sample CMake project is provided in the subfolder `BLF_Project`, which demonstrates the usage of the binlog library. The resulting sample program `bl` creates the BL file `test.blf`.

In the subfolder `Demo` CANoe/CANalyzer configurations are provided, which make use of the generated sample BL file.

2 BL Functions

2.1 Overview

```
BLAPI( BLHANDLE) BLCreateFile( const char * lpFileName,
                               uint32_t dwDesiredAccess);

BLAPI( BLHANDLE) BLCreateFileW( const wchar_t * lpFileName,
                               uint32_t dwDesiredAccess);

BLAPI( BLHANDLE) BLCreateFileEx( const char * lpFileName,
                               uint32_t dwDesiredAccess,
                               const char * lpServer,
                               const char * lpHost);

BLAPI( BLHANDLE) BLCreateFileExW( const wchar_t * lpFileName,
                               uint32_t dwDesiredAccess,
                               const wchar_t * lpServer,
                               const wchar_t * lpHost);

BLAPI( BLHANDLE) BLCreateFileEx2( const char * lpFileName,
                               uint32_t dwDesiredAccess,
                               const char * lpServer,
                               const char * lpHost,
                               IBLCallback* pCallback);

BLAPI( BLHANDLE) BLCreateFileEx2W( const wchar_t * lpFileName,
                               uint32_t dwDesiredAccess,
                               const wchar_t * lpServer,
                               const wchar_t * lpHost,
                               IBLCallback* pCallback);

BLAPI( BLHANDLE) BLCreateFileEx3W( const wchar_t * lpFileName,
                               uint32_t dwDesiredAccess,
                               const wchar_t * lpServer,
                               void * pProviderInfo,
                               const wchar_t * lpHost,
```

```
                                IBLCallback* pCallback);

BLAPI( int32_t)  BLCloseHandle( BLHANDLE hFile);

BLAPI( int32_t)  BLWriteObject( BLHANDLE hFile,
                                VBLObjectHeaderBase* pBase);

BLAPI( int32_t)  BLPeekObject(BLHANDLE hFile,
                                VBLObjectHeaderBase* pBase);

BLAPI( int32_t)  BLSkipObject(BLHANDLE hFile,
                                VBLObjectHeaderBase* pBase);

BLAPI( int32_t)  BLReadObject(BLHANDLE hFile,
                                VBLObjectHeaderBase* pBase);

BLAPI( int32_t)  BLReadObjectSecure(BLHANDLE hFile,
                               VBLObjectHeaderBase* pBase,
                                size_t expectedSize);

BLAPI( int32_t)  BLFreeObject(BLHANDLE hFile,
                                VBLObjectHeaderBase* pBase);

BLAPI( int32_t)  BLSeekTime( BLHANDLE hFile,
                   uint64_t timeStamp,
                   void* arg,
                   int32_t(*pProgressCallback)(void*, float),
                   uint16_t callbackRate);

BLAPI( int32_t)  BLSetApplication(BLHANDLE hFile,
                                     uint8_t appID,
                                     uint8_t appMajor,
                                     uint8_t appMinor,
                                     uint8_t appBuild);
```

```
BLAPI( int32_t)  BLSetWriteOptions( BLHANDLE hFile,
                                    uint32_t dwCompression,
                                    uint32_t dwReserved);

BLAPI( int32_t)  BLSetMeasurementStartTime( BLHANDLE hFile,
                                const LPSYSTEMTIME lpStartTime);

BLAPI( int32_t)  BLGetFileStatistics( BLHANDLE hFile,
                                VBLFileStatistics* pStatistics);

BLAPI( int32_t)  BLGetFileStatisticsEx( BLHANDLE hFile,
                                VBLFileStatisticsEx* pStatistics);

BLAPI( int32_t)  BLFlushFileBuffers( BLHANDLE hFile,
                                uint32_t dwFlags);

BLAPI( int32_t)  BLSetCommentAttributeString( BLHANDLE hFile,
                                const wchar_t* lpName,
                                const wchar_t* lpValue);

BLAPI( int32_t)  BLGetNumCommentAttributes( BLHANDLE hFile,
                                uint32_t* lpNumAttributes);

BLAPI( int32_t)  BLGetCommentAttributeName(BLHANDLE hFile,
                                uint32_t dwIndex,
                                wchar_t* lpName,
                                uint32_t* lpNameSize);

BLAPI( int32_t)  BLGetCommentAttributeString(BLHANDLE hFile,
                                const wchar_t* lpName,
                                wchar_t* lpValue,
                                uint32_t* lpValueSize);

BLAPI( int32_t) BLPeekTimestamp(BLHANDLE hFile,
                                const VBLObjectHeaderBase* pBase,
                                uint64_t* lpTimestamp);
```

2.2 BLCreateFile

Syntax	`BLAPI(BLHANDLE) BLCreateFile(`
	`  const char *lpFileName,`
	`  uint32_t dwDesiredAccess)`

Description	Use this function to open a BL file with the desired access.

Parameters	`const char * lpFileName`
	Pointer to a null-terminated string that specifies the name of the file to create or open.
	`uint32_t dwDesiredAccess`
	Specifies the type of access to the file. An application can obtain read access or write access. This parameter can be GENERIC_READ or GENERIC_WRITE.

Return values	If the function succeeds, the return value is an open handle to the specified file. If the function fails, the return value is BLINVALID_HANDLE_VALUE.

2.3 BLCreateFileW

Syntax	`BLAPI(BLHANDLE) BLCreateFileW(`
	`  const wchar_t * lpFileName,`
	`  uint32_t dwDesiredAccess)`

Description	Use this function to open a BL file with the desired access.

Parameters	`const wchar_t * lpFileName`
	Pointer to a null-terminated wide string that specifies the name of the file to create or open.
	`uint32_t dwDesiredAccess`
	Specifies the type of access to the file. An application can obtain read access or write access. This parameter can be GENERIC_READ or GENERIC_WRITE.

Return values	If the function succeeds, the return value is an open handle to the specified file. If the function fails, the return value is BLINVALID_HANDLE_VALUE.

2.4 BLCreateFileEx

Syntax	**BLAPI(BLHANDLE) BLCreateFileEx(** **const char * lpFileName,** **uint32_t dwDesiredAccess,** **const char * lpServer,** **const char * lpHost)**
Description	Use this function to open a BL file with the desired access. Under Linux `lpServer` and `lpHost` are ignored.

Parameters	`const char * lpFileName`
	Pointer to a null-terminated string that specifies the name of the file to create or open.
	`uint32_t dwDesiredAccess`
	Specifies the type of access to the file. An application can obtain read access or write access. This parameter can be GENERIC_READ or GENERIC_WRITE.
	`const char * lpServer`
	Pointer to a null-terminated string that specifies an external logging provider, with the syntax <GUID>|<dll name>. If a null-pointer is passed, no external logging provider is used
	`const char * lpHost`
	Pointer to a null-terminated string that specifies a logging host. Currently unused.

Return values	If the function succeeds, the return value is an open handle to the specified file. If the function fails, the return value is BLINVALID_HANDLE_VALUE.

2.5 BLCreateFileExW

Syntax	`BLAPI( BLHANDLE) BLCreateFileExW(`
	`  const wchar_t * lpFileName,`
	`  uint32_t dwDesiredAccess,`
	`  const wchar_t * lpServer,`
	`  const wchar_t * lpHost)`

Description	Use this function to open a BL file with the desired access. Under Linux `lpServer` and `lpHost` are ignored.

Parameters	`const wchar_t * lpFileName`
	Pointer to a null-terminated wide string that specifies the name of the file to create or open.
	`uint32_t dwDesiredAccess`
	Specifies the type of access to the file. An application can obtain read access or write access. This parameter can be GENERIC_READ or GENERIC_WRITE.
	`const wchar_t * lpServer`
	Pointer to a null-terminated wide string that specifies an external logging provider, with the syntax <GUID>|<dll name>. If a null-pointer is passed, no external logging provider is used
	`const wchar_t * lpHost`
	Pointer to a null-terminated wide string that specifies a logging host. Currently unused.

Return values	If the function succeeds, the return value is an open handle to the specified file. If the function fails, the return value is BLINVALID_HANDLE_VALUE.

2.6 BLCreateFileEx2

Syntax	**BLAPI(BLHANDLE) BLCreateFileEx2(** **const char * lpFileName,** **uint32_t dwDesiredAccess,** **const char * lpServer,** **const char * lpHost,** **IBLCallback* pCallback)**
Description	Use this function to open a BL file with the desired access. Under Linux `lpServer` and `lpHost` are ignored.

Parameters	`const char * lpFileName`
	Pointer to a null-terminated string that specifies the name of the file to create or open.
	`uint32_t dwDesiredAccess`
	Specifies the type of access to the file. An application can obtain read access or write access. This parameter can be GENERIC_READ or GENERIC_WRITE.
	`const char * lpServer`
	Pointer to a null-terminated string that specifies an external logging provider, with the syntax <GUID>|<dll name>. If a null-pointer is passed, no external logging provider is used
	`const char * lpHost`
	Pointer to a null-terminated string that specifies a logging host. Currently unused.
	`IBLCallback* pCallback`
	Pointer to a callback function where binlog can write status and error messages. If a null-pointer is passed, the messages are supressed.

Return values	If the function succeeds, the return value is an open handle to the specified file. If the function fails, the return value is INVALID_HANDLE_VALUE.

2.7 BLCreateFileEx2W

Syntax	`BLAPI(BLHANDLE) BLCreateFileEx2W(`
	`  const wchar_t * lpFileName,`
	`  uint32_t dwDesiredAccess,`
	`  const wchar_t * lpServer,`
	`  const wchar_t * lpHost,`
	`  IBLCallback* pCallback)`

Description	Use this function to open a BL file with the desired access. Under Linux `lpServer`, `lpHost` and `pCallback` are ignored.

Parameters	`const wchar_t * lpFileName`
	Pointer to a null-terminated wide string that specifies the name of the file to create or open.
	`uint32_t dwDesiredAccess`
	Specifies the type of access to the file. An application can obtain read access or write access. This parameter can be GENERIC_READ or GENERIC_WRITE.
	`const wchar_t * lpServer`
	Pointer to a null-terminated wide string that specifies an external logging provider, with the syntax <GUID>|<dll name>. If a null-pointer is passed, no external logging provider is used
	`const wchar_t * lpHost`
	Pointer to a null-terminated wide string that specifies a logging host. Currently unused.
	`IBLCallback* pCallback`
	Pointer to a callback function where binlog can write status and error messages. If a null-pointer is passed, the messages are supressed.

Return values	If the function succeeds, the return value is an open handle to the specified file. If the function fails, the return value is BLINVALID_HANDLE_VALUE.

2.8 BLCreateFileEx3W

Syntax	`BLAPI(BLHANDLE) BLCreateFileEx3W(`
	`  const wchar_t * lpFileName,`
	`  uint32_t dwDesiredAccess,`
	`  const wchar_t * lpServer,`
	`  void * pProviderInfo,`
	`  const wchar_t * lpHost,`
	`  IBLCallback* pCallback)`

Description	Use this function to open a BL file with the desired access , callback function, and additional info for an external provider. Under Linux `lpServer`, `lpHost` and `pCallback` are ignored.

Parameters	`const wchar_t * lpFileName`
	Pointer to a null-terminated wide string that specifies the name of the file to create or open.
	`uint32_t dwDesiredAccess`
	Specifies the type of access to the file. An application can obtain read access or write access. This parameter can be GENERIC_READ or GENERIC_WRITE.
	`const wchar_t * lpServer`
	Pointer to a null-terminated wide string that specifies an external logging provider, with the syntax <GUID>|<dll name>. If a null-pointer is passed, no external logging provider is used
	`void * pProviderInfo`
	Pointer to additional info that is passed to the external logging provider.
	`const wchar_t * lpHost`
	Pointer to a null-terminated wide string that specifies a logging host. Currently unused.
	`IBLCallback* pCallback`
	Pointer to a callback function where binlog can write status and error messages. If a null-pointer is passed, the messages are supressed.

Return values	If the function succeeds, the return value is an open handle to the specified file. If the function fails, the return value is BLINVALID_HANDLE_VALUE.

2.9 BLCloseHandle

Syntax	**BLAPI(int32_t) BLCloseHandle(BLHANDLE hFile)**
Description	Use this function to close a BL file opened with BLCreateFile.
Parameters	`BLHANDLE hFile` The file handle returned by BLCreateFile.
Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.10 BLWriteObject

Syntax	**BLAPI(int32_t) BLWriteObject(BLHANDLE hFile, VBLObjectHeaderBase* pBase)**
Description	Use this function to write a BL object to the file.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_WRITE access to the file.
	`VBLObjectHeaderBase* pBase`
	Pointer to a BL object structure containing the data to be written to the file.

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.11 BLPeekObject

Syntax	**BLAPI(int32_t) BLPeekObject(BLHANDLE hFile, VBLObjectHeaderBase* pBase)**
Description	Use this function to read the base header part of a BL object.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`VBLObjectHeaderBase* pBase`
	Pointer to a BL object structure that receives the object header description.

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.12 BLSkipObject

Syntax	**BLAPI(int32_t) BLSkipObject(BLHANDLE hFile, VBLObjectHeader Base* pBase)**
Description	Use this function to skip a BL object.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`VBLObjectHeaderBase* pBase`
	Pointer to a BL object structure that describes the object to be skipped.

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.13 BLReadObject (Obsolete)

Obsolete: This function has been replaced by BLReadObjectSecure.

Syntax	**BLAPI(int32_t) BLReadObject(BLHANDLE hFile, VBLObjectHeaderBase* pBase)**
Description	Use this function to read a BL object.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`VBLObjectHeaderBase* pBase`
	Pointer to a BL object structure that describes the object to be read.

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.14 BLReadObjectSecure

Syntax	**BLAPI(int32_t) BLReadObjectSecure(BLHANDLE hFile, VBLObjectHeaderBase* pBase, size_t expectedSize)**
Description	Use this function to read a BL object.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`VBLObjectHeaderBase* pBase`
	Pointer to a BL object structure that describes the object to be read.
	`size_t expectedSize`
	Size of BL object structure which is provided by pointer pBase.

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.15 BLFreeObject

Syntax	**BLAPI(int32_t) BLFreeObject(** **BLHANDLE hFile,** **VBLObjectHeaderBase* pBase)**
Description	Use this function to free the memory which has been allocated for a previously read BL object. Altough this is only required for dynamic sized objects such as environment variables it doesn't harm to call this method for fixed sized objects like CAN messages as well.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`VBLObjectHeaderBase* pBase`
	Pointer to a BL object structure that describes the object to be freed.

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.16 BLSeekTime

Syntax	**BLAPI(int32_t) BLSeekTime (** **BLHANDLE hFile,** **uint64_t timeStamp,** **void* arg,** **int32_t(*pProgressCallback)(void*, float),** **uint16_t callbackRate)**
Description	Use this function to seek forward in a BLF file to the first object with a certain time stamp. This function is not supported under Linux.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`uint64_t timestamp`
	The time stamp value you are searching for.
	`void* arg`
	Argument which is passed back to the pProgressCallback call. It can be used as a bridge between the C-Style binlog interface and C++ (by passing the class this pointer).
	`int32_t (*pProgressCallback) (void*, float)`
	Callback function, which passes back the arg pointer and the progress value (between 0 and 1.0).
	`uint16_t callbackRate`
	Rate how often pProgressCallback is called (in ms).

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.17 BLSetApplication

Syntax	**BLAPI(int32_t) BLSetApplication(** **BLHANDLE hFile,** **uint8_t appID,** **uint8_t appMajor,** **uint8_t appMinor,** **uint8_t appBuild)**
Description	Use this function to specify the application which writes the file.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_WRITE access to the file.
	`uint8_t appID`
	The application identifier.
	`uint8_t appMajor`
	The application major version number.
	`uint8_t appMinor`
	The application minor version number.
	`uint8_t appBuild`
	The application build version number.

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.18 BLSetWriteOptions

Syntax	**BLAPI(int32_t) BLSetWriteOptions(** **BLHANDLE hFile,** **uint32_t dwCompression,** **uint32_t dwReserved)**
Description	Use this function to set the compression.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_WRITE access to the file.
	`uint32_t dwCompression`
	The compression to be used during write. Valid values range from 0 (no compression) to 10 (maximum compression).
	`uint32_t dwReserved`
	Reserved. Must be zero.

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.19 BLSetMeasurementStartTime

Syntax	**BLAPI(int32_t) BLSetMeasurementStartTime(** **BLHANDLE hFile,** **const LPSYSTEMTIME lpStartTime);**
Description	Use this function to set the measurement start time

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_WRITE access to the file.
	`LPSYSTEMTIME lpStartTime`
	The pointer to the windows system time structure

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.20 BLGetFileStatistics

Syntax	**BLAPI(int32_t) BLGetFileStatistics(** **BLHANDLE hFile,** **VBLFileStatistics* pStatistics)**
Description	Use this function to retrieve the file statistics.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`VBLFileStatistics* pStatistics`
	The pointer to the file statistics structure.

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.21 BLGetFileStatisticsEx

Syntax	**BLAPI(int32_t) BLGetFileStatisticsEx(** **BLHANDLE hFile,** **VBLFileStatisticsEx* pStatistics)**
Description	Use this function to retrieve the extended file statistics.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`VBLFileStatisticsEx* pStatistics`
	The pointer to the extended file statistics structure.

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.22 BLFlushFileBuffers

Syntax	**BLAPI(int32_t) BLFlushFileBuffers(** **BLHANDLE hFile,** **uint32_t dwFlags)**
Description	Use this function to flush the file buffers.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_WRITE access to the file.
	`uint32_t dwFlags`
	Flag indicating how to flush. Valid values are:
	BL_FLUSH_STREAM - flushes all internal streams
	BL_FLUSH_FILE - flushes the file and combinations thereof.

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.23 BLSetCommentAttributeString

Syntax	**BLAPI(int32_t) BLSetCommentAttributeString(** **BLHANDLE hFile,** **const wchar_t* lpName,** **const wchar_t* lpValue)**
Description	Use this function to set a comment attribute.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_WRITE access to the file.
	`const wchar_t* lpName`
	The name of the comment attribute
	`const wchar_t* lpValue`
	The value of the comment attribute

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.24 BLGetNumCommentAttributes

Syntax	`BLAPI( int32_t) BLGetNumCommentAttributes(`
	`  BLHANDLE hFile,`
	`  uint32_t* lpNumAttributes)`

Description	Use this function to get the number of comment attributes in the log file.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`uint32_t* lpNumAttributes`
	The number of comment attributes will be written here

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.25 BLGetCommentAttributeName

Syntax	`BLAPI( int32_t) BLGetCommentAttributeName(`
	`  BLHANDLE hFile,`
	`  uint32_t dwIndex,`
	`  wchar_t* lpName,`
	`  uint32_t* lpNameSize)`

Description	Use this function to read the name (key) of a comment attribute. The attribute value will be written to `lpName`. On success, `TRUE` will be returned, `lpName` contains the attribute name, and `lpNameSize` contains the size in bytes of the name (including terminating 0 character). If the buffer size is too small, `lpNameSize` will contain the required size of the buffer in bytes and `FALSE` will be returned. If the logfile does not contain an attribute with the given index, `FALSE` will be returned and `lpName` will be set to 0.

Syntax	**BLAPI(int32_t) BLGetCommentAttributeName(** **BLHANDLE hFile,** **uint32_t dwIndex,** **wchar_t* lpName,** **uint32_t* lpNameSize)**

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`uint32_t dwIndex`
	The index of the comment attribute
	`wchar_t* lpName`
	Buffer where the name of the comment attribute is written to
	`uint32_t* lpNameSize`
	Initial size of the buffer; after the call, size of the written attribute name or (if initially too small) required buffer size

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.26 BLGetCommentAttributeString

Syntax	**BLAPI(int32_t) BLSetCommentAttributeString(** **BLHANDLE hFile,** **const wchar_t* lpName,** **wchar_t* lpValue,** **uint32_t valueSize)**
Description	Use this function to read a comment attribute. The attribute value will be written to `lpValue`. On success, `TRUE` will be returned, `lpValue` contains the attribute value, and `lpValueSize` contains the size in bytes of the value (including terminating 0 character). If the buffer size is too small, `lpValueSize` will contain the required size of the buffer in bytes and `FALSE` will be returned. If the logfile does not contain an attribute with the given name, `FALSE` will be returned and `lpValue` will be set to 0.

Syntax	**BLAPI(int32_t) BLSetCommentAttributeString(BLHANDLE hFile, const wchar_t* lpName, wchar_t* lpValue, uint32_t valueSize)**

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`const wchar_t* lpName`
	The name of the comment attribute
	`wchar_t* lpValue`
	Buffer where the attribute value is written to
	`uint32_t* lpValueSize`
	Initial size of the buffer; after the call, size of the written attribute or (if initially too small) required buffer size

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

2.27 BLPeekTimestamp

Syntax	**BLAPI(int32_t) BLSetCommentAttributeString(BLHANDLE hFile, const VBLObjectHeaderBase* pBase, uint64_t* lpTimestamp)**
Description	Use this function to read the timestamp in ns of the next object. The attribute value will be written to `lpTimestamp`.

Parameters	`BLHANDLE hFile`
	The file handle returned by BLCreateFile. The file handle must have been created with GENERIC_READ access to the file.
	`const VBLObjectHeaderBase pBase`
	Pointer to a BL object header structure that describes the object of which the timestamp should be extracted. This header must have been obtained by a preceding call to `BLPeekObject`.
	`uint64_t* lpTimestamp`
	Pointer where the result is written to, must be initialized.

Syntax	`BLAPI( int32_t) BLSetCommentAttributeString(`
	`  BLHANDLE hFile,`
	`  const VBLObjectHeaderBase* pBase,`
	`  uint64_t* lpTimestamp)`

Return values	If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

3 BL structures

3.1 Overview

VBLObjectHeaderBase

VBLFileStatistics

3.2 VBLObjectHeaderBase

```
typedef struct VBLObjectHeaderBase_t
{
  uint32_t mSignature;   /* signature (BL_OBJ_SIGNATURE) */
  uint16_t mHeaderSize;  /* sizeof object header */
  uint16_t mHeaderVersion; /* header version (1) */
  uint32_t mObjectSize;  /* object size */
  uint32_t mObjectType;  /* object type (BL_OBJ_TYPE_XXX) */
} VBLObjectHeaderBase;
```

3.3 VBLFileStatistics

```
typedef struct VBLFileStatistics_t
{
  uint32_t mStatisticsSize; /* sizeof (VBLFileStatistics) */
  uint8_t  mApplicationID;    /* application ID */
  uint8_t  mApplicationMajor; /* application major number */
  uint8_t  mApplicationMinor; /* application minor number */
  uint8_t  mApplicationBuild; /* application build number */
  uint64_t mFileSize; /* file size in bytes */
  uint64_t mUncompressedFileSize;
                        /* uncompressed file size in bytes */
  uint32_t mObjectCount;/* number of objects */
  uint32_t mObjectsRead;/* number of objects read */
} VBLFileStatistics;
```

3.4 VBLFileStatisticsEx

```
typedef struct VBLFileStatisticsEx_t
{
    uint32_t mStatisticsSize; /* sizeof(VBLFileStatisticsEx) */
    uint8_t mApplicationID; /* application ID */
    uint8_t mApplicationMajor;  /* application major number */
    uint8_t mApplicationMinor;  /* application minor number */
    uint8_t mApplicationBuild;  /* application build number */
    uint64_t mFileSize;   /* file size in bytes */
    uint64_t mUncompressedFileSize;
 /* uncompressed file size in bytes */
    uint32_t mObjectCount;  /* number of objects */
    uint32_t mObjectsRead;  /* number of objects read */
    SYSTEMTIME mMeasurementStartTime;
     /* measurement start time */
    SYSTEMTIME mLastObjectTime; /* last object time */
    uint32_t   mReserved[18]; /* reserved */
} VBLFileStatisticsEx;
```

4 Additional informations

4.1 Extended CAN identifiers

The following structure is used to write CAN frames:

```
typedef struct VBLCANMessage_t
{
    VBLObjectHeader mHeader;    /* object header */
    uint16_t        mChannel;   /* application channel */
    uint8_t         mFlags;     /* CAN dir & rtr */
    uint8_t         mDLC;       /* CAN dlc */
    uint32_t        mID;        /* CAN ID */
    uint8_t         mData[8];   /* CAN data */
} VBLCANMessage;
```

The member mID is used for the numeric identifier of the frame. If you want to write an extended frame identifier, you must set the highest bit of the mID field. E.g. if you want to write a frame with the extended identifier 0x100, you must do the following:

message.mID = 0x80000100

For the same frame with a standard identifier you would use the field mID in the following way:

message.mID = 0x00000100

5 License

5.1 Acknowledgement

The compression routines used in the BL library derive from the zlib library.

5.2 The zlib Software License

zlib (http://www.gzip.org/zlib/) Copyright (C) 1995-2002 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

- The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
- Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
- This notice may not be removed or altered from any source distribution.

Jean-loup Gailly (jloup@gzip.org) Mark Adler (madler@alumni.caltech.edu)

The data format used by the zlib library is described by RFCs (Request for Comments) 1950 to 1952 in the files https://www.ietf.org/rfc/rfc1950.txt (zlib format), rfc1951.txt (deflate format) and rfc1952.txt (gzip format).