Description for Integration of FlashTools in a Customer Application. As of 10/23/2003

Modifications:

Section 1.1: Description, parameters and return values for all functions revised.

Section 1.1: New function DmCheckOldFlashTools added.

Section 1.1: New function DmGetProgress added.

Description for Integration of FlashTools in a Customer Application. As of 03/26/2003

Modifications:

Section 2.2 Hint for using the Pragma *pack* when declaring structures.

Description for Integration of FlashTools in a Customer Application. As of 06/11/2002

Modifications:

Section 1.1 Modifications of the function DmGetGeneralInfo in DmGetGeneralInfos

Section 1.1 new function DmGetHexFileInfo added

Section 2 typedef tMemType added

Sectoin 2 typedef tPLProgFlash added

Section 2 typedef tRLReadFlash added

Section 2 typedef tRLFileInfo added

Section 2 typedef for return values updated

Modifications of previous versions:

Section 1.1 – Description of the function DmLoadFile:

The parameter dwStartAddress no longer exists (the address is now given with DmProgFlash)

Section 1.1 – Description of the function DmSaveFile:

Parameter added for thread structure.

Section 1.1 – Description of the function DmProgFlash

The parameter dwOffsetToAddress was added.

Section 1.1 – New function DmFreeMemory added.

This function is used to free the memory allocated in the DLL

Section 3 Implementation Hints, paragraph 1: Parameter values

The description for freeing the memory allocated in the DLL is expanded by the function DmFreeMemory.

1. Using the FlashTools Library

1.1 Integrating the DLL in a User Application

This section contains an explanation of how you can integrate the DLL PHYDm.DLL in your own application program.
The DLL enables access to the external Flash populating a PHYTEC Single Board Computer (SBC) module.

The following description refers to the programming environment *Microsoft Visual C++ 6.0*.

Place the DLL in the Window system folder or copy the DLL to your application folder. Use the API function *LoadLibrary(…)* to load the DLL and thereby enable access to the functions contained within the DLL.

```
HINSTANCE hLib;
hLib = LoadLibrary(„PHYDm.dll“);
```

A function pointer is required for each function to access the DLL functions.

Example:
Function to be used:

```
tReturnCode DmProgFlash (sThread *sProgFlashThread)
```

Declaration of the Function Pointer:

```
tReturnCode (PASCAL *lpfn_DmProgFlash) (sThread *);
```

In order to determine the starting address of the function within the DLL, the API function GetProcAddress(…) is used.

lpfn_DmProgFlash = (tReturnCode (PASCAL)(sThread) GetProcAddress (hLib, „DmProgFlash“);

Now the function can be called as follows:

lpfn_ProgFlash(&sProgFlashThread);

If the DLL is no longer required, it must be removed from the memory. This is accomplished with the API function **FreeLibrary(…).**

```
FreeLibrary(hLib);
```

Additional information about the API functions used can be found in the documentation for your development environment.

Interfaces

tReturnCode DmLogin (char szIniFileName, tRLLogin *pRetList_p, sThread *sLoginThread) ;		
	Function	This function establishes a connection to a module
	Parameter	szIniFileName: Name of the *.ini file that contains information about the module characteristics. tRLLogin *pRetList: Information about the module (Firmware version, TxBufferSize) sThread: Information for communication with the thread (ThreadID, Handle..)
	Return	kSuccessful: Command was executed without error kFailed: Aborted by user kCommandDllNotFound: Command module not found kCommandDllError: Function CmmLogin not found kInvalidFileFormat: Faulty entry in the *.ini file kNetworkDllNotFound: The network DLL given in the *.ini file was not found kNetworkDllError: Faulty network DLL

tReturnCode DmExitFlashTools (tPLExitFlashTools* pParamList_p) ;		
	Function	If the variable bStartUserApp in the structure tPLExitFlashTools is 1, then the application's start address specified in dwLogStartAddr is used for start-up. Normally this address is set to 0x00000000. In this case the address is called with the reset vector of the current controller. If the variable bStartUserApp = 2, then a software reset will be generated on the module. If the variable bStartUserApp = 3, then a hardware reset will be generated on the module. This requires hardware support on the Development Board or custom carrier board to control the BOOT and RESET signals via RTS/DTR modem lines.
	Parameter	tPLExitFlashTools: bStartUserApp and start address of the user program
	Return	kSuccessful: Command was executed without error kNetworkDllError: Faulty network DLL kNetworkReceiveFailed: Error during communication with the module kNetworkSendFailed: Error during communication with the module kInvalidFileFormat: Faulty entry in the *.ini file

tReturnCode DmFindFlashLib (tPLGetFlashLibInfo* pParamList_p, tRLGetFlashLibInfo* pRetList_p) ;		
	Function	This function demands information from the SBC about the libraries present. It is possible to determine whether the first Lib from the chain list or the next Lib is desired. Thus the list of all Libs present on the SBC can be checked.
	Parameter	tPLGetFlashLibInfo: Structure with information about FlashLib (0 = first FlashLib, 1 = next FlashLib). This enables a list with all FlashLibs on the SBC to be searched. The parameter dwFlashAddress determines the start address of the Flash. tRLGetFLashLibInfo: Structure with manufacturer, device and additional ID
	Return	kSuccessful: Command was executed without errors kCommandDllError: CommandDLL was not loaded or was faulty kInvalidFileFormat: The required information could not be acquired from the *.ini file. KnetworkReceiveFailed: Error during communication with the module KNetworkSendFailed: Error during communication with the module

tReturnCode DmSelectFlashLib (tPLSelectFlashLib* pParamList_p);		
	Function	This function selects the Flash library on the SBC for the active Flash.
	Parameter	tPLSelectFlashLib: structure with LibId, AdditionalID, read via DmFindFlashLib
	Return	kSuccessful: Command was executed without error kCommandDllError: Command DLL did not load or is faulty kNetworkReceiveFailed: Error during communication with the module kNetworkSendFailed: Error during communication with the module

tReturnCode DmEraseSectors (tPLEraseSectors* pParamList_p , tRLEraseSectors* pRetList_p, sThread* sEraseSectorsThread) ;		
	Function	This function erases all sectors located between dwStartAddress and dwEndAddress. Only the return codes of the sectors for which errors occurred while they were erased are written to the send buffer. The DLL allocates memory for each sector that couldn´t be erased and returns it in the structure tRLEraseSectors. The pointer tEraseStatus points to this memory. If this memory is no longer required by the application it must be released. This must be done by calling the DLL function DmFreeMemory().
	Parameter	tPLEraseSectors: Structure with information about start and end sectors. tRLEraseSectors: Structure with information about sectors that could not be erased. sThread: information for communication with the thread (ThreadID, Handle..)
	Return	KSuccessful: command was executed without error KFailed: user abort KNetworkReceiveFailed: error during communication with the module KNetworkSendFailed: error during communication with the module KNoLibSelected: no FlashLibrary was selected yet

tReturnCode DmEraseChip (tRLEraseSectors* pRetList_p) ;		
	Function	This function erases the entire Flash.
	Parameter	TRLEraseSectors: Structure with Errorcode. The DLL allocates memory for each sector that couldn´t be erased. The pointer to this memory is returned in tEraseStatus in the structure tRLEraseSector. This memory must be released when it is no longer required by the application. This has to occur by a call of the DLL function DmFreeMemory().
	Return	KSuccessful: command was executed without error KNetworkReceiveFailed: error during communication with the module KNetworkSendFailed: error during communication with the module KNoLibSelected: no FlashLibrary was selected yet KSoftwareProtected: a portion of the Flash is protected. Therefore the command cannot be executed

tReturnCode DmLoadFile (char* pszFileName_p, BYTE bType, sThread* sLoadFileThread) ;		
	Function	This function loads a hex file or a binary file to the memory on the host computer. Thus it is possible to program the hex/binary file on multiple targets without having to continually reload it from the hard drive.
	Parameter	pszFileName_p: Name of the hex or binary file btype: Type of file (1 = Bin/0 = Hex) sThread: Information for communication with the Thread (ThreadID, Handle..)
	Return	kSuccessful: command was executed without error kFileNotFound: file could not be opened kInvalidFileFormat: no Hex or Bin file

tReturnCode DmSaveFile (char* pszFileName_p, BYTE bType, sThread* sSaveFileThread) ;		
	Function	The function stores the data read by DmReadFlash in a file.
	Parameter	pszFileName_p: Name of the output file. btype: Type of the file (1 = bin/0 = hex) sThread: information for communication with the Thread (ThreadID, Handle..)
	Return	kSuccessful: Command was executed without error kFailed: user abort kFileNotFound: no data in internal buffer kLoadFileError: file could not be opened for writing

tReturnCode DmProgFlash (DWORD dwOffsetToAddress, sThread *sProgFlashThread);		
	Function	This function programs a hex/bin file in the Flash. The hex/bin file must be loaded with the function DmLoadFile() prior to this.
	Parameter	dwOffsetToAddress: address where the program is downloaded (e.g. for bin files or hex files that were loaded to another bank.) sThread: information for communication with the Thread (ThreadID, Handle..)
	Return	kSuccessful: OK kFailed: user abort kCommandDllError: command DLL not loaded or faulty kFileNotFound: no hex/bin file loaded kNoLibSelected: no FlashLib was selected yet kNetworkReceiveFailed: error during communication with the module kNetworkSendFailed: error during communication with the module

tReturnCode DmReadFlash (tPLReadFlash* pParamList_p, sThread* sReadFlashThread);		
	Function	The function reads a byte sequence from the Flash and stores it in the internal buffer.
	Parameter	tPLReadFlash: Structure with address and number of bytes to be read. sThread: information about communication with the Thread (ThreadID, Handle..)
	Return	kSuccessful: command was executed without error kFailed: user abort kCommandDllError: command DLL not loaded or faulty kTimeout: timeout of the network layer or error code received from SBC kNoAccess: no read access to the Flash kSoftwareProtected: Flash sector(s) is protected kNetworkReceiveFailed: command during communication with the module kNetworkSendFailed: error during communication with the module

tReturnCode DmSendPassword (tPLSendPassword* pParamList_p) ;		
	Function	This function sends a password to the SBC in order to activate the Upload/Update functionality.
	Parameter	tPLSendPassword: structure with password
	Return	kSuccessful: Upload/Update activated kInvalidPassword: password invalid kNetworkReceiveFailed: error during communication with the module kNetworkSendFaild: error during communication with the module

Information functions:

tReturnCode DmGetSectorStatus (tPLGetSectorStatus* pParamList_p , tRLGetSectorStatus* pRetList_p, sThread *sGetSectorStatusThread) ;		
	Function	The function returns the sector status. This function returns the status (Blank/NotBlank) for all sectors that are located between dwFirstSectorNumber and dwLastSEctorNumber. The DLL allocates memory for every sector that is returned in the structure tRLGetSectorStatus. The pointer tSectorStatus points to this memory. This memory must be released when it is no longer required by the application. This is done by calling the DLL function DmFreeMemory().
	Parameter	tPLGetSectorStatus: structure with start and end sector tRLGetSectorStatus: structure with status of each individual sector. SThread: information for communication with the Thread (ThreadID, Handle..)
	Return	kSuccessful: command was executed without error kFailed: user abort kCommandDllError: Command DLL was not loaded or was faulty kSectorInvalid: address is not a valid sector kNetworkReceiveFailed: error during communication with the module kNetworkSendFailed: error during communication with the module kNoLibSelected: no FlashLib was selected yet

tReturnCode DmGetSectorInfos (tRLGetFlashLibInfo *pParamList_p, tRLFlashInfo* pRetList_p) ;		
	Function	The function returns the sector structure from the active Flash
	Parameter	tRLGetFlashLibInfo: Structure with ManufacturerID, DeviceID, …(return value of FindFlashLib), or NULL. tRLFlashInfo: structure with sector information that is read from the Flash and module *.ini file. The Flash is specified by the tRLGetFLashLibInfo specification. If NULL is given for this parameter, then the DLL uses the information from the currently selected Flash. The DLL allocates memory for the information, which is returned in the structure tRLGetFLashLibInfo. The pointer tBLockInfo points to this memory. This memory must be released when it is no longer required by the application. This must be done by calling the DLL function DmFreeMemory(). Before this function can be executed a module must be selected with the DmLogin.
	Return	KSuccessful: command was executed without error KNoModulSelected: no connection could be established with a module KInvalidFileFormat: the required information could not be found in the *.ini file.

tReturnCode DmGetGeneralInfos (tRLGeneralInfo* pRetList_p) ;		
	Function	The function returns general information about the target module and the active Flash.
	Parameter	tRLGeneralInfo: structure with module information retrieved from the module's *.ini file. Before this function can be executed, a module must be selected with DmLogin.
	Return	kSuccessful: command was executed without error kNoModulSelected: no connection could be established with a module kInvalidFileFormat: the required information could not be found in the *.ini file.

tReturnCode DmGetProtectedAreas (tRLGetProtAreas* pRetList_p) ;		
	Function	The function returns all of the module’s protected Flash areas.
	Parameter	tRLGetProtAreas: structure with information about the protected areas of the Flash. The DLL allocates memory for the protected or NoAccess areas, which is returned in the structure tRLGetProtAreas. The pointer tAddrRange point to this memory. This memory must be released when it is no longer required by application. This must occur by calling the DLL function DmFreeMemory(). Before this function can be executed, a module must be selected with the DmLogin.
	Return	KSuccessful: command was executed without error KNoModulSelected: no connection to the module could be established KInvalidFileFormat: the required information could not be found in the *.ini file.

Thread:

tReturnCode DmStopCurrentThread () ;		
	Function	The function ends the currently running thread.
	Parameter	-
	Return	kSuccessful: command was executed without error kFailed: no thread available kCommandDllError: command DLL did not load or is faulty

tReturnCode DmGetCurrentThreadStatus () ;		
	Function	The function returns the status of the current thread.
	Parameter	-
	Return	KThreadRunning: thread is still running KCommandDllError: command DLL did not load or is faulty. Error status of the thread.

BYTE DmGetProgress()		
	Function	This function reports the progress of the running thread as a percentage
	Parameter	-
	Return	Progress

Memory:

tReturnCode DmFreeMemory(void* Buffer)		
	Function	The function releases the passed memory area. This memory area must be allocated by the DLL before passed to internal functions.
	Parameter	Buffer: pointer to the memory area
	Return	kSuccessful: Ok kFailed: memory area not valid kCommandDllError: command DLL did not load or is faulty

tReturnCode DmGetHexFileInfo(DWORD dwOffset, tRLFileInfo *pParamList_p)		
	Function	This function returns information about the loaded hex file (memory size, occupied sectors)
	Parameter	dwOffset: Offset of the hex file pParamList_p: structure with information
	Return	kSuccessful: Ok kNoModulSelected: no connection could be established with the module

tReturnCode DmCheckOldFlashTools()		
	Function	This function tests whether the old version of FlashTools (FlashTools98) or the current version is present in Flash Bank0 on a module with firmware. Using this function makes only sense when using a module with a pre-programmed flash.
	Parameter	-
	Return	kSuccessful: current version kCommandDllError: command DLL did not load or is faulty kNetworkReceiveFailed: error during communication with the module kNetworkSendFailed: error during communication with the module kOldFlashToolsVersionFound: old version found

1.2 Structure Definitions

When using the structures it is important to note that for technical reasons the structures with the note *#pragma pack (1)* were saved in a packed manner. Please use the header files included with the software for declaration of the structures. These contain the required pack instructions.

1.2.1 General Structures

General structures are mostly used to make it easier to write to the send buffer and read from the receive buffer.

```
typedef enum {
    k8Bit_16BitMode = 0x00,
    k8Bit = 0x01,
    k16Bit = 0x02
} tMemType;

typedef struct
  {
      char        szFlashName [ 32 ] ;
      DWORD       dwFlashSize ;
      DWORD       dwFlashID ;
      WORD        wNumOfBlocks ;
      tBlockInfo  aBlockInfo[wNumOfBlocks] ;
  } tRLFlashInfo ;
  typedef struct
  {
      DWORD    dwOffset ;
      DWORD    dwSectorSize ;
      WORD     wNumberOfSectors ;
   } tBlockInfo ;

  typedef struct
  {
      DWORD       dwSectorNumber ;
      tReturnCode ErrorCode ;
  } tEraseStatus ;
```

```
typedef struct
{
    DWORD  dwSectorNumber ;
    BYTE   bSectorStatus ;   // LSB: ProtectionStatus (nur auf PC-Seite)
                             //  MSB: BlankStatus
} tSectorStatus ;

typedef struct
{
    DWORD  dwStartAddress ;
    DWORD  dwEndAddress ;
} tAddrRange ;
```

1.2.2 Structures for Parameter Lists

```
typedef struct
  {
      BYTE  bEvenOdd ;
  } tPLInitFlashLib ;

  typedef struct
  {
      DWORD  dwStartAddress ;
      DWORD  dwEndAddress ;
      DWORD  dwSectorSize ;
  } tPLEraseSectors ;

  typedef struct
  {
      DWORD  dwStartAddress ;
      DWORD  dwEndAddress ;
      DWORD  dwBlockSize ;
  } tPLEraseBlocks ;

  typedef struct
  {
      BYTE  bFlashLibPtr ;              // 0 = FirstLib, 1 = NextLib
      DWORD dwFlashAddress;
  } tPLGetFlashLibInfo ;

  typedef struct
  {
      DWORD  dwStartAddress ;
      DWORD  dwEndAddress ;
      DWORD  dwSectorSize ;
  } tPLGetSectorStatus ;

typedef struct
  {
      DWORD  dwStartAddress ;
      WORD   wNumOfBytes ;
      BYTE   aData [ m_wNumOfBytes ] ;
  } tPLProgFlash ;
```

```
typedef struct
   {
       DWORD  dwStartAddress ;
       WORD   wNumOfBytes ;
   } tPLReadFlash ;

   typedef struct
   {
       WORD  wFlashLibID;
       WORD  wAdditionalID;
   } tPLSelectFlashLib ;

typedef struct
   {
        DWORD dwPwdLength;
       char *Password ;
   } tPLSendPassword ;

typedef struct
   {
        BYTE bStartUserApp;
       DWORD  dwLogStartAddr ;
   } tPLExitFlashTools ;
```

1.2.3 Structures for Return Values

```
  typedef struct
  {
      WORD          wNumOfEraseStatus ;
      tEraseStatus  aEraseStatus [ wNumOfEraseStatus ] ;
  } tRLEraseSectors ;

  typedef struct
  {
      WORD          wNumOfEraseStatus ;
      tEraseStatus  aEraseStatus [ wNumOfEraseStatus ] ;
  } tRLEraseBlocks ;

typedef struct
  {
      DWORD  dwNumOfBytesRead ;
      BYTE   aData [ wNumOfBytesRead ] ;
  } tRLReadFlash ;

typedef struct
  {
      WORD      wFlashLibID ;
      WORD      bManufacturerID ;
      WORD      bDeviceID ;
      WORD      wAdditionalID ;

  } tRLGetFlashLibInfo ;


typedef struct
  {
      char      szModuleName [ 32 ] ;
      char      szMicroController [ 32 ] ;
      tMemType  MemType ;
  } tRLGeneralInfo ;


typedef struct
  {
      WORD           wNumOfSectorsInBuffer ;
      tSectorStatus  aSectorStatus [ wNumOfSectorsInBuffer ] ;
```

```
   } tRLGetSectorStatus ;

typedef struct
   {
       WORD   wTxBufferSize ;
       BYTE   bMajorRelease ;
       BYTE   bMinorRelease ;
       BYTE   bPatchLevel ;
   } tRLLogin ;

typedef struct
   {
       WORD        m_wNumOfProtAreas ;
       WORD        m_wNumOfNonAccAreas ;
       tAddrRange  m_aProtectedAreas [ m_wNumOfProtAreas ] ;
       tAddrRange  m_aNonAccessAreas [ m_wNumOfNonAccAreas ] ;
   } tRLGetProtAreas ;


typedef struct {
    DWORD dwDataSize;
    DWORD dwNumberOfSectorsInBuffer;
    tPLEraseSectors *pSectorInfo;
} tRLFileInfo;
```

1.2.4 Structures for Threads

Functions that require more time on the PC, e.g. functions that wait for an answer from the SBC, are implemented as threads so that the status of the function can be queried or the function can be aborted. The DLL provides a function for stopping the thread (DmStopCurrentThread() and a function that returns the status of the thread.

A structure is given for each function that is implemented via a thread. This structure is filled out when the thread is generated and then returned to the application. The structure contains the handle and the ID of the thread, which can be used to access the thread directly.

```
typedef struct {
    UINT wID;
    HANDLE hHandle;
    DWORD dwReserved1;
    DWORD dwReserved2;
    WORD  wReserved;
} sThread;
```

1.2.5 Used enum Definitions

```
typedef enum {

   kSuccessful             = 0x00,
   kFailed                 = 0x01,
   kCommBufferTooSmall     = 0x02,
   kMissingInformation     = 0x03,
   kTimeOut                = 0x10,
   kSectorInvalid          = 0x20,
   kNotBlank               = 0x21,
   kBlank                  = 0x22,
   kUnknown                = 0x23,
   kAddressInvalid         = 0x30,
   kAddrNoAccess           = 0x31,
   kAddrProtected          = 0x40,
   kHardwareProtected      = 0x41,
   kSoftwareProtected      = 0x42,
   kFlashtoolsProtected    = 0x43,
   kPartialAccess          = 0x44,
   kNoAccess               = 0x45,
   kFullAccess             = 0x46,
   kLoadFileError          = 0x50,
   kLibNotFound            = 0x80,
   kNoLibSelected          = 0x81,
   kInvalidPassword        = 0x90,
   kToFewBytes             = 0xA0,
   kLoadBootFileError      = 0xB0,
   kBootLoaderNotActive    = 0xB1,
   kLoadFlashFileError     = 0xB2,
   kUnknownCommand         = 0xB3,
   kChecksumError          = 0xB4,
   kNoRamAtAddress         = 0xB5,
   kThreadRunning          = 0xC0,
   kFileNotFound           = 0xE0,
   kInvalidFileFormat      = 0xE1,
   kNoModulSelected        = 0xE2,
   kInvalidComPort         = 0xE4,
   kInitComPortError       = 0xE8,
   kCommandNotSupported    = 0xED,
   kUndefinedCommand       = 0xEE,
```

```
   kOldFlashToolsVersionFound = 0xEF,
   kNetworkSendFailed      = 0xF0,
   kNetworkReceiveFailed   = 0xF1,
   kNetworkDllNotFound     = 0xF2,
   kNetworkDllError        = 0xF3,
   kNetworkInitError       = 0xF4,
   kCommandDllNotFound     = 0xFA,
   kCommandDllError        = 0xFB,
   kDmDllNotFound          = 0xFC,
   kDmDllError             = 0xFD,
   kUnknownErrorCode       = 0xFF
} tReturnCode;
```

2 Implementation Hints

2.1 Parameters values

The parameter passing to the DLL is realised with the help of structures. The majority of the functions accept pointers to two structures as parameters. The first structure (starting with *tPL*) contains the parameters that are passed to the DLL. In the second structure (beginning with *rRL*) data from the DLL is returned to the program that performed the call.

A few structures contain an element that describes a pointer to a data area (e.g. the functions that work with sectors such as DmEraseSector, DmGetSectorStatus). Since the number of elements (structures with sectors) that can be returned is not always fixed or known at the time these functions are called, memory cannot be allocated in advance by the application. Therefore the memory is allocated by the function in the DLL. The application is responsible for releasing this memory if it is no longer needed. This memory must be released in the DLL with the function *DmFreeMemory*, once it is no longer required.
The following example is intended to further clarify this:

```
tPLEraseSectors sEraseSectorsIn;   // initialize parameters
tRLEraseSectors sEraseSectorsOut;
sThread sEraseSectorsThread;

sEraseSectorsIn.dwStartAddress = 0x0000;   // Startadress
sEraseSectorsIn.dwEndAddress   = 0x100000; // Endaddress
sEraseSectorsIn.dwSectorSize   = 0x10000;  // SectorSize
// call function in DLL
    if (m_pDmAccess->EraseSectors(&sEraseSectorsIn,&sEraseSectorsOut,
        &sEraseSectorsThread) != kSuccessful) {
            AfxMessageBox("Error erasing sectorsfile");
            return;
        }
```

The structure sEraseSectorsOut now contains in the element wNumOfEraseStatus, the number of sectors for which an error occurred.
The element aEraseStatus is a pointer to an array of type tEraseStatus.
Now these sectors can be read:

```
if (sEraseSectorsOut.wNumOfEraseStatus) {  // al least one sector errornous
    for (WORD wCurrentSectorStatus = 0; wCurrentSectorStatus <
         sEraseSectorsOut.wNumOfEraseStatus; wCurrentSectorStatus++) {
       UpdateSector(sEraseSectorsOut.aEraseStatus[wCurrentSectorStatus].ErrorCode);
    }
```

Now the memory can be released.

```
m_pDmAccess->DmFreeMemory(sEraseSectorsOut.aEraseStatus);
```

2.2 Threads

Functions that require more time were implemented in the DLL as a thread, this means that a call of the function in the DLL starts a thread there and then returns immediately.

The state of the thread can be queried with the function tReturnCode DmGetCurrentThreadStatus (). As long as the thread is running, this function will return *kThreadRunning*. Otherwise it will deliver a value with which the thread was ended.

In order to stop the thread from the application program the function DmStopCurrentThread () is called. This will end the current thread.
As an alternative the parameter *sThread** can be used as well. All function implemented as a thread have this parameter. This structure has a handle to a thread and a thread ID as elements. Both of these values are filled out by the DL and can be used by the application to access a thread directly (before using the thread structure directly please read the documentation for your development environment).

We recommend using the functions made available by the DLL.

2.2.1 Operation sequence

First the connection to the module must be established. The function

```
tReturnCode DmLogin (char *szIniFileName, tRLLogin pRetList_p, sThread *sLoginThread);
```

is used for this ;

The parameters for this function contains the path and file name for the *.ini file, which describes the connected module. These *.ini files are located in the folder TARGET*MODULNAME*\modul.ini.
If this function was called successfully, then a FlashLib must be selected on the microcontroller module. First the FlashLibs on the SBC module will be queried with the function

```
tReturnCode DmFindFlashLib ( tPLGetFlashLibInfo* pParamList_p,
                             tRLGetFlashLibInfo* pRetList_p) ;
```

First the element bFlashLibPtr must be set to 0 to access the first FlashLib. In addition, the start address of the Flash has to be given. The microcontroller responds by returning information about the FlashLib found in the structure tRLGetFlashLibInfo.
Now additional FlashLibs can be searched for with the function DmFindFlashLib() and bFlashLibPtr = 1, until the right one is found or until no more are present[1]. The desired FlasLib can be selected on the microcontroller module with the function

```
tReturnCode DmSelectFlashLib ( tPLSelectFlashLib* pParamList_p);
```

This concludes the initialisation and the module can be used.

To stop communication with the module FlashTools must be stopped on the SBC. The following function is used to do this.

```
tReturnCode DmExitFlashTools ( tPLExitFlashTools* pParamList_p ) ;
```

[1] At this time PHYTEC FlashTools only include a FlashLib for AMD and AMD-compatible Flash devices.

This resets the communication with the microcontroller module. If desired a user program can be started on the microcontroller too. To do this the parameter bStartUserApp must be set to 1 and the start address of the program must be given in the parameter dwLogStartAddr.