Short Description

Main

AddEventHandler	Installs a event handler for a specific event ID
BootReason	Returns the reason of the last power up
CallBIOS	Calls the firmware's BIOS routine
CallFirmware	Calls a firmware routine
CallFirmwareA	Calls a firmware routine
CallFirmwareA64	Calls a firmware routine
CallFirmwareT	Calls a firmware routine
CallTraceEnable	Enables or disables a call trace
CallTraceEnter	Marks the start of a subroutine
CallTraceExit	Marks the end of a subroutine
CallTraceComment	Adds a text string to the log
CallTraceExitResult	Marks the end of a subroutine and adds a text string to the output
CallTraceInit	Initializes a call trace
DelEventHandler	Removes a handler installed by AddEventHandler
EnqueueEvent	Puts an event into the event queue
FindInstructionSequence	Locates a pattern in the firmware
FlushCache	Flushes the data cache and invalidates the instruction cache
GetFWInfo	Returns details about the current firmware
GetHeapParameter	Returns some details about the heap
GetToppyString	Returns a text string to a SysID
IdentifyFirmware	Returns some info about a firmware (needs Firmware.dat)
InitTAPex	Initializes the library.
InitTAPexFailedMsg	Displays a message box with some device and firmware info. Should be called when InitTAPex returns FALSE.
intLock	Locks all CPU interrupts
intUnlock	Unlocks all CPU interrupts
isMasterpiece	Returns whether the Toppy is a Masterpiece/6000 style device
iso639_1	Returns the ISO639-1 language ID
isValidRAMPtr	Checks if a pointer is location within RAM (0x80000000-0x83ffffff, 0xa0000000-0xa3ffffff)
LE16	Swaps the endian of a 16-bit word
LE32	Swaps the endian of a 32-bit dword
LoadFirmwareDat	Loads the file Firmware.dat and returns pointer to the structure
Log	Appends a text line to a log file and optionally writes it to the console
Now	Returns the current date and time in the Topfield format
PatchInstructionSequence	Patches a sequence of instructions in RAM
SendToFP	Sends data to the front panel processor
SetCrashBehaviour	Defines how the firmware reacts upon the next crash. (can modify the firmware only once!)
SuppressedAutoStart	Returns whether the automatic start of TAPs has been suppressed with the 0 key

Audio / Video / OSD

CaptureScreen	Captures the screen (with or without OSD)
DrawOSDLine	Draws a line
EndMessageWin	Removes a asynchronous message window
FindDBTrack	Returns the track index of the dolby channel
FreeOSDRegion	Deletes a OSD region without erasing the onscreen graphic
GetAudioTrackPID	Returns the PID of one of the 64 possible audio tracks
GetCurrentEvent	Returns information about the EPG event of the current channel which is marked as running
GetFrameBufferPixel	Returns the colour of pixel in the video frame buffer
GetFrameSize	Returns the size of the current video frames
GetOSDMapAddress	Returns a pointer to the OSD map in memory
GetOSDRegionHeight	Returns the height of a OSD region
GetOSDRegionWidth	Returns the width of a OSD region

GetPinStatus	Returns the status of the PIN entry window [doesn't work right now]
GetPIPPosition	Returns the current location of the PIP
GetSysOsdControl	Has a system OSD infobox been deactivated?
InteractiveGetStatus	Returns whether interactive is enabled (TF5800 only)
InteractiveSetStatus	Activates or deactivates the interactive mode (TF5800 only)
isAnyOSDVisible	Checks if a specified screen area contains an OSD graphic
isOSDRegionAlive	Checks if a OSD region is still active
MHEG_Status	Returns the current MHEG status
OSDCopy	Copies a OSD buffer x times
SaveBitmap	Saves a captured screen into a file
SetRemoteMode	Enables or disables a specific remote mode
ShowMessageWin	Displays a message window on the screen
ShowMessageWindow	Extended version of ShowMessageWin
SoundSinus	Beep
SubtitleGetStatus	Returns if subtitle is enabled
SubtitleSetStatus	En-/disables the subtitle
TAP_Osd_PutFreeColorGd	Copies a Gd object into a OSD region with a specific colors
TunerGet	Returns the tuner number of the main or PIP service
TunerSet	Sets the tuner number of the main service
YUVB	Returns the blue component of a YUV color
YUVG	Returns the green component of a YUV color
YUVR	Returns the red component of a YUV color

DIALOG

DialogEvent	Forwards events to a dialog window. Used events (e.g. key presses for scrolling) will be set to 0.
DialogMsgBoxButtonAdd	Adds a user defined button to a messagebox (5 max)
DialogMsgBoxExit	Terminates a message box
DialogMsgBoxInit	Initializes the structures of a message box
DialogMsgBoxShow	Displays a message box
DialogMsgBoxShowInfo	Displays a message box without any buttons
DialogMsgBoxShowOK	Displays a message box with a OK button
DialogMsgBoxShowOKCancel	Displays a message box with the buttons OK and Cancel
DialogMsgBoxShowYesNo	Displays a message box with the buttons Yes and No
DialogMsgBoxShowYesNoCancel	Displays a message box with the buttons Yes, No and Cancel
DialogMsgBoxTitleSet	Modifies the title of the message box
DialogProfileChange	Assigns a new user profile
DialogProfileLoad	Loads a user profile
DialogProfileLoadDefault	Loads the default user profile (/ProgramFiles/DialogProfile.ini)
DialogProfileSave	Saves a user profile
DialogProfileSaveDefault	Makes the current profile the default profile
DialogProfileScrollBehaviourChange	Changes the scroll behaviour of the current profile
DialogProgressBarExit	Terminates a progress bar window
DialogProgressBarInit	Initializes the structures of a progress bar
DialogProgressBarSet	Sets the progress bar to a specific value
DialogProgressBarShow	Displays a progress bar
DialogProgressBarTitleSet	Modifies the title of a progress bar window
DialogWindowAlpha	Temporarily changes the alpha blending of a dialog window
DialogWindowChange	Assigns a new window profile
DialogWindowCursorChange	Turns the cursor on or off
DialogWindowCursorSet	Sets the cursor to a specific line
DialogWindowExit	Terminates a dialog window
DialogWindowHide	Hides a dialog window
DialogWindowInfoAddIcon	Adds an icon to the info area
DialogWindowInfoAddS	Adds a text string to the info area
DialogWindowInfoDeleteAll	Erases the info area
DialogWindowInit	Initializes the dialog structures
DialogWindowItemAdd	Adds a new item to the dialog window

DialogWindowItemAddSeparator	Adds a blank line to the dialog window
DialogWindowItemChangeFlags	Modifies the flags of a dialog item (selectable, enabled)
DialogWindowItemChangeIcon	Modifies the icon of a dialog item
DialogWindowItemChangeParameter	Changes the parameter text of a item
DialogWindowItemChangeValue	Changes the value of a item
DialogWindowItemDelete	Deletes a single item [not yet implemented]
DialogWindowItemDeleteAll	Deletes all items
DialogWindowRefresh	Redraws a dialog window
DialogWindowReInit	Modifies the window geometry
DialogWindowScrollDown	Scroll down one line
DialogWindowScrollDownPage	Scroll down one page
DialogWindowScrollUp	Scroll up one line
DialogWindowScrollUpPage	Scroll up one page
DialogWindowShow	Displays a dialog window
DialogWindowTabulatorSet	Sets the tabulator position of the parameter or value column
DialogWindowTitleChange	Modifies the window title
DialogWindowTypeChange	Changes the window style
InfoTestGrid	Draws 10x10 pixel grid into the info area (for TAP development)

COMPRESSION

CompressBlock	Compresses a data block
CompressedTFDSize	Returns the size of a compressed TFD structure
CompressTFD	Compresses a data block into a TFD structure
CRC16	Calculates the 16-bit CRC of a memory area
TFDSize	Returns te size of a TFD structure
UncompressBlock	Decompresses a data block
UncompressedFirmwareSize	Returns the uncompressed size of a packed firmware
UncompressedLoaderSize	Returns the uncompressed size of a packed loader
UncompressedTFDSize	Returns the uncompressed size of a packed TFD file
UncompressFirmware	Decompresses a packed firmware
UncompressLoader	Decompresses a packed loader
UncompressTFD	Decompresses a packed TFD file

FILESELECTOR

FileSelector	Opens the FileSelector
FileSelectorKey	Checks if a key is used by FileSelector

FLASH

AddSec	Adds or subtracts the number of seconds from a time value.
AddTime	Adds or subtracts the number of minutes from a time value.
DATE	Converts a date / hour / minute into the Topfield internal date format
FlashAddFavourite	Adds a new favourite
FlashDeleteFavourites	Deletes all favourites
FlashFindEndOfServiceNameTableAddress	Returns a pointer to the end of the service name table
FlashFindEndOfServiceTableAddress	Returns a pointer to the end of the service table
FlashFindServiceAddress	Finds and returns a pointer to a specific service
FlashFindTransponderIndex	Finds and returns a pointer to a specific transponder
FlashGetBlockStartAddress	Returns a pointer to the start of a block
FlashGetChannelNumber	Returns the frequency and channel number of a service (DVB-t only)
FlashGetSatelliteByIndex	Returns a pointer to a specific satellite (DVB-s only)
FlashGetServiceByIndex	Returns a pointer to a specific service
FlashGetServiceByName	Returns a pointer to a specific service referenced by it’s name
FlashGetTransponderCByIndex	Returns the transponder info (DVB-c version)
FlashGetTransponderSByIndex	Returns the transponder info (DVB-s version)
FlashGetTransponderTByIndex	Returns the transponder info (DVB-t version)
FlashGetType	Returns the system of the Toppy (DVB-s, -t, or -c)
FlashProgram	Writes the shadow of the Flash into the Flash chip

FlashReindexFavourites	Reindexes the favourites if a service has been added or deleted
FlashReindexTimers	Reindexes the timers if a service has been added or deleted
FlashRemoveCASServices	Deletes all services marked with a $ sign
FlashRemoveServiceByIndex	Deletes a service addressed by an index
FlashRemoveServiceByIndexString	Deletes a service addressed by an index string
FlashRemoveServiceByLCN	Deletes one or more services addressed by a LCN (DVB-c and DVB-t)
FlashRemoveServiceByName	Deletes a service addressed by its name
FlashRemoveServiceByPartOfName	Deletes a service addressed by a part of its name
FlashRemoveServiceByUHF	Deletes one or more services addressed by a VHF/UHF channel number
FlashServiceAddressToServiceIndex	Converts a pointer to an service index
FlashWrite	Writes data directly into the flash memory (Use only if you know what you're doing!)
GetEEPROMAddress	Returns the address of the EEPROM shadow in memory
GetEEPROMPin	Returns the user defined PIN code
HOUR	Extracts the hour from the Topfield internal date format
MINUTE	Extracts the minutes from the Topfield internal date format
MJD	Extracts the date from the Topfield internal date format
TimeDiff	Calculates the difference in minutes between two time values

HARDDISK

HDD_AAM_Disable	Disables Automatic Acoustic Management
HDD_AAM_Enable	Enables Automatic Acoustic Management
HDD_APM_Disable	Disables Advanced Power Management
HDD_APM_Enable	Enables Advanced Power Management
HDD_BigFile_Read	Reads a sector from a file and is not restricted to the 2GB limit
HDD_BigFile_Size	Returns the size of a file in sectors.
HDD_BigFile_Write	Writes a sector from a file and is not restricted to the 2GB limit. This function can not increase the size of a file!
HDD_ChangeDir	Changes the current directory and accepts a full path
HDD_FappendOpen	Opens a file to append text
HDD_FappendWrite	Appends text to the end of a text file
HDD_FreeSize	Returns the free disk space in MB. This function doesn’t access the disk thus doesn’t wake it up from standby.
HDD_GetClusterSize	Returns the size of one cluster in sectors
HDD_GetFirmwareDirCluster	Returns the cluster number of the firmware’s current directory
HDD_GetHddID	Returns the type, serial number and firmware revision of the HDD
HDD_GetHddInfo	Returns some Data about the built in HDD
HDD_IdentifyDevice	Returns the IdentifyDevice information block
HDD_LiveFS_GetChainLength	Returns the number of cluster of a FAT chain
HDD_LiveFS_GetFAT1Address	Returns the address of the FAT in memory
HDD_LiveFS_GetFAT2Address	Returns the address of the FAT copy in memory
HDD_LiveFS_GetFirstCluster	Returns the first cluster of a FAT chain
HDD_LiveFS_GetLastCluster	Returns the last cluster of a FAT chain
HDD_LiveFS_GetNextCluster	Returns the next cluster of a FAT chain
HDD_LiveFS_GetPreviousCluster	Returns the previous cluster of a FAT chain (very slow!)
HDD_LiveFS_GetRootDirAddress	Returns the address of the root directory in memory
HDD_LiveFS_GetSuperBlockAddress	Returns the address of the superblock in memory
HDD_Move	Moves a file from one to another directory
HDD_ReadClusterDMA	Reads a cluster from the HDD
HDD_ReadSector	Reads a sector in PIO mode. At this time, no other processes should access the disk (Play, Rec, TS)
HDD_ReadSectorDMA	Reads a sector from the HDD
HDD_SetCryptFlag	Sets the crypt flag of an open file
HDD_SetFileDateTime	Sets the date and time of an open file
HDD_SetSkipFlag	Sets the skip flag of an open file
HDD_SetStandbyTimer	Sets the standby timeout of the HDD

HDD_Smart_DisableAttributeAutoSave	Disables SMART attribute auto-save
HDD_Smart_DisableOperations	Disables SMART
HDD_Smart_EnableAttributeAutoSave	Enables SMART attribute auto-save
HDD_Smart_EnableOperations	Enables SMART
HDD_Smart_ExecuteOfflineImmediate	Starts an offline check
HDD_Smart_ReadData	Reads the SMART information. At this time, no other processes should access the disk (Play, Rec, TS)
HDD_Smart_ReadThresholdData	Reads the SMART threshold information. At this time, no other processes should access the disk (Play, Rec, TS)
HDD_Smart_ReturnStatus	Returns the SMART Status (good, failed)
HDD_Stop	Shuts down the disk
HDD_TouchFile	Sets the file date to the current time
HDD_TranslateDirCluster	Translates a cluster number into a directory string
HDD_TruncateFile	Truncates a file to a specific size [doesn't support cluster boundaries]
HDD_Write	Buffered Write
HDD_WriteClusterDMA	Writes a cluster to the HDD
HDD_WriteSectorDMA	Writes a sector to the HDD

HOOKS

HookEnable	Enables/disables a hook
HookExit	Disables all hooks
HookIsEnabled	Returns the enable/disable status of a hook
HookMIPS_Clear	Removes a firmware hook
HookMIPS_Set	Creates a firmware hook (these are different to the Hook* functions)
HookSet	Sets a Firmware hook

IIC BUS

ReadEEPROM	Reads the contents of the EEPROM
ReadIICRegister	Reads a register of one of the IIC chips (EEPROM, A/V-matrix, tuner).
WriteIICRegister	Writes a register of an IIC chip

IMEM

IMEM_Alloc	Allocates memory
IMEM_Compact	Reorganizes the internal tables
IMEM_Free	Frees allocated memory
IMEM_GetInfo	Returns information about the available memory
IMEM_Init	Initializes the memory manager
IMEM_isInitialized	Reports if the memory manager has been initialized
IMEM_Kill	Closes the memory manager and frees the allocated memory

INI FILES

INICloseFile	Closes an INI file
INIGetARGB	Returns a ARGB color from a specific key
INIGetRGB	Returns a RGB color from a specific key
INIGetRGB8	Returns a RGB color from a specific key, assuming that the INI uses 8 bit values
INIGetHexByte	Returns a byte from a hex string from a specific key
INIGetHexDWord	Returns a dword from a hex string from a specific key
INIGetHexWord	Returns a word from a hex string from a specific key
INIGetInt	Returns an integer from a specific key
INIGetString	Returns a string from a specific key
INIKeyExists	Checks if a specific key exists
INIKillKey	Removes a key from a INI file
INILocateFile	Returns the location of a file (current dir, /PF, /PF/Settings, /PF/Settings/<AppName>)
INIOpenFile	Opens an INI file
INISaveFile	Saves an INI file
INISetARGB	Writes a ARGB color to a specific key

INISetRGB	Writes a RGB color to a specific key
INISetRGB8	Writes a RGB color to a specific key, expanding the colors to 8 bit values
INISetComments	[Comments are not yet implemented]
INISetHexByte	Writes a byte to a specific key
INISetHexDWord	Writes a dword to a specific key
INISetHexWord	Writes a word to a specific key
INISetInt	Writes an integer to a specific key
INISetString	Writes a string to a specific key
LangGetString	Returns a language-dependent string
LangLoadStrings	Loads a language-dependent text ini
LangUnloadStrings	Releases the memory for the language-dependet strings

Masterpiece / 6000 VFD

MPDisplayClearDisplay	Clears the display
MPDisplayClearSegments	Clears single segments
MPDisplayDisplayLongString	Displays a string in the 8 digit display
MPDisplayDisplayShortString	Displays a string in the 4 digit display
MPDisplayGetDisplayByte	Returns a byte out of the 48 byte display buffer
MPDisplayGetDisplayMask	Returns a byte out of the 48 byte mask buffer
MPDisplayInstallMPDisplayFwHook	Activates the MP-display hook
MPDisplaySetAmFlag	Sets the AM flag
MPDisplaySetColonFlag	Sets the colon in the 4 digit display
MPDisplaySetDisplayByte	Sets a byte in the 48 byte display buffer
MPDisplaySetDisplayMask	Sets a byte in the 48 byte mask buffer
MPDisplaySetDisplayMemory	Copies a 48 byte buffer into the display buffer
MPDisplaySetDisplayMode	Selects the different display modes
MPDisplaySetPmFlag	Sets the PM flag
MPDisplaySetSegments	Sets single segments
MPDisplayToggleSegments	Toggles single segments
MPDisplayUninstallMPDisplayFwHook	Deactivates the MP-display hook
MPDisplayUpdateDisplay	Writes the VFD buffer to the display

Patches

PatchLoadModule	Loads a TFP file
PatchLoadModuleGP	Loads the specific patch for a particular $gp and all generic patches from a TFP file. It is therefore more resource friendly than PatchLoadModule.
PatchUnloadModule	Removes a TFP file from memory
PatchFindType	Scans a f/w if a patch is installable
PatchApply	Installs or removes a patch
PatchInstallID	Registers a PatchID
PatchIsInstalled	Returns the info if a specific PatchID is registered
PatchGetInstalled	Returns a string with all PatchIDs
PatchRemoveID	Removes a PatchID
PatchCleanList	Copies all PatchIDs into a temp. list and removes them from the firmware
PatchReinstallList	Copies the PatchIDs from the temp. list back into the f/w

REC Streams

HDD_DecodeRECHeader	Identifies the type of REC header and decodes it
HDD_EncodeRECHeader	Creates a REC header
HDD_FindPCR	Returns the first PCR from a REC stream buffer
HDD_FindPMT	Locates a PMT in a REC stream buffer and updates the REC header structure.
HDD_isAnyRecording	Is the Toppy recording anything right now?
HDD_isCryptedStream	Checks if a buffer contains crypted TS packets
HDD_isRecording	Is the Toppy recording on one slot right now?
HDD_MakeNewRECName	Adds a sequence number or 2 random characters to a REC file name
HDD_PausePlayback	Pausiert eine Aufnahme und setzt sie wieder fort
HDD_PlaySlotGetAddress	Returns the address of the tPlaySlot structure in the memory
HDD_RecalcPlaytime	Calculates the playtime of a REC file
HDD_RECSlotGetAddress	Returns the address of the tRECSlot structure in the memory
HDD_RECSlotIsPaused	Returns the info if a recording is pausing
HDD_RECSlotPause	This function can pause a recording
HDD_RECSlotSetDuration	Changes the duration of a active recording

Shutdown

Reboot	Reboots the Toppy
Shutdown	Stops a task (Rec, Play, Video or Audio) or shuts the Toppy down

Strings

ExtractLine	Returns a line from a text block
GetLine	Returns a line from a text block (improved version)
LowerCase	Converts all letters into lower case
MakeValidFileName	Removes invalid characters from a string
ParseLine	Searches for a substring in a string and divides the string at that position
RTrim	Removes spaces from the end of a string
SeparatePathComponents	Separates a full path into its components
StrEndsWith	Checks if a string ends with a specific pattern
TimeFormat	Creates a String from a date and time
UpperCase	Converts all letters into upper case
ValidFileName	Removes invalid characters from a file name

TAPs

HDD_TAP_Callback	Calls a function in an other TAP
HDD_TAP_Disable	Disables a TAP (the target won’t receive any events)
HDD_TAP_DisableAll	Disables all TAPs except the caller
HDD_TAP_GetCurrentDir	Returns the current directory path of the TAP
HDD_TAP_GetCurrentDirCluster	Returns the HDD cluster number of the current directory
HDD_TAP_GetIDByFileName	Returns the ID from a TAP-File
HDD_TAP_GetIDByIndex	Returns the ID from a TAP table index
HDD_TAP_GetIndexByID	Returns the TAP table index of a specific TAPID
HDD_TAP_GetInfo	Returns a lot of info about a loaded TAP
HDD_TAP_GetStartParameter	Returns a pointer to the parameter block in the server TAP
HDD_TAP_isAnyRunning	Checks if any TAP beside the caller is running
HDD_TAP_isBatchMode	Checks if the calling TAP has been launched in batch mode
HDD_TAP_isDisabled	Has a TAP being disabled? This also includes TAPs disabled by the TSRCommander.
HDD_TAP_isDisabledAll	Have all TAPs being disabled?
HDD_TAP_isRunning	Checks if a specific TAP is running
HDD_TAP_PopDir	Pops one entry from the dir. stack and changes to that dir
HDD_TAP_PushDir	Pushes the current TAP directory onto a stack
HDD_TAP_SendEvent	Sends an event to all TAPs. There are no restrictions to event, param1 and param2. The event is not passed to the firmware.

HDD_TAP_SetCurrentDirCluster	Sets the current directory
HDD_TAP_Start	Launches a TAP.
HDD_TAP_StartedByTAP	Has this TAP being started by an other TAP?
HDD_TAP_Terminate	Terminates a TAP [untested]

TAPCOM

TAPCOM_CloseChannel	Client closes the communication channel
TAPCOM_Finish	Server has finished processing the request
TAPCOM_GetChannel	Server gets the details about the clients request
TAPCOM_GetReturnValue	Tells the client the return value of the server
TAPCOM_GetStatus	Client checks the current state of the channel
TAPCOM_LastAlive	When was the server's last StillAlive response? (asynchronous communication only)
TAPCOM_OpenChannel	Client opens a communication channel to a server
TAPCOM_Reject	Server can not execute the request (right now)
TAPCOM_StillAlive	Server reports that it is still busy with the request

TAPAPIFIX

InitTAPAPIFix	Initializes this part of the library. See below for details.

TAPAPIFIX

The following bugs have been intercepted by tapapifix:

TAP_Hdd_GetPlayInfo	Bufferoverrun
TAP_Hdd_GetRecInfo	Bufferoverrun
TAP_Hdd_Fseek	Wrong SEEK_END-position with files, which size is a multiple of 512
TAP_Hdd_Flen	If the files size is a multiple of 512, Flen reports 512 bytes too little
TAP_Hdd_ChangeDir	Different return value for firmware versions prior to V5.12.0
TAP_Channel_SetAudioTrack	Jumping in a replay will change back to the default audio track
TAP_Hdd_Delete	Deleting a file with a long name and a dot in the first part of the name, may freeze the Toppy
TAP_Hdd_Fwrite	Doesn't write junk to the end of a file
TAP_Hdd_StopTs	Doesn't crash in normal mode

INCLUDING THE FIREBIRDLIB

* Add a #include "libFirebird.h"   into your .c-file
* Modify the Build.bat and add the FireBird-archive to the linker command e.g.:

```
  mips-ld --cref -o filer.elf -T ..\TAP.LD filer.o filerdb.o -l FireBird -l tap -l c -l gcc -l FireBird -Map filer.map
```

The FireBird-library needs to be the first –l option. If you're using the original Topfield compiler, the library needs to be linked again as the last library (see above).

HOWTO

FIRMWARE HOOKS

A hook redirects the execution in the firmware to a TAP. At this state, it is possible to analyze and modify CPU registers or memory locations. But it is not possible to hook all addresses. Some hooks are quite stable, others let the Toppy crash immediately.

The first step is to define a hook handler:

```
void HookHandler (dword HookIndex, tCPURegs *CPURegs)
```

Afterwards a hook can be installed:

```
dword HookIndex = HookSet (FirmwareEntryPoint, (dword *) HookHandler);
HookEnable (HookIndex, TRUE);
```

It is possible to set more than one hook. HookSet returns an index, which can be used the enable or disable a hook and which is also reported in the HookIndex parameter of the HookHandler. So it is possible to distinguish the different hooks in a single hook handler. And don't forget to deactivate the hooks before terminating a TAP!

```
HookExit();
```

TAPCOM
Launching a TAP

HDD_TAP_Start launches a TAP and allows passing parameters and a batch flag to the launched TAP. The batch flag might be used to perform some tasks without any user invention. 3PG, for example, can be launched in batch mode. In this mode, 3PG performs a full scan and terminates itself after it has finished the scan. The parameters and the batch flag is only valid within TAP_Main!

-- the client --

```
typedef struct
{
    char       Dir [80];
    char       FileName [80];
} tParameters;

tParameters    Parameters;
dword          TAPID;

//The client launches an editor and passes the directory and name of the file to edit
//Batch mode is not used.
TAP_SPrint (Parameters.Dir, "/ProgramFiles");
TAP_SPrint (Parameters.FileName, "3PG.ini");
HDD_TAP_Start ("editor.tap", FALSE, &Parameters, &TAPID);

//Wait until the editor has been terminated
do
{
  TAP_SystemProc();
} while (HDD_TAP_isRunning (TAPID_Editor));
```

-- the server --

```
typedef struct
{
    char       Dir [80];
    char       FileName [80];
} tParameters;

tParameters    *Parameters;

int TAP_Main(void)
{
  //The editor doesn't use batch mode but this is how it works
  if (HDD_TAP_isBatchMode())
  {
    ...
  }

  //Are there any paramters?
  Parameters = HDD_TAP_GetStartParameter();
  if (Parameters != NULL)
  {
    //Save them as they will only be valid within TAP_Main
  }
  ...
}
```

SYNCHRONOUS TAP TO TAP COMMUNICATION

This means that the client immediately receives a result and there won't be any additional action from the server. In contrast there are also the asynchronous communication (execution of a command lasts some time) and broadcasts (an information is sent to several servers).

The simple demo (see .Demo directory) consists of 1 server and 2 clients. The server offers a service which adds two integer numbers. To make it a little bit harder, the client must reserve the server before requesting the service. This reservation has nothing to do with TAPCOM, but is a feature of the demo server (e.g. like a client may reserve the front panel display so only one client has access at the same time). The demo uses the following remote keys:

1 = Client 1 reserves server
2 = Client 1 lets the server do the calculation
3 = Client 1 releases the server

The keys 4 to 6 do the same with client 2. UHF exits the server and both clients. All output is sent to the serial console.

The Client

The order of events is always the same:

- Open a communication channel and inform the server about what he should do
- Check the status of the channel and the return value (if necessary)
- Close the communication channel

//Calls a server and sends the command ReserveServer. No parameters are passed.
Channel = ***TAPCOM_OpenChannel*** (TAPCOM_App_DemoServer, TAPCOM_DemoServer_ReserveServer, 0, NULL);

//Check the channel status and act accordingly.
Status = ***TAPCOM_GetStatus*** (Channel);

//Close the channel
TAPCOM_CloseChannel (Channel);

The client may receive the following status codes:

TAPCOM_Status_SERVER_NOT_AVAILABLE: the server TAP hasn't been launched. It may be launched by the client.

TAPCOM_Status_REJECTED: the command is unknown or can't be executed right now (e.g. the server has been reserved by a different client).

TAPCOM_Status_FINISHED: the command has been finished. If successful or not can be checked via ***TAPCOM_GetReturnValue***.

TAPCOM_Status_VERSIONMISMATCH: client and server use an incompatible version of the TAPCOM library. The server didn't receive that command.

There are 2 more status codes but aren't used in a synchronous communication:

TAPCOM_Status_OPEN: the server has been loaded, but didn't react to our command (bad with sync. comm., normal for broadcasts).

TAPCOM_Status_ACKNOWLEDGED: the server has received the command but the execution takes time (used with the async. comm.)

Parameters and data are optional and the data structure is defined by the server. The client reserves the necessary memory and the server accesses it via a pointer. Therefore a client must not free that memory as long as the channel status is ACKNOWLEDGED or OPEN. In our sample, the client initializes the first two variables of the structure and receives the result from the server in the variable Result. The following lines do not check if the server has done the calculation (see the demo for more details).

```
typedef struct
{
  dword                 Number1;
  dword                 Number2;
  dword                 Result;
} tTAPCOM_DemoServer_Parameter;      //defined in TAPCOM_DemoServer.h

tTAPCOM_DemoServer_Parameter Parameter;

Parameter.Number1 = 1;
Parameter.Number2 = 2;

Channel = TAPCOM_OpenChannel (TAPCOM_App_DemoServer, TAPCOM_DemoServer_ExecAdd,
TAPCOM_DemoServer_ParameterVersion, (void*) &Parameter);

TAP_Print ("Sum = %d\n", Parameter.Result);
```

The Server

The implementation of a server is similar to a client. The server receives the request via an event (EVT_TAPCOM). This is not a real event, like EVT_KEY, and therefore the firmware doesn't recognize it. Afterwards the server picks up the details with ***TAPCOM_GetChannel***. If the return value of this function is 0, then the event wasn't meant for that particular server. Otherwise the server should perform the requested operation and close the channel with ***TAPCOM_Finish***. The return value transmitted with this function can be checked by the client with **TAPCOM_GetReturnValue**. If the server is not able to execute the function, it has to inform the client with a ***TAPCOM_Reject***. This concludes the servers TAPCOM commands. Coding a server might be a little bit more difficult as he has to do all the plausibility checks. But this doesn't have to do with the TAPCOM lib. The simplest case would be a ***TAPCOM_GetChannel*** followed by a ***TAPCOM_Finish***.

Our sample server performs the following tasks:

- Execution of the 3 self defined commands „ReserveServer", „ExecAdd" and „ReleaseServer".
- Rejection of requests from clients which didn't reserve the server.
- Checks if the client, who has reserved the server, is still running. If not, the server terminates the reservation.

See the .Demo directory for some examples.

Asynchronous Communication

The asynchronous communication is used whenever the execution of an operation may take longer.

Client

Comparison with the sync. comm:

- The server immediately returns the status *TAPCOM_Status_ACKNOWLEDGED* or *TAPCOM_Status_REJECTED*
- The client needs to call ***TAPCOM_GetStatus*** to recognize the end of the command execution
- The client may call ***TAPCOM_LastAlive***. This function returns the tick count of the servers last "I'm still busy with your command".

Server

Comparison with the sync. comm:

- The server needs to return *TAPCOM_Status_ACKNOWLEDGED* or *TAPCOM_Status_REJECTED* immediately
- The server needs to call ***TAPCOM_StillAlive*** continuously to notify the client that it is still busy with its request

BROADCAST

A broadcast sends a message to all TAPs (TargetID at OpenChannel = TAPCOM_App_BROADCAST). A parameter block may be passed to the servers and they may return data and a result code. In this case, the last server returning data wins.

Comparison with the sync. comm:

- A broadcast is always synchronous
- A server may not answer with *TAPCOM_Status_ACKNOWLEDGED* or *TAPCOM_Status_REJECTED*
- After control has been passed back to the client, the channel status is *TAPCOM_Status_Open*