KURZBESCHREIBUNG

MAIN

AddEventHandler	Installiert einen Event-Handler für eine bestimmte Event ID
BootReason	Liefert den Auslöser der letzten Einschaltung
CallBIOS	Ruft eine BIOS-Funktion auf
CallFirmware	Ruft eine Routine in der Firmware auf
CallFirmwareA	Ruft eine Routine in der Firmware auf
CallFirmwareA64	Ruft eine Routine in der Firmware auf
CallFirmwareT	Ruft eine Routine in der Firmware auf
CallTraceEnable	Aktiviert oder deaktiviert den Call Trace
CallTraceEnter	Markiert den Beginn eines Unterprogramms
CallTraceExit	Markiert das Ende eines Unterprogramms
CallTraceComment	Fügt einen Text zum Log hinzu
CallTraceExitResult	Markiert das Ende eines Unterprogramms und fügt einen Textstring zum Log hinzu
CallTraceInit	Initialisiert einen Call Trace
DelEventHandler	Entfernt einen von AddEventHandler installierten Handler
EnqueueEvent	Fügt einen Event in die Event-Queue ein
FindInstructionSequence	Sucht nach einem Muster in der Firmware
FlushCache	Entleert den Daten- und Instructioncache
GetFWInfo	Liefert Details über die aktuell installierte Firmware
GetHeapParameter	Liefert Details über den Heap-Speicher
GetToppyString	Liefert einen Text zu einer SysID
IdentifyFirmware	Liefert Inforationen über eine Firmware (benötigt Firmware.dat)
InitTAPex	Initialisiert die Bibliothek.
InitTAPexFailedMsg	Zeigt eine Messagebox mit Geräte- und Firmwareinfos. Sollte aufgerufen werden, wenn InitTAPex FALSE retour liefert.
intLock	Sperrt alle CPU-Interrupts
intUnlock	Entsperrt die CPU-Interrupts
isMasterpiece	Meldet, ob es sich um einen Masterpiece / TF6000 handelt
iso639_1	Liefert den ISO639-1 Sprachkennung
isValidRAMPtr	Prüft, ob ein Pointer im RAM liegt (0x80000000-0x83ffffff, 0xa0000000-0xa3ffffff)
LE16	Wechselt eine 16-Bit-Zahl zwischen Big- und Little Endian
LE32	Wechselt eine 32-Bit-Zahl zwischen Big- und Little Endian
LoadFirmwareDat	Lädt die Datei Firmware.dat und liefert Pointer auf die Strukturen
Log	Fügt einen Text an eine Log-Datei an und kann diesen auch auf der Konsole ausgeben.
Now	Gibt das aktuelle Datum und die Uhrzeit im Topfield-Format retour
PatchInstructionSequence	Patcht eine Sequenz im RAM
SendToFP	Sendet Daten an das Front Panel
SetCrashBehaviour	Bestimmt, wie sich die Firmware beim nächsten Crash verhält. (kann die F/W nur ein Mal modifizieren!)
SuppressedAutoStart	Wurde der Topf mittels 0-Taste ohne TAPs gestartet?

AUDIO / VIDEO / OSD

CaptureScreen	Captured den Bildschirminhalt (mit oder ohne OSD)
DrawOSDLine	Zeichnet eine Linie
EndMessageWin	Löscht das Meldungsfenster
FindDBTrack	Liefert den Track-Index des Dolby-Kanals
FreeOSDRegion	Löscht eine OSD-Region, ohne die Bildschirmgrafik zu löschen
GetAudioTrackPID	Liefert die PID von einem der 64 möglichen Audio-Tracks
GetCurrentEvent	Liefert den EPG-Event des aktuellen Kanals, welcher als “Running” markiert ist.
GetFrameBufferPixel	Liefert die Farbe eines Punkte im Video-Framebuffer
GetFrameSize	Liefert die Frame-Grösse des aktuellen Video-Streams
GetOSDMapAddress	Liefert einen Pointer auf die OSD-Map im Speicher
GetOSDRegionHeight	Liefert die Höhe einer OSD-Region
GetOSDRegionWidth	Liefert die Breite einer OSD-Region
GetPinStatus	Liefert den Status des PIN-Eingabefensters [funktioniert noch nicht]
GetPIPPosition	Liefert die Bildschirmkoordinaten des PIP-Fensters

GetSysOsdControl	Ist eine System-OSD-Infobox deaktiviert?
InteractiveGetStatus	Meldet, ob der interaktive Modus eingeschaltet ist (nur TF5800)
InteractiveSetStatus	Aktiviert oder deaktiviert den interaktive Modus (nur TF5800)
isAnyOSDVisible	Prüft, ob ein bestimmter Bildschirmbereich ein OSD enthält
isOSDRegionAlive	Prüft, ob eine OSD-Region noch aktiv ist
MHEG_Status	Liefert den aktuellen MHEG-Status
OSDCopy	Kopiert einen OSD-Buffer x-mal
SaveBitmap	Speichert den gecaptureten Bildschirminhalt in eine BMP-Datei
SetRemoteMode	(De-)aktiviert einen Remote-Modus
ShowMessageWin	Zeigt ein Meldungsfenster an
ShowMessageWindow	Erweiterte Version von ShowMessageWin
SoundSinus	Beep
SubtitleGetStatus	Meldet, ob die Untertitel aktiviert sind
SubtitleSetStatus	(De-)Aktiviert die Untertitel
TAP_Osd_PutFreeColorGd	Kopiert ein Gd-Objekt in einen OSD-Bereich mit einer vorgegebenen Farbe
TunerGet	Liefert die Tuner-Nummer des Haupt- oder PIP-Fensters
TunerSet	Setzt die Tuner-Nummer des Hauptservices
YUVB	Liefert die blaue Komponente einer YUV-Farbe
YUVG	Liefert die grüne Komponente einer YUV-Farbe
YUVR	Liefert die rote Komponente einer YUV-Farbe

DIALOG

DialogEvent	Leitet Events an die Dialogfenster weiter. Verarbeitete Events (z.B. Tastendrücke für Scrollvorgänge) werden auf 0 zurückgesetzt.
DialogMsgBoxButtonAdd	Fügt einen Knopf zu einer Messagebox hinzu (5 max)
DialogMsgBoxExit	Beendet eine Messagebox
DialogMsgBoxInit	Initialisiert die Strukturen eine Messagebox
DialogMsgBoxShow	Blendet eine allgemeine Messagebox ein
DialogMsgBoxShowInfo	Zeigt eine knopflose Messagebox an
DialogMsgBoxShowOK	Zeigt eine Messagebox mit einem OK-Knopf an
DialogMsgBoxShowOKCancel	Zeigt eine Messagebox mit "OK"- und "Abbrechen"-Knopf an
DialogMsgBoxShowYesNo	Zeigt eine Messagebox mit "Ja"- und "Nein"-Knopf an
DialogMsgBoxShowYesNoCancel	Zeigt eine Messagebox mit "Ja"-, "Nein"- und "Abbrechen"-Knopf an
DialogMsgBoxTitleSet	Ändert den Titel einer Messagebox
DialogProfileChange	Setzt ein neues Userprofil
DialogProfileLoad	Lädt ein Userprofil
DialogProfileLoadDefault	Lädt das Standarduserprofil (/ProgramFiles/DialogProfile.ini)
DialogProfileSave	Sichert ein Userprofil
DialogProfileSaveDefault	Macht das aktuelle Profil zum Standardprofil
DialogProfileScrollBehaviourChange	Ändert das Scrollverhalten im Userprofil
DialogProgressBarExit	Blendet den Fortschrittsbalken aus und beendet diesen
DialogProgressBarInit	Initialisiert die Strukturen einer Fortschrittsbalken
DialogProgressBarSet	Setzt den Balken an eine bestimmte Position
DialogProgressBarShow	Blendet den Fortschrittsbalken ein
DialogProgressBarTitleSet	Ändert den Titel des Forschrittsbalken
DialogWindowAlpha	Macht ein Dialogfenster temporär durchsichtiger
DialogWindowChange	Setzt ein neues Fensterprofil
DialogWindowCursorChange	Schaltet den Cursor ein- bzw. aus
DialogWindowCursorSet	Setzt den Cursor auf eine bestimmte Zeile
DialogWindowExit	Beendet ein Dialogfenster
DialogWindowHide	Blendet ein Dialogfenster aus
DialogWindowInfoAddIcon	Fügt ein Icon in den Info-Bereich eines Dialogs hinzu
DialogWindowInfoAddS	Fügt einen Text in den Info-Bereich eines Dialogs hinzu
DialogWindowInfoDeleteAll	Löscht den Info-Bereich eines Dialogs
DialogWindowInit	Initialisiert die Dialogstrukturen
DialogWindowItemAdd	Fügt einen neuen Eintrag zu einem Dialogfenster hinzu
DialogWindowItemAddSeparator	Fügt eine Abstandszeile zu einem Dialogfenster hinzu
DialogWindowItemChangeFlags	Ändert die Flags einer Dialogzeile (auswählbar, aktiv)

DialogWindowItemChangeIcon	Ändert das Icon einer Dialogzeile
DialogWindowItemChangeParameter	Ändert den Parametertext einer Dialogzeile
DialogWindowItemChangeValue	Ändert den Wertetext einer Dialogzeile
DialogWindowItemDelete	Löscht einen Dialogeintrag [noch nicht implementiert]
DialogWindowItemDeleteAll	Löscht alle Dialogeinträge
DialogWindowRefresh	Zeichnet alle geänderten Parameter
DialogWindowReInit	Verändert die Fenstergeometrie
DialogWindowScrollDown	Scrollt eine Zeile hinunter
DialogWindowScrollDownPage	Scrollt eine Seite hinunter
DialogWindowScrollUp	Scrollt eine Zeile hinauf
DialogWindowScrollUpPage	Scrollt eine Seite hinauf
DialogWindowShow	Blendet ein Dialogfenster ein
DialogWindowTabulatorSet	Setzt die Tabulatorposition in der Parameter- oder Wertespalte
DialogWindowTitleChange	Ändert den Titel eines Dialogs
DialogWindowTypeChange	Ändert den Fenstertyp
InfoTestGrid	Zeichnet ein 10x10 Pixel-Gitter in den Info-Bereich (für TAP-Entwicklung)

COMPRESSION

CompressBlock	Komprimiert einen Datenblock
CompressedTFDSize	Liefert die Größe eines komprimierten TFD-Struktur
CompressTFD	Komprimiert einen Datenblock in eine TFD-Struktur
CRC16	Berechnet eine 16-Bit CRC eines Speicherbereichs
TFDSize	Liefert die Größe einer TFD-Struktur
UncompressBlock	Dekomprimiert einen Datenblock
UncompressedFirmwareSize	Liefert die dekomprimierte Größe eines komprimierten Firmware
UncompressedLoaderSize	Liefert die dekomprimierte Größe eines komprimierten Loaders
UncompressedTFDSize	Liefert die dekomprimierte Größe einer komprimierten TFD-Datei
UncompressFirmware	Dekomprimiert eine gepackte Firmware
UncompressLoader	Dekomprimiert einen gepackten Loader
UncompressTFD	Dekomprimiert eine gepackte TFD-Datei

FILESELECTOR

FileSelector	Ruft den FileSelector auf
FileSelectorKey	Prüft, ob die Taste vom FileSelector verwendet wird

FLASH

AddSec	Addiert oder subtrahiert Sekunden von einer Uhrzeit
AddTime	Addiert oder subtrahiert Minuten von einer Uhrzeit
DATE	Rechnet Datum, Stunden und Minuten in das Topfield interne Datumsformat um
FlashAddFavourite	Fügt einen neuen Favoriten hinzu
FlashDeleteFavourites	Löscht alle Favoriten
FlashFindEndOfServiceNameTableAddress	Liefert einen Pointer zum Ende Service-Name-Tabelle
FlashFindEndOfServiceTableAddress	Liefert einen Pointer zum Ende der Service-Tabelle
FlashFindServiceAddress	Liefert einen Pointer zu einem bestimmten Service
FlashFindTransponderIndex	Liefert einen Pointer zu einem bestimmten Transponder
FlashGetBlockStartAddress	Liefert einen Pointer zum Anfang eines Blocks
FlashGetChannelNumber	Liefert die Frequenz und Kanalnummer zu einem Service (nur DVB-t)
FlashGetSatelliteByIndex	Liefert einen Pointer auf einen bestimmten Satelliten (nur DVB-s)
FlashGetServiceByIndex	Liefert einen Pointer auf ein per Index referenziertes Service
FlashGetServiceByName	Liefert einen Pointer auf ein per Namen referenziertes Service
FlashGetTransponderCByIndex	Liefert einen Pointer auf einen bestimmten Transponder (nur DVB-c)
FlashGetTransponderSByIndex	Liefert einen Pointer auf einen bestimmten Transponder (nur DVB-s)
FlashGetTransponderTByIndex	Liefert einen Pointer auf einen bestimmten Transponder (nur DVB-t)
FlashGetType	Liefert den Typ des Receivers (DVB-s, -t, oder -c)
FlashProgram	Schreibt die Kopie des Flash im Speicher in den Flash-Baustein
FlashReindexFavourites	Passt die Favoriten-Indices an, falls ein Service hinzugefügt oder gelöscht wurde

FlashReindexTimers	Passt die Timer-Indices an, falls ein Service hinzugefügt oder gelöscht wurde
FlashRemoveCASServices	Entfernt alle als verschlüsselt gekennzeichneten ($) Services
FlashRemoveServiceByIndex	Entfernt ein per Index angegebenes Service
FlashRemoveServiceByIndexString	Entfernt ein per Index angegebenes Service (Index liegt als String vor)
FlashRemoveServiceByLCN	Entfernt ein per LCN angegebenes Service (DVB-c und DVB-t)
FlashRemoveServiceByName	Entfernt ein per Namen angegebenes Service
FlashRemoveServiceByPartOfName	Entfernt ein per Teil des Namens angegebenes Service
FlashRemoveServiceByUHF	Entfernt ein per UHF-Kanal angegebenes Service
FlashServiceAddressToServiceIndex	Berechnet aus einem Service-Pointer einen Index
FlashWrite	Schreibt Daten in den Flash-Speicher (nur verwenden, wenn man weiss was man tut!)
GetEEPROMAddress	Liefert einen Pointer auf das EEPROM Shadow im Speicher
GetEEPROMPin	Liefert den aktuellen Pin-Code
HOUR	Extrahiert die Stunden aus dem Topfield internen Datumsformat
MINUTE	Extrahiert die Minuten aus dem Topfield internen Datumsformat
MJD	Extrahiert das Datum aus dem Topfield internen Datumsformat
TimeDiff	Berechnet die Differenz in Minuten zwischen zwei Zeiten

HARDDISK

HDD_AAM_Disable	Schaltet das Automatic Acoustic Management aus
HDD_AAM_Enable	Schaltet das Automatic Acoustic Management ein
HDD_APM_Disable	Schaltet das Advanced Power Management aus
HDD_APM_Enable	Schaltet das Advanced Power Management ein
HDD_BigFile_Read	Liest einen Sektor aus einer Datei (nicht auf 2GB beschränkt)
HDD_BigFile_Size	Liefert die Größe einer Datei in Sektoren
HDD_BigFile_Write	Schreibt einen Sektor aus einer Datei (nicht auf 2GB beschränkt). Diese Funktion kann eine Datei nicht vergrößern!
HDD_ChangeDir	Ändert das aktuelle Verzeichnis und akzeptiert einen vollständigen Pfad
HDD_FappendOpen	Öffnet eine Datei um Text anzuhängen
HDD_FappendWrite	Hängt Text an das Ende einer Textdatei an
HDD_FreeSize	Liefert den freien Platz der HDD in MB. Diese Funktion greift nicht auf die Disk zu und weckt sie somit nicht aus dem Standby.
HDD_GetClusterSize	Liefert die Clustergröße in Sektoren
HDD_GetFirmwareDirCluster	Liefert die Clusternummer des aktuellen Verzeichnisses der Firmware
HDD_GetHddID	Liefert den Typ, die Seriennummer und die Firmwareversion der Festplatte
HDD_GetHddInfo	Liefert ein paar Daten zur eingebauten HDD
HDD_IdentifyDevice	Liefert den IdentifyDevice-Informationsblock
HDD_LiveFS_GetChainLength	Liefert die Anzahl der Cluster einer FAT-Kette.
HDD_LiveFS_GetFAT1Address	Liefert einen Pointer auf die FAT im Speicher
HDD_LiveFS_GetFAT2Address	Liefert einen Pointer auf die Kopie der FAT im Speicher
HDD_LiveFS_GetFirstCluster	Liefert den ersten Cluster einer FAT-Kette
HDD_LiveFS_GetLastCluster	Liefert den letzten Cluster einer FAT-Kette
HDD_LiveFS_GetNextCluster	Liefert den nächsten Cluster einer FAT-Kette
HDD_LiveFS_GetPreviousCluster	Liefert den vorhergehenden Cluster einer FAT-Kette (sehr langsam!)
HDD_LiveFS_GetRootDirAddress	Liefert einen Pointer auf das Root-Verzeichnis im Speicher
HDD_LiveFS_GetSuperBlockAddress	Liefert einen Pointer auf den Superblock im Speicher
HDD_Move	Verschiebt eine Datei in ein anderes Verzeichnis
HDD_ReadClusterDMA	Liest einen Cluster im DMA-Modus
HDD_ReadSector	Liest einen Cluster im PIO-Modus. Zu diesem Zeitpunkt darf kein anderer Prozess auf die HDD zugreifen (Play, Rec, TS).
HDD_ReadSectorDMA	Liest einen Sektor im DMA-Modus
HDD_SetCryptFlag	Setzt das Crypt-Flag einer geöffneten Datei
HDD_SetFileDateTime	Setzt das Datum und die Uhrzeit einer geöffneten Datei
HDD_SetSkipFlag	Setzt das Skip-Flag einer geöffneten Datei
HDD_SetStandbyTimer	Setzt den Standby-Timeout der Festplatte
HDD_Smart_DisableAttributeAutoSave	Deaktiviert das SMART Attribute Auto-Save Feature
HDD_Smart_DisableOperations	Deaktiviert SMART
HDD_Smart_EnableAttributeAutoSave	Aktiviert das SMART Attribute Auto-Save Feature

HDD_Smart_EnableOperations	Aktiviert SMART
HDD_Smart_ExecuteOfflineImmediate	Startet einen SMART-Offline-Check
HDD_Smart_ReadData	Liest die SMART-Information von der Festplatte. Zu diesem Zeitpunkt darf kein anderer Prozess auf die Platte zugreifen. (Play, Rec, TS)
HDD_Smart_ReadThresholdData	Liest die SMART-Threshold-Information von der Festplatte. Zu diesem Zeitpunkt darf kein anderer Prozess auf die Platte zugreifen. (Play, Rec, TS)
HDD_Smart_ReturnStatus	Liefert den SMART Status (good, failed)
HDD_Stop	Schaltet die HDD ab
HDD_TouchFile	Setzt das Dateidatum auf die aktuelle Zeit
HDD_TranslateDirCluster	Übersetzt eine Clusternummer in einen Verzeichnisnamen
HDD_TruncateFile	Schneidet eine Datei an einer bestimmten Stelle ab [Cluster-Grenzen werden nicht unterstützt]
HDD_Write	Gebuffertes Schreiben von kleinen Datenmengen
HDD_WriteClusterDMA	Schreibt einen Cluster auf die Festplatte
HDD_WriteSectorDMA	Schreibt einen Sektor auf die Festplatte

HOOKS

HookEnable	(De-)Aktiviert einen Hook
HookExit	Deaktiviert alle Hooks
HookIsEnabled	Wurde ein bestimmter Hook aktiviert?
HookMIPS_Clear	Deaktiviert den MIPS-Hook
HookMIPS_Set	Setzt einen Firmware-Hook, der den Ablauf auf ein MIPS-Programm umleitet
HookSet	Setzt einen Firmware-Hook, der den Ablauf auf ein TAP umleitet

IIC BUS

ReadEEPROM	Liest den Inhalt des EEPROMs
ReadIICRegister	Liest ein Register eines IIC-Controllers (EEPROM, A/V-Matrix, Tuner)
WriteIICRegister	Schreibt ein Register eines IIC-Controllers

IMEM

IMEM_Alloc	Reserviert Speicher
IMEM_Compact	Reorganisiert die internen Tabellen des Speichermanagers
IMEM_Free	Gibt reservierten Speicher frei
IMEM_GetInfo	Liefert eine Info über den verfügbaren Speicher
IMEM_Init	Initialisiert den Memory Manager
IMEM_isInitialized	Wurde der Speichermanager bereits initialisiert?
IMEM_Kill	Schließt den Speichermanager und gibt den reservierten Speicher frei

INI FILES

INICloseFile	Schließt eine INI-Datei
INIGetARGB	Liefert eine ARGB-Farbe
INIGetRGB	Liefert eine RGB-Farbe
INIGetRGB8	Liefert eine RGB-Farbe. In der INI-Datei stehen 8-Bit-Werte
INIGetHexByte	Liefert ein Byte
INIGetHexDWord	Liefert ein DWord
INIGetHexWord	Liefert ein Word
INIGetInt	Liefert einen Integer
INIGetString	Liefert einen String
INIKeyExists	Prüft, ob ein bestimmter Schlüssel existiert
INIKillKey	Entfernt einen Key aus der geöffneten INI-Datei
INILocateFile	Liefert die Position einer Datei (aktuelles dir, /PF, /PF/Settings, /PF/Settings/<AppName>)
INIOpenFile	Öffnet eine INI-Datei
INISaveFile	Speichert die geöffnete INI-Datei
INISetARGB	Setzt eine ARGB-Farbe
INISetRGB	Setzt eine RGB-Farbe
INISetRGB8	Setzt eine auf 8-Bit erweiterte RGB-Farbe

INISetComments	[Kommentare sind noch nicht implementiert]
INISetHexByte	Setzt ein Byte
INISetHexDWord	Setzt ein DWord
INISetHexWord	Setzt ein Word
INISetInt	Setzt einen Integer
INISetString	Setzt einen String
LangGetString	Liefert einen sprachabhängigen Text
LangLoadStrings	Ladet eine sprachabhängige Text-INI
LangUnloadStrings	Gibt den Speicher mit den sprachabhängigen Texten frei

MASTERPIECE / 6000 VFD

MPDisplayClearDisplay	Löscht das Display
MPDisplayClearSegments	Löscht einzelne Segmente
MPDisplayDisplayLongString	Zeigt einen String im 8-stelligen Display
MPDisplayDisplayShortString	Zeigt einen String im 4-stelligen Display
MPDisplayGetDisplayByte	Liefert ein Byte des 48 Byte großen Displaybuffers
MPDisplayGetDisplayMask	Liefert ein Byte des 48 Byte großen Maskbuffers
MPDisplayInstallMPDisplayFwHook	Aktiviert den MP-Display Hook
MPDisplaySetAmFlag	Setzt die AM-Anzeige
MPDisplaySetColonFlag	Setzt den Doppelpunkt im 4-stelligen Display
MPDisplaySetDisplayByte	Setzt ein Byte des 48 Byte großen Displaybuffers
MPDisplaySetDisplayMask	Setzt ein Byte des 48 Byte großen Maskbuffers
MPDisplaySetDisplayMemory	Kopiert einen 48 Byte großen Buffer in den Displaybuffer
MPDisplaySetDisplayMode	Aktiviert die verschiedenen Anzeigenmodi
MPDisplaySetPmFlag	Setzt die PM-Anzeige
MPDisplaySetSegments	Setzt einzelne Segmente
MPDisplayToggleSegments	Ändert einzelne Segmente
MPDisplayUninstallMPDisplayFwHook	Deaktiviert den MP-Display Hook
MPDisplayUpdateDisplay	Zeigt den VFD-Buffer an

PATCHES

PatchLoadModule	Lädt eine TFP-Datei
PatchLoadModuleGP	Lädt aus einer TFP-Datei nur den spezifischen Patch für das angegebene $gp und alle allgemeinen Patches. Damit ist sie speichersparender als PatchLoadModule.
PatchUnloadModule	Entfernt die TFP-Datei aus dem Speicher
PatchFindType	Durchsucht den Speicher, ob ein Patch installierbar ist
PatchApply	Installiert oder entfernt einen Patch
PatchInstallID	Registriert eine PatchID
PatchIsInstalled	Liefert die Info, ob eine bestimmte PatchID registriert ist
PatchGetInstalled	Liefert einen String mit allen PatchIDs
PatchRemoveID	Entfernt eine PatchID
PatchCleanList	Kopiert alle PatchIDs in eine temporäre Liste und entfernt sie aus der F/W
PatchReinstallList	Kopiert die PatchIDs aus der temp. Liste zurück in die F/W

REC STREAMS

HDD_DecodeRECHeader	Identifiziert einen REC-Header und dekodiert diesen
HDD_EncodeRECHeader	Schreibt einen REC-Header
HDD_FindPCR	Liefert den ersten PCR eines REC-Streams
HDD_FindPMT	Sucht und dekodiert die erste PMT in einem REC-Stream und schreibt die Information in die REC-Header-Struktur.
HDD_isAnyRecording	Nimmt der Topf überhaupt gerade auf?
HDD_isCryptedStream	Prüft, ob ein REC-Stream verschlüsselte Pakete enthält
HDD_isRecording	Nimmt der Topf gerade auf einem Slot auf?
HDD_MakeNewRECName	Fügt eine Sequenznummer oder 2 Zufallszeichen an einen REC-Dateinamen an
HDD_PausePlayback	Pauses a playback and continues it again
HDD_PlaySlotGetAddress	Liefert die Adresse der tPlaySlot-Struktur im Speicher
HDD_RecalcPlaytime	Berechnet die Spielzeit einer REC-Datei
HDD_RECSlotGetAddress	Liefert die Adresse der tRECSlot-Struktur im Speicher
HDD_RECSlotIsPaused	Liefert die Information, ob eine Aufnahme gerade pausiert
HDD_RECSlotPause	Kann ein Aufnahme pausieren lassen
HDD_RECSlotSetDuration	Ändert die Aufnahmedauer einer laufenden Aufnahme

SHUTDOWN

Reboot	Rebootet den Topf
Shutdown	Stop einen Task (Rec, Play, Video or Audio) oder schaltet den Topf ab

STRINGS

ExtractLine	Liefert eine Zeile aus einem Text
GetLine	Liefert eine Zeile aus einem Text (verbesserte Version)
LowerCase	Konvertiert einen String in Kleinbuchstaben
MakeValidFileName	Entfernt illegale Zeichen aus einem String
ParseLine	Sucht in einem String nach einem Zeichen und teilt den String an dieser Stelle
RTrim	Schneidet Leerzeichen am Ende eines Strings ab
SeparatePathComponents	Zerlegt einen Dateipfad in seine Komponenten
StrEndsWith	Prüft, ob ein String mit bestimmten Zeichen endet
TimeFormat	Erzeugt einen String aus einer Uhrzeit
UpperCase	Konvertiert einen String in Grossbuchstaben
ValidFileName	Entfernt alle illegalen Zeichen aus einem Dateinamen

TAPS

HDD_TAP_Callback	Ruft eine beliebige Routine eines andern TAPs auf.
HDD_TAP_Disable	Deaktiviert ein TAP (das Ziel empfängt keine Events mehr)
HDD_TAP_DisableAll	Deaktiviert alle TAPs außer den Aufrufer
HDD_TAP_GetCurrentDir	Liefert das aktuelle Verzeichnis des TAPs
HDD_TAP_GetCurrentDirCluster	Liefert den HDD-Cluster, der das aktuelle Verzeichnis enthält
HDD_TAP_GetIDByFileName	Liefert die TAPID aus einer .tap-Datei
HDD_TAP_GetIDByIndex	Liefert die TAPID eines laufenden TAPs via Index der TAP-Tabelle
HDD_TAP_GetIndexByID	Sucht nach TAPID in den laufenden TAPs und liefert dessen Index in der TAP-Tabelle
HDD_TAP_GetInfo	Liefert einige Informationen über ein geladenes TAP
HDD_TAP_GetStartParameter	Liefert einen Pointer auf den Parameter-Block eines Server-TAPs
HDD_TAP_isAnyRunning	Prüft, ob irgendein anderes TAP läuft
HDD_TAP_isBatchMode	Prüft, ob das TAP im Batch-Mode gestartet wurde.
HDD_TAP_isDisabled	Wurde einspezifisches TAP deaktiviert? Das inkludiert auch TAPs, die via TAPCommander deaktiviert wurden.
HDD_TAP_isDisabledAll	Wurden alle TAPs deaktiviert?
HDD_TAP_isRunning	Prüft, ob ein spezifisches TAP läuft.
HDD_TAP_PushDir	Schreibt das aktuelle TAP-Verzeichnis in einen Stack
HDD_TAP_PopDir	Holt einen Eintrag vom Verzeichnisstack und wechselt zu diesem
HDD_TAP_SendEvent	Sendet einen Event an eines oder alle TAPs. Es gibt keine Beschränkungen bezüglich event,

	param1 und param2. Der Event wird nicht an die Firmware weitergegeben.
HDD_TAP_SetCurrentDirCluster	Setzt das aktuelle Verzeichnis
HDD_TAP_Start	Startet ein anderes TAP.
HDD_TAP_StartedByTAP	Wurde das TAP von einem anderen TAP gestartet?
HDD_TAP_Terminate	Beendet ein TAP [ungetested]

TAPCOM

TAPCOM_CloseChannel	Client schließt den Kommunikationskanal zum Server
TAPCOM_Finish	Server hat den Befehl abgearbeitet
TAPCOM_GetChannel	Server holt die Details zur Anfrage des Clients
TAPCOM_GetReturnValue	Liefert dem Client den Rückgabewert des Servers
TAPCOM_GetStatus	Client ruft den aktuellen Status des Kommunikationskanals ab
TAPCOM_LastAlive	Wann war der Server zuletzt aktiv? (asynchrone Kommunikation)
TAPCOM_OpenChannel	Client öffnet eine Verbindung zum Server
TAPCOM_Reject	Server kann den Auftrag des Clients (derzeit) nicht ausführen
TAPCOM_StillAlive	Server meldet, dass er noch beschäftigt ist

TAPAPIFIX

InitTAPAPIFix	Initialisiert diesen Teil der Library. Dadurch werden verschiedene Funktionen durch bugbereinigte Versionen ersetzt (siehe unten).

TAPAPIFIX

Folgende Bugs werden durch tapapifix abgefangen:

TAP_Hdd_GetPlayInfo	Bufferüberlauf
TAP_Hdd_GetRecInfo	Bufferüberlauf
TAP_Hdd_Fseek	Falsche SEEK_END-Position bei Dateien, die ein Vielfaches von 512 Bytes groß sind.
TAP_Hdd_Flen	Bei Dateien, die ein Vielfaches von 512 Bytes groß sind, werden 512 Bytes zu wenig gemeldet
TAP_Hdd_ChangeDir	Unterschiedlicher Rückgabewert bei Firmwareversionen vor V5.12.0
TAP_Channel_SetAudioTrack	Nach dem Springen in einer Wiedergabe wird der default Audio-Track wiedergegeben
TAP_Hdd_Delete	Freeze beim Löschen einer Datei, die einen Punkt weit vorne im Dateinamen enthält
TAP_Hdd_Fwrite	Schreibt keinen Müll an das Ende einer Datei
TAP_Hdd_StopTs	Stürzt im Normal-Mode nicht mehr ab

FIREBIRDLIB EINBINDEN

* Ein #include "libFirebird.h" in die .c-Datei einfügen
* Die Build.bat modifizieren und das FireBird-Archiv hinzufügen. Z. B.:

```
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
```

Die FireBird-Library muss die erste –l Option sein. Wird der original Topfield-Compiler verwendet, so ist die Lib an der letzten Stelle nocheinmal notwendig (siehe oben).

HOWTO

FIRMWARE HOOKS

Ein Hook leitet die Ausführung eines Codes in der Firmware auf ein TAP um. Damit lassen sich CPU-Register oder Speicherstellen analysieren und modifizieren. Leider lässt sich nicht jede beliebige Stelle in der Firmware hooken. Während manche Routinen trotz Hook stabil weiterlaufen, nehmen andere den Hook übel und der Topf stürzt ab.

Zuerst wird im TAP ein Hook-Handler definiert:

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

Danach kann man einen Hook setzen und aktivieren:

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

Es lassen sich auch mehrere Hooks gleichzeitig aktivieren. Der von HookSet zurückgegebene Index wird im HookIndex-Parameter des HookHandlers mitgegebene. Dadurch lassen sich die einzelnen Hooks unterscheiden. Zum Schluss nicht auf’s Deaktivieren vergessen!

```
HookExit();
```

TAPCOM

EIN ANDERES TAP STARTEN

HDD_TAP_Start startet ein anders TAP und erlaubt die Übergabe von Parametern und einem Batch-Flag. Das Batch-Flag kann dafür benützt werden, um einen bestimmte Operation ohne Benutzerintervention auszuführen. 3PG zum Beispiel kann im Batch-Modus gestartet werden. In diesem Modus führt es einen vollständigen Scan aus und beendet sich danach wieder. Parameter und Batch-Flag sind nur in der TAP_Main gültig!

```
-- der client ----------------------------------------------------

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

tParameters    Parameters;
dword          TAPID;

//Der Client startet den Editor und übergibt das Verzeichnis und den Namen der zu bearbeiteten Datei
//Der Batch-Modus wir dnicht verwendet.
TAP_SPrint (Parameters.Dir, "/ProgramFiles");
TAP_SPrint (Parameters.FileName, "3PG.ini");
HDD_TAP_Start ("editor.tap", FALSE, &Parameters, &TAPID);

//Auf das Beenden des Editors warten
do
{
  TAP_SystemProc();
} while (HDD_TAP_isRunning (TAPID_Editor));


-- der server ----------------------------------------------------

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

tParameters    *Parameters;

int TAP_Main(void)
{
  //Der Editor benützt den batch-Modus nicht, aber so würde es funktionieren
  if (HDD_TAP_isBatchMode())
  {
    ...
  }

  //Gibt es Parameter?
  Parameters = HDD_TAP_GetStartParameter();
  if (Parameters != NULL)
  {
    //Parameter sichern, da sie nur in der TAP_Main gültig sind
  }
  ...
}
```

SYNCHRONE KOMMUNIKATION

Das bedeutet, dass der Client sofort eine Rückmeldung erhält und dass vom Server keine weitere Reaktion zu erwarten ist. Im Gegensatz dazu stehen die asynchrone Kommunikation (Abarbeitung eines Befehls dauert länger) und der Broadcast (eine Info wird an mehrere Server gesandt). Dazu aber ein anderes Mal.

Das Simple Demo-Beispiel (siehe .Demo-Verzeichnis) besteht aus einem Server und 2 Clients. Der Server bietet eine Funktion an, die 2 Zahlen addiert. Damit die Sache nicht zu einfach wird, muss der Client vor der Berechnung den Server für sich reservieren. Diese Reservierung hat aber nichts mit der TAPCOM zu tun, sondern ist einfach ein Feature unseres Demo-Servers (z.B. wie sich ein Client die Kontrolle über das FP-Display reservieren kann). In der Demo werden folgende Tasten verwendet:

1 = Client 1 reserviert Server
2 = Client 1 lässt Server die Berechnung durchführen
3 = Client 1 koppelt sich vom Server wieder ab

Die Tasten 4 bis 6 machen das Selbe mit dem Client 2. UHF beendet die Clients und den Server. Ausgaben erfolgen nur über die serielle Konsole.

Der Client

Der Ablauf ist immer der Selbe:

- Einen Kanal öffnen und damit den Befehl an den Server absetzen
- Den Status und eventuell den Rückgabewert des Kanals abfragen
- Den Kanal wieder schließen

```
//Ruft den Server auf und sendet den Befehl ReserveServer. Es werden keine Parameter übergeben.
Channel = TAPCOM_OpenChannel (TAPCOM_App_DemoServer, TAPCOM_DemoServer_ReserveServer, 0, NULL);

//Status abfragen und entsprechen reagieren.
Status = TAPCOM_GetStatus (Channel);

//Kanal schliessen
TAPCOM_CloseChannel (Channel);
```

Der Client kann folgende Statuswerte erhalten:

TAPCOM_Status_SERVER_NOT_AVAILABLE: das Server-TAP wurde gar nicht gestartet. Entweder lädt es der Client nach oder setzt irgendwelche Verzweiflungstaten.

TAPCOM_Status_REJECTED: der Befehl ist unbekannt oder kann zum jetzigen Zeitpunkt nicht abgearbeitet werden (z.B. wenn der Server bereits von einem anderen Client reserviert wurde).

TAPCOM_Status_FINISHED: der Befehl wurde abgearbeitet. Ob erfolgreich oder nicht steht im Rückgabewert, der via ***TAPCOM_GetReturnValue*** abgefragt werden kann.

TAPCOM_Status_VERSIONMISMATCH: Client und Server verwenden inkompatible Versionen der TAPCOM-Lib. Der Server hat den Befehl somit gar nicht erhalten.

Es gibt noch 2 weitere Werte, die in unserem Fall nicht vorkommen sollten:

TAPCOM_Status_OPEN: der Server ist zwar geladen, hat aber nicht auf unseren Befehl reagiert (schlecht bei direkter Kommunikation, normal bei Broadcasts).

TAPCOM_Status_ACKNOWLEDGED: der Server hat den Befehl erhalten, die Abarbeitung dauert aber. Dieser Status wird bei der asynchronen Kommunikation verwendet.

Parameter und Daten sind optional und deren Struktur wird vom Server vorgegeben. Der Client reserviert den Speicherplatz für die Daten und der Server greift danach via Pointer auf die Daten zu. Deswegen darf ein Client den Speicherplatz nicht freigeben, solange der Status ACKNOWLEDGED oder OPEN ist. In unserem Beispiel initialisiert der Client die beiden ersten Variablen der Struktur und erhält das Ergebnis vom Server in der Variable Result. Bei den folgenden Zeilen fehlt der Einfachheit halber die Überprüfung, ob der Server die Berechnung überhaupt durchgeführt hat (ist in der Demo vollständig).

```
typedef struct
{
  dword                 Number1;
  dword                 Number2;
  dword                 Result;
} tTAPCOM_DemoServer_Parameter;      //In der TAPCOM_DemoServer.h deklariert

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

Wie oben zu sehen ist, sind für den kompletten Ablauf maximal 4 TAPCOM-Befehle notwendig.

Der Server

Die Implementierung eines Servers läuft ähnlich wie beim Client. Der Server erhält die Anforderung eines Clients als Event (EVT_TAPCOM) serviert. Dabei handelt es sich aber um keinen echten Event, wie z.B. EVT_KEY, und deshalb wird dieser auch nicht zur Firmware weitergeleitet. Danach holt er sich die Details mittels ***TAPCOM_GetChannel***. Ist der Rückgabewert dieser Funktion 0, dann war dieser Server nicht gemeint und der Event kann einfach ignoriert werden. Sonst führt der Server die gewünschte Operation durch und beendet die Kommunikation mittels ***TAPCOM_Finish***. Der in dieser Funktion übermittelte Rückgabewert kann vom Client mittels **TAPCOM_GetReturnValue** abgefragt werden. Falls der Befehl nicht ausgeführt werden kann, teilt der Server dies dem Client mittels ***TAPCOM_Reject*** mit. Und damit sind auch bereits alle serverseitigen TAPCOM-Befehle besprochen. Ein Server kann in der Programmierung deshalb etwas komplizierter werden, da er die Aufrufe der Clients auf Plausibilität prüfen sollte. Dies hat aber nichts direkt mit der TAPCOM-Lib zu tun. Im einfachsten Fall ist es aber auch nur ein ***TAPCOM_GetChannel*** und ein ***TAPCOM_Finish***.

Unser Beispiel-Server führt folgende Aufgaben durch:

- Ausführen der 3 selbst definierten Befehle „ReserveServer“, „ExecAdd“ und „ReleaseServer“.
- Ablehnen von Anforderungen von Clients, die den Server nicht für sich reserviert haben.
- Prüfung, ob der Client, der den Server reserviert hat, überhaupt noch aktiv ist. Freigabe der Reservierung falls dem nicht so ist.

Details siehe Beispiel im .Demo-Verzeichnis.

Asynchrone Kommunikation

Die asynchrone Kommunikation wird dann verwendet, wenn die Abarbeitung eines Befehls länger dauert.

Client

Vergleich mit synchroner Kommunikation:

- Vom Server wird sofort der Status *TAPCOM_Status_ACKNOWLEDGED* oder *TAPCOM_Status_REJECTED* zurückgegeben.
- Der Client muss laufend ***TAPCOM_GetStatus*** aufrufen, um die Beendigung des Auftrages mitzubekommen
- Der Client kann laufend ***TAPCOM_LastAlive*** aufrufen. Diese Funktion liefert den Tickcount retour, wann sich der Server zuletzt als noch aktiv gemeldet hat.

Server

Vergleich mit synchroner Kommunikation:

- Der Server gibt sofort der Status *TAPCOM_Status_ACKNOWLEDGED* oder *TAPCOM_Status_REJECTED* zurück.
- Der Server muss laufend ***TAPCOM_StillAlive*** aufrufen, damit der Client merkt, dass der Server noch mit seinem Auftrag beschäftigt ist.

Broadcast

In diesem Fall wird eine Meldung an alle TAPs gesandt (TargetID bei OpenChannel = TAPCOM_App_BROADCAST). Es kann ebenfalls ein Parameterblock übergeben werden und die Server können auch Daten zurückliefern, jedoch „gewinnt“ der letzte Server.

Vergleich mit synchroner Kommunikation:

- Ein Broadcast läuft immer synchron ab
- Die Server sollten weder mit TAPCOM_Finish noch mit TAPCOM_Reject antworten
- Nach dem Durchlauf aller TAPs ist der Channel-Status *TAPCOM_Status_OPEN*