TMA1
API Documentation

1. Introduction

TMA1 is the name of the protocol used by the WebControl system.

The TMA1 API is designed to allow a Topfield TMS PVR to be managed via a web interface or any other system capable of generating HTTP requests and parsing JSON responses.

WebContol accepts commands via URL syntax from a web browser as shown by the following example: `http://<PVR IP>:8000/api?function=config&action=get!` and responds to these commands are returned using JSON object syntax.

```
{"content":"config",
"sysid":33021,
"device":"TRF-7160",
"apptype":"",
"systemtype":6,
"bigendian":0,
"remotetype":2,
"maxrecstreams":4,
"version":263,
"recextension":".mpg",
"language_code":"en",
"utftoppy":1,
"tma1ver":"A-026",
"blockfactor":9024,
"autoppy":1,
"analogueinput":0,
"utcoffset":600,
"dst":1,
"timestamp":"yyyy/mm/dd hh:mm:ss"}
```

Values of the object properties returned are normally those that the Topfield TAP API provides or those provided by FireBirdLib. Although you do not have to know how to program a TAP to use this API, understanding some of the constants, structures and enumerated lists from the TAP programming environment would be extremely helpful. A list of useful TAP-related information can be found commencing on page 31.

2. URL Command Syntax

TMA1 syntax can be used with WebControl to control the PVR by using a variety of specially formed URLs to convey the command required. The format of the URL is as follows:

```
http://<PVR IP>:8000/api?function=<function>&action=<action>&tap=<tap>&session=<session>!param=value&param=value&
```

The `function` and `action` parameters are mandatory, all other parameters are optional.

Parameter	Description
`function`	Command group for WebControl to execute.
`action`	Specific command within the group specified by the `function`.
`tap`	The decimal TAPID of a TAP to forward this command to. If this field is present, WebControl does not require and other fields to be present but will pass all fields received unaltered.
`session`	Reserved for use.

All parameters and syntax must be in lower case. Parameter values can be in mixed-case.

All functions will respond with both the "GET" and "POST" HTTP commands, however, a number of functions pass parameters within the POST data and will only work correctly if invoked with a POST command. These commands will be noted accordingly in the body of the documentation.

The command is effectively broken into 2 parts: The section before the "!" and the section after the "!". Within WebControl, the section before the "!" is decoded first, this is used to identify the required command and activate the appropriate internal routine. The section after the "!" is then decoded as required by the internal routine that was called.

Parameters passed with a POST command must be terminated with a new line character, for example:

parmater1=value 1<new line>
parmater2=value 2<new line>

A list of valid functions can be found in section 4 on page 5.

WebControl can also serve files from the PVR's hard drive. When no absolute path is provided in the URL, it is assumed that the files reside in "`/mnt/hd/ProgramFiles/Settings/WebControl/public_html/`". When a full Linux path is provided, WebContol will serve any file from the PVR's internal and externally mounted media, for example: "`/mnt/hd/PhotoFiles/holidays/day1.jpg`".

WARNING: WebControl does not take into account the file size or location when serving a file. It will treat a 200 KB JPG the same as a 9 GB recording. Attempting to serve an extraordinarily large file (like a 9 GB recording) may result in an unpredictable outcome.

3. JSON Response Syntax

Once a TMA1API command is been received by WebControl, a response is sent to the browser in JSON format.

All JSON responses should also contain a "`content`" property. This property can be used to determine what data is represented in the object returned.

All JSON responses should also contain a "`timestamp`" property. This property can be used to determine when a command was executed and therefore if the contents may be stale and need refreshing. **Note:** The timestamp is based on the local time of the PVR, not the device requesting the data.

All dates are in the following format: "`YYYY/MM/DD hh:mm:ss`". This format is not 100% JSON-compliant (`YYYY-MM-DDThh:mm:ss.sssZ`), however, during the early development of TMA1 it was found that not all browsers accepted the standard JSON date format. The format used is a compromise that was found to be accepted by all browsers tested.

A number of JSON responses contain a "`count`" property. This property can be used to determine the number of child objects (like a timers list) are contained within the object returned.

A number of properties also have a "`_hex`" companion property, for example "`filename_hex`". A number of properties are processed by WebControl so that they are in a format compatible with UTF8 and JSON. Unfortunately this makes the property different from the actual value on the PVR meaning that some processes, such as renaming a file, will not work because the filename in the processed property does not actually exist. By passing the "`_hex`" properties, the TMA1 API can ensure that the original property is provided unaltered.

4. Supported Functions

Function	Action	Description	POST
`config`	`get`	Retrieve the configuration of the PVR.	
`setting /`	`get`	Retrieve a single front-end setting value.	
`settings`	`set`	Save a single front-end setting value.	
	`delete`	Delete a single front-end setting value.	
	`list`	Retrieve all front-end setting values in a single JSON object.	
	~~`<store>`~~	~~Future - Save multiple front-end setting values.~~	
	~~`<?update?>`~~		
	~~`<?purge>`~~		
`language`	`get`	Retrieve the content of the `language-XX.txt` file, where "`XX`" is the ISO language code of the PVR OSD.	
`status`	`get`	Retrieve the current point-in-time status of the PVR.	
`channel`	`get`	Retrieve a list of channels configured on the PVR.	
	`set`	Change the current channel displayed on the screen.	
	`getlogos`	Return a JSON object describing the channel ID and CSS sprite coordinates for channels with logos.	
`favourites / favorites`	`get`	Retrieve the "Favourites" groups configured on the PVR and the channels that those groups contain.	
`timers /`	`get`	Retrieve a list of timer reservations from the PVR.	
`timer`	`new / create`	Create a new timer reservation on the PVR.	√
	`modify / update`	Modify an existing timer reservation on the PVR.	√
	`delete`	Delete an existing timer reservation on the PVR.	√
`files / file`	`get`	Retrieve a list of recording files located on the PVR.	
	`play`	Commence playback of a specified recording file.	√
	`delete`	Delete a specified recording file.	√
	`rename`	Rename a specified recording file.	√
	`stop / stopplay`	Stop playback of the current file and return to live TV.	
	`stoprec`	Stop recording for the slot number provided.	
	`tree`	Return a list of directories on the PVR.	

Function	Action	Description	POST
	jump / skip	Move the current playback position.	
	pause	Pause current playback file.	
	resume	Resume current playback file.	
	duration	Change the duration of a recording-in-progress.	
logo / logos	get	Return a JSON object describing the channel ID and sprite file coordinates for channels with logos.	
	rebuild	Request that the TAP rebuild the logos files. This action will not be performed if noautologos=1.	
remote	send	Generate a remote control key press.	
ascii	send	Generate a "Remote ASCII" key event compatible withTMSRemote "direct" mode codes.	
epg / guide	get	Return a JSON object listing the EPG event details for the specified channel and time range.	
log	write	Write the contents of the POST parameter to the log file.	√
tap / taps	get	Return a JSON object listing the running TAPs.	
vol	get	Return a JSON object containing the current volume settings.	
	set	Change the current volume setting.	

4.1 "config"

`api?function=config&action=get!`

This command retrieves the relatively static configuration information from the PVR. This information is static unless the OSD language is changed or a DST time zone change occurs.

Property	Description
`content`	Literal "config".
`sysid`	Value returned by `TAP_GetSystemId()`.
`device`	Device name as contained in FirmwareTMS.dat.
`apptype`	Application type as contained in FirmwareTMS.dat.
`systemtype`	System type as contained in FirmwareTMS.dat.
`bigendian`	From FirmwareTMS.dat, 1 if PVR is big endian, 0 if little endian.
`remotetype`	Remote Control type as contained in FirmwareTMS.dat. Refer to the `REMOTE_TYPE` enumerated list.
`maxrecstreams`	Maximum number of concurrent recording streams as contained in FirmwareTMS.dat.
`version`	Value returned by `TAP_GetVersion()`.
`recextension`	The recording extension, "mpg." or "rec." as contained in FirmwareTMS.dat.
`language_code`	2 character ISO language language code derived from the results of `TAP_GetSystemVar(SYSVAR_OsdLan)` as per FireBird's `iso639_1()` function.
`utftoppy`	Result from FireBird's `isUTFToppy()`.
`tma1ver`	The current executing version of the WebControl TAP.
`blockfactor`	Factor to divide the file size by to calculate the number of "blocks" contained within a recording. 9024 for MPG models and 9028 for REC models. This is useful for calculating jumps into a file on playback.
`autoppy`	1 if this is an Australian model PVR, 0 if not. This is used for EPG genre descriptions because Australian models use non-ETSI descriptions for free-to-air services.
`analogueinput`	1 if this model support capture from external analogue sources.
`utcoffset`	Current time zone offset from UTC in minutes. **Note**: This includes any DST offset if DST is active.
`dst`	1 if the PVR is currently observing daylight savings time, 0 if not.
`hddmodel`	The model number of the hard drive in the PVR.
`hddserial`	The serial number of the hard drive in the PVR. This field can be used as a proxy to uniquely identify a PVR.
`hddfirmware`	The firmware version of the hard drive in the PVR.
`pvrname`	Descriptive name for the PVR from pvrname.ini.

`logginglevel`	The current logging level being used by WebControl: 0 - None; 1 - Basic; 2 - Detailed; 3 - Detailed + Network.
`epgsource`	WebControl can read EPG data from the SmartEPG v6.2a database. Values for this field can be "pvr" or "smartepg".
`module_count`	The number of subdirectories in the `public_html/modules` directory.
`modules`	Contains an array containing the `name` and `hex_name` for every subdirectory in the `public_html/modules` directory.
`timestamp`	Timestamp of when the config was generated based on the local PVR time.

4.2 "setting"

`api?function=setting&action=get!key=`*`keyname`*`&module=<`*`module`*`>&`
`api?function=setting&action=set!key=`*`keyname`*`&value=`*`keyvalue`*`&`
`api?function=setting&action=delete!key=`*`keyname`*`&`
`api?function=setting&action=list!module=<`*`module`*`>&`

This command group sets/retrieves/deletes front-end settings stored on the PVR. Including the optional "`modules`" parameter will return a setting specific to that web module.

Actions

Action	Meaning	Content Property
`get`	Retrieve a single front-end setting value.	setting_get
`set`	Save a new or update an existing single front-end setting value.	setting_set
`delete`	Delete a single front-end setting value.	setting_delete
`list`	Retrieve all front-end setting values in a single JSON object.	setting_list

For `get`, `set` and `delete`:

Property	Description
`content`	Literal "setting_get", "setting_set" or "setting_delete".
`status`	1 if the operation succeeded, 0 if it did not. If a non-existent key is requested, this will be set to 0.
`setting_key`	The name of the key that the operation was performed on.
`setting_value`	The **string** value of the key that the operation was performed on. This will be a null string if a non-existent key is requested. **Please note:** all numbers are returned as **stings**.
`timestamp`	Timestamp of when the operation was performed.

For `list`:

Property	Description

`content`	Literal "setting_list".
`count`	The number of settings returned.
`setting`	Container object holding the actual setting names and their valued. Each setting is a property of "`setting`". Eg: `myvariable = settingObject.setting.mysetting`
`timestamp`	Timestamp of when the list operation was performed.

4.3 "language"

`api?function=language&action=get!lng=<`*`language`*`>&module=<`*`module`*`>&`

Retrieve the content of the `language-XX.txt` file, where "XX" is the ISO language code specified by the `lng` parameter. If the specified language is not found, English will be returned. If the `lng` parameter is not specified, the language file for the OSD language will be returned. If the OSD language does not exist, English will be returned. If no suitable language can be found, a 404 error will be returned.

It is assumed that the language file is expressed in correctly formatted JSON and the entire contents are returned unaltered. If the required language file is absent, the contents of "`language-en.txt`" will be returned.

Please note: `language-XX.txt` is not returned as a HTML file, the contents of the file are returned as a JSON object.

The name of the OSD language file can be predicted by using the `language_code` property from the `config` function.

When the optional `module` parameter is provided, WebControl will return the language file from that web module using the language rules defined above.

4.4 "log"

`api?function=log&action=write!module=<`*`module`*`>&`

Write an entry to WebControl.log. The `module` will be written to the console only and defaults to "TMA1 Web" if omitted. The entire contents of the POST parameter are written to the log.

This API function will write to the log independent of the TAP logging level set. The current TAP logging level can be obtained from the `config` API command and should be used to determine whether to write a log entry or not.

4.5 "status"

`api?function=status&action=get!`

Response to the status functions consists of 3 areas:

1. General PVR status.
2. Current playback status.
3. Current recording status for each recording slot.

Property	Description
`content`	Literal "`status`".
`state`	The "`state`" returned from `TAP_GetState()` as per `TYPE_State`.
`substate`	The "`substate`" returned from `TAP_GetState()` as per `TYPE_SubState`.
`current_chtype`	The "`svcType`" of the current channel as per `TYPE_ServiceType`.
`current_chnum`	The "`svcNum`" of the current channel matching the "`chnum`" returned by the channels function.
`current_logoid`	The unique ID for the logo returned by the `LogoManager_GetChannelID()` function based on the "`svcnum`" and "`svctype`".
`totaldisk`	The total disk size in bytes returned from `TAP_Hdd_TotalSize()`.
`freedisk`	The free disk space in bytes returned from `TAP_Hdd_FreeSize()`.
`heapsize`	The memory heap size in bytes returned from `TAP_MemInfo()`.
`freeheapsize`	The free memory heap size in bytes returned from `TAP_MemInfo()`.
`availheapsize`	The available memory heap size in bytes returned from `TAP_MemInfo()`.
`drive_count`	The number of external media connected to the PVR.
`timestamp`	Timestamp of when the status was generated.

Property	Description
`playstatus`	The returned value from the `TAP_Hdd_GetPlayInfo()` call. If this value is not "1" then the API call failed and the returned data is unreliable.
`playmode`	The "`playMode`" returned from `TAP_Hdd_GetPlayInfo()` as per `TYPE_PlayMode`.
`trickmode`	The "`trickMode`" returned from `TAP_Hdd_GetPlayInfo()` as per `TYPE_TrickMode`.
`speed`	The "`speed`" returned from `TAP_Hdd_GetPlayInfo()`.
`svctype`	The "`svcType`" returned from `TAP_Hdd_GetPlayInfo()`as per `TYPE_ServiceType`.
`svcnum`	The "`svcNum`" returned from `TAP_Hdd_GetPlayInfo()`.
`logoid`	The unique ID for the logo returned by the `LogoManager_GetChannelID()` function based on the "`svcnum`" and "`svctype`".

Property	Description
duration	The "duration" returned from TAP_Hdd_GetPlayInfo(). This is the number of whole minutes contained in a recording.
durationsec	The "durationSec" returned from TAP_Hdd_GetPlayInfo(). This is the number of additional seconds in a recording. Total recording seconds = (duration x 60) + durationsec.
currentblock	The "currentBlock" returned from TAP_Hdd_GetPlayInfo(). This represents the current playback position within a recording in "blocks".
totalblock	The "totalBlock" returned from TAP_Hdd_GetPlayInfo(). This represents the total number of "blocks" in a recording.
filename	The "fileName" returned from TAP_Hdd_GetPlayInfo(). This is relative to the PVR's API path, "/DataFiles".
filename_hex	As per "filename", but without conversion to UFT8 or filtering for control characters. Represented in HEX for use with play/rename/delete/etc functions.
absolutefilename	As per "filename", but starting from the absolute Linux file path "/mnt/hd/DataFiles".

"recording" object with 1 element per recording slot. The number of recording slots can be determined by inspecting the "maxrecstreams" property in the "config" response.

Property	Description
slot	The slot number of the recording in progress. This value is constant throughout the duration of a recording. The first slot is "0".
rectype	The "recType" returned from TAP_Hdd_GetRecInfo(). As per TYPE_RecType.
tuner	The tuner number of the recording in progress. This value is constant throughout the duration of a recording. The first tuner is "0".
svctype	The "svcType" returned from TAP_Hdd_GetRecInfo()as per TYPE_ServiceType.
svcnum	The "svcNum" returned from TAP_Hdd_GetRecInfo().
logoid	The unique ID for the logo returned by the LogoManager_GetChannelID() function based on the "svcnum" and "svctype".
duration	The "duration" returned from TAP_Hdd_GetRecInfo(). This is the current scheduled recording duration. starttime + duration = endtime
starttime	The "startTime" returned from TAP_Hdd_GetRecInfo(). This is the time that the recording commenced.
endtime	The "endTime" returned from TAP_Hdd_GetRecInfo(). This is the current projected recording completion time assuming that it is not stopped early or extended.
recordedsec	The "recordedSec" returned from TAP_Hdd_GetRecInfo(). The total number of seconds currently recorded.
filename	The "fileName" returned from TAP_Hdd_GetRecInfo(). This is relative to the PVR's API path, "/DataFiles".

Property	Description
filename_hex	As per "filename", but without conversion to UFT8 or filtering for control characters. Represented in HEX for use with play/rename/delete/etc functions.

"drives" object with 1 element per mounted drive. The number of external drives can determined by inspecting the "drive_count" property in the "status" response.

Property	Description
name	The name assigned to the external media.
path	The path of the device.
mountpath	The mounted path to access the external media.
partitiontype	The type of the partition. 1 = jfs, 4 = FAT32, 5 = NTFS.
free_mb	The free space on the external media in megabytes.
size_mb	The size of the external media in megabytes.

4.6 "channels"

The "channels" function returns a list of channels configured on the PVR.

The channels function supports the "get" and "set" "action". When a command is issued with the set action, the chtype and chnum parameters must be provided.

Examples:

```
http://<PVR IP>:8000/api?function=channels&action=get!
http://<PVR IP>:8000/api?function=channels&action=set!chtype=0&chnum=12&
http://<PVR IP>:8000/api?function=channels&action=get!favourites=<group name>&
http://<PVR IP>:8000/api?function=channels&action=get!favourites_hex=<hex group name>&
```

1. chtype represents the channel type as defined in the Topfield TAP documentation where 0 = TV and 1 = Radio.
2. chnum represents the internal number of the channel required and has no relation to the LCN. The easies way to obtain the chnum is to issue a get for channels and process the channel list returned.

Response to the favourites function consists of 2 areas:

1. General channels information.
2. List of channels.

Property	Description
content	Literal "channels".
current_chtype	The "svcType" of the current channel as per TYPE_ServiceType.
current_chnum	The "svcNum" of the current channel matching the "chnum" returned by the channels function.

Property	Description
current_logoid	The unique ID for the logo returned by the LogoManager_GetChannelID() function based on the "svcnum" and "svctype".
favourites	"0" if all channels listed, "1" if the listing is for a specific favourites group.
fav_name	The name of the requested favourites group if applicable, otherwise null.
fav_name_hex	The name of the requested favourites group if applicable in hexadecimal notation, otherwise null.
channels	An array property containing the member services for each group.
timestamp	Timestamp of when the details of the channels listed.

For each of the "channels" property:

Property	Description
chnum	The "svcNum" for the channel.
chtype	The "svcType" value as per TYPE_ServiceType.
satindex	The "SatIndex" value returned by the FlashServiceGetInfo() function.
logoid	The unique ID for the logo returned by the LogoManager_GetChannelID() function based on the "chnum" and "chtype".
videostreamtype flagdelete flagcas flaglock flagskip tuner transponderindex serviceid pmtpid pcrpid videopid audiopid lcn audiostreamtype servicename providername namelock flags2 unknown2	The values of these remaining properties correspond to the elements in the tFlashService structure returned by the FlashServiceGetInfo() function based on the "chnum" and "chtype".

4.7 "favourites"

The "favourites" function returns a list of favourites groups configured on the PVR with the service details for each group member. The favourites function supports the "get" "action".

Examples:

http://<PVR IP>:8000/api?function=favourites&action=get!

Response to the favourites function consists of 3 areas:

1. General favourites information.
2. List of groups.
3. List of group members.

Property	Description
content	Literal "favourites".
totalgroups	The total number of configurable groups on the PVR.
groupsize	The maximum number of members that can be in any one group.
count	The number of defined groups on the PVR.
currentgroup	The name of the currently selected group on the PVR.
currentgroup_hex	The name of the currently selected group on the PVR in hexadecimal notation.
groups	An array property containing the groups defined on the PVR.
timestamp	Timestamp of when the favourites list was generated.

For each of the "groups" property:

Property	Description
name	The name of the group.
name_hex	The name of the group in hexadecimal notation.
count	The number of items in the group, this can be zero.
services	An array property containing the member services for each group.

For each of the "services" property:

Property	Description
svctype	The "svcType" returned as per TYPE_ServiceType.
svcnum	The "svcNum" matching the "chnum" returned by the channels function.
lcn	The logical channel number associated with the service.
logoid	The unique ID for the logo returned by the LogoManager_GetChannelID() function based on the "svcnum" and "svctype".
chname	The name of the service.

4.8 "timers"

The "timers" function has various actions for listing and manipulating timer reservations.

Actions

Action	Meaning	HTTP
get	Get a listing of the current timers.	GET
new	Create a new timer.	POST
create	Alias of **new**.	POST
modify	Modify an existing timer.	POST
update	Alias of **modify**.	POST
delete	Delete an existing timer.	POST

Please note: For modify and delete actions, the index, old_servicetype, old_serviceindex, old_duration and old_starttime parameters must be provided. These are used to validate that the correct index has been used because it is possible that a one-time timer has executed and deleted since the timer list was extracted, thus renumbering the timers and invalidating the indices.

Keywords for creating/editing/deleting timers:

Keyword	Value
isrec	0 = Play, 1 = Record.
tunerindex	Normally 3 for auto, 4 for analogue input.
servicetype	As per TYPE_ServiceType: 0 = TV, 1 = Radio. "chtype" from "channels" or "servicetype" from "timers".
serviceindex or servicenum	"chnum" from "channels" or "serviceindex" from "timers".
reservationtype	As per TYPE_ReservationType
nameset	The "nameset" flag from the timers flash area.
duration	Duration of timer in whole minutes.
starttime	Start time: yyyy/mm/dd HH:MM:00
filename	Name of recording, remember to add ".mpg" or ".rec". "recextension" from "config-get".
epgmarker	Icon to show in the EPG. 1 = Play, 2 = Record, 6=IR.
eventid1	Event ID from the EPG for this timer.
eventid2	Event ID from the EPG for this timer.
analoguetype	2 = Composite, 4 = Component.

---

index	Index number of existing timer to modify/delete.
old_servicetype	The service type, before the change, of the timer corresponding to "index". As per TYPE_ServiceType.

Keyword	Value
old_serviceindex or old_servicenum	The service number, before the change, of the timer corresponding to "index".
old_duration	The duration in whole minutes, before the change, of the timer corresponding to "duration".
old_starttime	The start time, before the change, of the timer corresponding to "index". yyyy/mm/dd HH:MM:00

Properties returned from a "get" action:

Property	Value
index	The index location for the timer within the array maintained by the PVR is flash memory.
tunerindex	Normally 3 for auto, 4 for analogue input.
recmode	recmode
demuxpath	demuxpath
manualrec	0 = Created from EPG, 1 = Manual.
unused1	Unused.
satindex	Satellite table index for this service.
servicetype	As per TYPE_ServiceType: 0 = TV, 1 = Radio. "chtype" from "channels" or "servicetype" from "timers".
reservationtype	As per TYPE_ReservationType
unused2	Unused.
serviceid	The serviceid from the channels listing.
lcn	Logical channel number.
logoid	The unique ID for the logo returned by the LogoManager_GetChannelID() function based on the "serviceindex" and "servicetype".
chname	Channel name.
satname	Satellite name.
orgnetid	Orginating network ID.
duration	Duration of timer in whole minutes.
unused3	Unused.
filename	Name of recording, sanitised for JSON use and converted to Unicode text.
filename_hex	Name of recording, as stored on the PVR in hexadecimal notation.
starttime	Start time: yyyy/mm/dd HH:MM:00
endtime	End time calculated from the start time plus the duration: yyyy/mm/dd HH:MM:00

Property	Value
endtime_raw	The end time contained within flash: yyyy/mm/dd HH:MM:00 (Not always accurate!)
pmtid	DVB Program Map Table identifier.
isrec	0 = Play, 1 = Record.
nameset	The "nameset" flag from the timers flash area.
unused4	Unused.
epgmarker	Icon to show in the EPG. 1 = Play, 2 = Record, 6=IR.
unused5	Unused.
unknown1	Unknown.
eventid1	Event ID from the EPG for this timer.
eventid2	Event ID from the EPG for this timer.
serviceindex	"chnum" from "channels" or "serviceindex" from "timers".
analoguetype	2 = Composite, 4 = Component.
unused8	Unused. (unused8[6] contains the analogue input type)
icetv	Flag for ICE TV.
unused9	Unused.

4.9 "files"

Actions

Action	Meaning	HTTP
`get`	Retrieve a list of recording files located on the PVR.	GET
`play`	Commence playback of a specified recording file.	POST
`delete`	Delete a specified recording file.	POST
`rename`	Rename a specified recording file.	POST
`stop / stopplay`	Stop playback of the current file and return to live TV.	GET
`stoprec`	Stop recording for the slot number provided.	GET
`tree`	Return a list of directories on the PVR.	GET
`jump / skip`	Move the current playback position.	GET
`pause`	Pause playback.	GET
`resume`	Resume playback	GET
`duration`	Change the duration of a recording-in-progress	GET

get

`http://<PVR IP>:8000/api?function=files&action=get!dir=<directory>&`

The "`get`" command returns a listing of file in the directory specified. If a `dir` parameter is provided, it should be expressed using the absolute Linux path "`/mnt/<path name>`". If no directory is specified "`/mnt/hd/DataFiles`" is assumed.

The "`get`" command returns only Topfield recordings (.mpg or .rec) when listing directories within the "`/mnt/hd/DataFiles`" structure, but all files when listing other locations.

play

`http://<PVR IP>:8000/api?function=files&action=play!`

The "`play`" command plays the video file described by the POST parameters that accompany the command.

`dir=`	The directory containing the file. It should point to a full Linux path and may include internal and external media locations. If a location is not provided, "`/DataFiles`" is assumed.
`file=`	The name of the file to be played.
`file_hex=`	Optional - A hexadecimal representation of the file name.
`block=`	Optional - The block number to jump to after playback has commenced.

Where the file name provided can be matched with a corresponding INF file in he directory specified, the file will be played as if it is a Topfield recording. If no INF file is found, the file will be played as a "Media File".

delete

http://<PVR IP>:8000/api?function=files&action=delete!

The "delete" command deletes a file on the PVR. The following POST parameters are required:

dir= / old_dir	The directory containing the file. It should point to a full Linux path and may include internal and external media locations. If a location is not provided, "/DataFiles" is assumed.
file= / old_file=	The name of the file to be deleted.
file_hex= / old_file_hex=	Optional - A hexadecimal representation of the file name.

When deleting a file, WebControl will always try and delete the file requested as well as the INF an NAV of the associated file.

rename

http://<PVR IP>:8000/api?function=files&action=rename!

The "rename" command renamed a file on the PVR. The following POST parameters are required:

dir= / old_dir	The directory containing the file. It should point to a full Linux path and may include internal and external media locations. If a location is not provided, "/DataFiles" is assumed.
file= / old_file=	The name of the file to be renamed.
file_hex= / old_file_hex=	Optional - A hexadecimal representation of the file name.
new_dir=	The new directory for the file to be renamed. It should point to a full Linux path and may include internal and external media locations. If a location is not provided, "/DataFiles" is assumed. It can be he same as the existing directory to rename a file in place.
new_file=	The new name of the file to be renamed. It can be the same if the file is to be moved to another directory.

The "rename" function can be used to rename a file in place as well as move a file to another folder. The directory parameters must be in full Linux notation.

WARNING: TMA1 will permit renaming across mounted file systems. A file of a sufficiently large size moved to an external media, for example, will take some time to complete. This may result in the PVR freezing or other abnormal behaviour.

stop

http://<PVR IP>:8000/api?function=files&action=stop!

The "stop" command stops playback of the current file and returns to live programming.

stoprec

`http://<PVR IP>:8000/api?function=files&action=stoprec!slot=<recording slot>&`

The "`stoprec`" command stops the recording currently in progress for the slot number provided. The number of slots can be determined by the "`maxrecstreams`" property from the "`config`" response, and the specific slot for a recording in progress can be determined from the "`status`" response.

Return Codes

0 Failure
1 Success
99 Invalid slot number

tree

`http://<PVR IP>:8000/api?function=files&action=tree!dir=<directory>&`

The "`tree`" command returns a listing of directories commencing from `/mnt/hd/DataFiles` if no `dir` parameter is provided. If a `dir` parameter is provided, it should be expressed using the absolute Linux path "`/mnt/<path name>`". To obtain the tree of an external device, use the external device name, for example "`/mnt/sdb1/`".

Property	Description
`content`	Literal "files_tree".
`directory`	An array object containing multiple "`name`" objects representing the directories returned.
`count`	A count of directories returned.
`timestamp`	Timestamp of when the operation was performed.

jump / skip

http://<PVR IP>:8000/api?function=files&action=jump!block=+100&
http://<PVR IP>:8000/api?function=files&action=jump!time=-300&

The "jump" command moves the current playback position by specifying either an amount of time (in whole seconds) or a number of whole blocks.

In addition to this, the direction can also be controlled by prefixing the time/blocks with a direction indicator as follows: "+" move forward; "-" move backwards; "=" move to the location specified.

Examples

!block==1234&	Jump to block 1234.
!time=-30&	Jump back approximately 30 seconds.
!block=+1000&	Jump forward 1000 blocks.

It is also possible to jump to a specific percent mark within the playback file. Simply get the "totalblock" from the "status", calculate the block location for the percent jump required and then issue a "jump" with "block==<value>".

If playback is paused when a "jump" occurs, WebControl will automatically resume playback. To prevent this from occurring, include the "noresume" parameter with the command. For example "!time=+300&noresume=1".

Normally, the progress bar will be hidden. If the "nohide" parameter is present, the progress bar will not be hidden.

Error Codes

0	No error
1	TAP_Hdd_GetPlayInfo() failed
2	Not playing
3	Invalid jump value
4	Attempt to jump past end
5	TAP_Hdd_ChangePlaybackPos() failed
6	totalBlock = 0

pause

http://<PVR IP>:8000/api?function=files&action=pause! &

The "pause" command causes the current playback file to be paused.

If playback is already paused, this command has no effect.

This function uses API functions to pause playback and does not generate remote control keys.

resume

http://<PVR IP>:8000/api?function=files&action=resume! &

The "resume" command causes the current paused playback file to be resumed.

If playback is already playing, this command has no effect.

This function uses API functions to pause playback and does not generate remote control keys.

duration

http://<PVR IP>:8000/api?function=files&action=duration!slot=2&duration=120&

The "duration" command alters the duration of a recording-in-progress.

slot	The recording slot to change. This value can be obtained from the "recording" objects returned by the "status" command.
duration	The new total duration of the recording replacing the existing recording duration. No validation is performed on the new duration. Setting the new duration to a value less than the current elapsed recording time will cause the recording to stop.

4.10 "epg"

The "epg" function returns a list of programme events for a specified service between specified start and end times. The epg function only supports the "get" "action".

Example:

http://<PVR IP>:8000/api?function=epg&action=get!
chtype=0&chnum=12&starttime=yyyy/mm/dd HH:MM:SS&endtime=yyyy/mm/dd&epgsource=<source>&

1. chtype represents the channel type as defined in the Topfield TAP documentation where 0 = TV and 1 = Radio.
2. chnum represents the internal number of the channel required.
3. starttime represents the earliest event to return in PVR local time.
4. endtime represents the latest event to return in PVR local time.
5. source represents the EPG source, absent or "default", "pvr" or "api" return EPG data according to the TAP configuration setting. "smartepg" will return EPG data from the SmartEPG database if SmartEPG is running, otherwise EPG data will be returned from the firmware EPG source.

EPG events will be returned where the event start time is less than or equal to the endtime parameter **AND** where the event end time is greater than or equal to the startime parameter. In this way, the API returns all of the events, full and partial, falling within the bounds of the startime and endtime parameters.

WebControl uses the FireBird EPG libraries to access EPG data. These libraries automatically compensate for daylight savings times and transition periods.

Response to the EPG function consists of 2 areas:

1. General information relating to the request.
2. List of events.

Property	Description
content	Literal "epg_events".
requested_chtype	The "svcType" of the requested channel as per TYPE_ServiceType.
requested_chnum	The "svcNum" of the requested channel matching the "chnum" returned by the channels function.
requested_logoid	The unique ID for the logo returned by the LogoManager_GetChannelID() function based on the "svcnum" and "svctype".
requested_starttime	The requested start time.
requested_endtime	The requested end time.
requested_epgsource	The EPG source present in the request. "none" if no source was present.
epgsource	The actual source of the EPG data: "pvr" or "smartepg".
event	An array property containing the EPG events.
count	The number of EPG events returned.

Property	Description
`error`	True if a memory allocation error occurs within WebControl. Try reducing the timer range of the EPG search.
`timestamp`	Timestamp of when the details of the channels listed.

Generally speaking, the contents of the elements in the event property reflect the lower-case equivalent of those in the `TYPE_EPGInfo` structure described in section 6.13. However, there are some minor name changes and additions.

Property	Description
`starttime_local` `endtime_local`	These fields were renamed to emphasise that the dates and times returned in these fields are expressed in the local time of the PVR an not UTC.
`parental_text`	This represents the PVR's language-appropriate description of the parent rating code provided in the `parental` field.
`Genre_text`	This represents the PVR's language-appropriate description of the content identifier code provided in the `contentidentifier` field.

4.11 "remote"

The "remote" function generates a keystroke as if a key the remote control unit had been pressed.

http://<PVR IP>:8000/api?function=remote&action=send!key=<key name>

The only action supported by the remote function is send. Send takes one parameter key. key should be set to the name of the key to be pressed. The list of valid key names is as follows:

Code / Alias	RCU Code
menu	RKEY_Menu
exit	RKEY_Exit
ok / enter	RKEY_Ok
f1 / red	RKEY_NewF1
f2 /green	RKEY_F2
f3 / yellow	RKEY_F3
f4 / blue	RKEY_F4
chup	RKEY_ChUp
chdown	RKEY_ChDown
volup	RKEY_VolUp
voldown	RKEY_VolDown
vformat	RKEY_VFormat
up	RKEY_Up
down	RKEY_Down
left	RKEY_Left
right	RKEY_Right
guide	RKEY_Guide
info	RKEY_Info
playlist	RKEY_PlayList
recall	RKEY_Recall
play	RKEY_Play
pause	RKEY_Pause
stop	RKEY_Stop
record / rec	RKEY_Record
next	RKEY_Next
prev	RKEY_Prev
ab	RKEY_Ab
mute	RKEY_Mute

Code / Alias	RCU Code
uhf	RKEY_Uhf
sleep	RKEY_Sleep
easy	RKEY_Easy
audiotrk / audiotrack	RKEY_AudioTrk
tvradio	RKEY_TvRadio
fav	RKEY_Fav
subt / subtitle	RKEY_Subt
tvsat	RKEY_TvSat
teletext / ttx	RKEY_Teletext
rewind / rew	RKEY_Rewind
forward / fwd	RKEY_Forward
sat	RKEY_Sat
slow	RKEY_Slow
0	RKEY_0
1	RKEY_1
2	RKEY_2
3	RKEY_3
4	RKEY_4
5	RKEY_5
6	RKEY_6
7	RKEY_7
8	RKEY_8
9	RKEY_9
m	RKEY_M
power	RKEY_Power
option	RKEY_Option

Key names can be modified with a "long-" prefix, for example, "long-info". This will produce a press-hold-release sequence for the nominated key.

A standard key press sequence for the "Menu" key (0x1001c) generates:

1. Press 0x0001001c
2. Release 0x0201001c

Whereas a press-hold-release sequence generates the following key codes:

1. Press 0x0001001c
2. Hold 0x0101001c
3. Release 0x0201001c

4.12 "ascii"

The "`ascii`" function generates a keystroke as if a character had been typed using TMSRemote in "direct" mode.

`http://<PVR IP>:8000/api?function=ascii&action=send!char=<ascii code>&`

The only `action` supported by the `ascii` function is `send`. Send takes one parameter being the ASCII value of the character to generate.

4.13 "tap"

The "`tap`" function returns a JSON object containing a list of running TAPs.

`http://<PVR IP>:8000/api?function=tap&action=get!`

This function is provided to enable the browser to detect what other TAPs are running so that it can initiate communications with other TAPs if necessary.

External TAPs contacted by WebControl must be modified to receive and decode the parameters passed to it by WebControl.

4.14 "vol"

The "`vol`" function returns and optionally adjusts the volume on the PVR.

`http://<PVR IP>:8000/api?function=vol&action=get!`
`http://<PVR IP>:8000/api?function=vol&action=set!&level=[0-17]|mute|unmute`

Actions

Action	Meaning	Content Property
`get`	Retrieve a single front-end setting value.	setting_get
`set`	Save a new or update an existing single front-end setting value.	setting_set

For `get`:

Property	Description
`content`	Literal "volume".
`level`	Current volume level from 0 - 17.
`mute`	Boolean mute state.
`timestamp`	Timestamp of when the operation was performed.

For `set`:

Property	Description
content	Literal "volume_change".
level_before	Volume level prior to implementing the change.
mute_before	Boolean mute state prior to implementing the change.
level_before	Volume level after implementing the change.
mute_before	Boolean mute state after implementing the change.
timestamp	Timestamp of when the operation was performed.

Please note: There are currently know issues muting and un-muting.

4.15 "logo"

The "logo" function returns the JSON description for the channel logos and requests that the logos sprite file be rebuilt.

```
http://<PVR IP>:8000/api?function=logo&action=get!
http://<PVR IP>:8000/api?function=logo&action=rebuild!
```

Actions

Action	Meaning
get	Return a JSON object describing the logo location for each channel within the logos sprite service-logos.png.
rebuild	Request WebControl to rebuild the logos file. This action is performed asynchronously and the returned JSON is an acknowledgement of the request not acknowledgement that the logos have been rebuilt. Check the date stamp from the get command because this contains the date that the logos were built. Logos will not be rebuilt if noautologos=1 in WebControl.ini.

For get:

Property	Description
content	Literal "logos".
logos	An array property containing the logo details.
timestamp	Timestamp of when the operation was performed.

Each `logos` element contains the following properties:

Property	Description
`logoid`	The "`svcNum`" matching the "`chnum`" returned by the channels function.
`svctype`	The "`svcType`" returned as per `TYPE_ServiceType`.
`svcnum`	The unique ID for the logo returned by the `LogoManager_GetChannelID()` function based on the "`svcnum`" and "`svctype`".
`sprite_x`	The X coordinate of the upper left corner of the logo image within the log sprite image.
`sprite_y`	The Y coordinate of the upper left corner of the logo image within the log sprite image.

For `rebuild`:

Property	Description
`Content`	Literal "logo_rebuild_requested".
`timestamp`	Timestamp of when the acknowledgement was sent.

5. Communicating with other TAPs

The TMA1 protocol provides the ability to pass commands received via the API from a web browser to other TAPs and forward any response back to the web browser.

When WebControl communicates with another TAP, the called TAP will receive an `EVT_TMA1` via the `TAP_EventHandler()` function with `param1` containing a pointer to the following structure:

```
typedef struct TYPE_TMA1_COMMAND {
    dword           rsvp;               //Requesting TAP.
    int             status;             //Set to zero when calling, called TAP must set
                                        //this value or the response is ignored.
    char            *cFunction;         //Pointer to the parsed "function".
    char            *cAction;           //Pointer to the parsed "action".
    char            *cTAP;              //Pointer to the parsed "tap".
    char            *cSession;          //Pointer to the parsed "session".
    char            *params;            //Unparsed remaining parameters.
    char            *post;              //Unparsed contents of the HTTP POST command.
    int             postLen;            //Length of the HTTP POST command.
    int             retStatus;          //A return code from the TAP with more information.
    char            *response;          //Called TAP must MALLOC this pointer.
    unsigned int    responseLen;        //Number of bytes in the response to be sent to
                                        //the browser.
    char            responseType[128];  //Controls the "Content-Type:" in the HTTP
                                        //response header.
} TYPE_TMA1_COMMAND;
```

The sample TAP `TMA1-Comms` demonstrates the use of this feature.

Element	Description
`rsvp`	This is the TAP_ID of the calling TAP, this will normally be WebControl, but other TAPs may use this feature too.
`status`	This element should contain either `TMA1_STATUS_COMMAND` or `TMA1_STATUS_MEMFREE` when WebControl calls the called TAP. The called TAP is then expected to set this element to either `TMA1_STATUS_RESPONSE` or `TMA1_STATUS_ERROR`. Failure to set a response or error status will result in WebControl ignoring the response.
`*cFunction`	Pointer to the parsed "`function`" sent by the browser to WebControl.
`*cAction`	Pointer to the parsed "`action`" sent by the browser to WebControl.
`*cTAP`	Pointer to the parsed "`tap`" sent by the browser to WebControl.
`*cSession`	Pointer to the parsed "`session`" sent by the browser to WebControl.
`*params`	Pointer to the unparsed remaining parameters sent by the browser to WebControl.
`*post`	Pointer to the unparsed POST command sent by the browser to WebControl.
`postLen`	Length of the POST command.
`retStatus`	Return status from the command. This value is passed to the web browser unaltered if a failure occurs.
`*response`	WebControl passes an uninitialised pointer for the JSON response. The called TAP should MALLOC this pointer before copying any data to it.
`responseLen`	The called TAP is expected to set this element to the length of the response data.
`responseType`	This is a 128 byte string equating to the "Content-Type:" in the HTTP response and under most circumstances should be set to "`application/json`".

```
#define TMA1_STATUS_COMMAND     0
#define TMA1_STATUS_RESPONSE    1
#define TMA1_STATUS_ERROR       2
#define TMA1_STATUS_MEMFREE     9999
```

Provided that the `status` element is set to `TMA1_STATUS_RESPONSE`, WebControl will return the entire contents pointed to by the `response` element to the browser unaltered. It is critical that the returned data be in correct JSON format (please see the note on date formatting in section 3). A small library of functions is available in `tma1-helpers` to assist with this process.

If the called TAP receives a command with the `status` element set to `TMA1_STATUS_MEMFREE`, the called TAP should free the memory at the address in the pointer `response` and return control to WebControl immediately. Failure to do so may result in a memory leak.

It is important that the called TAP respond before any other "events" are permitted to occur. If the called TAP needs to wait for remote control events, or any other event, control will return to WebControl and the call will effectively fail.

6. Appendix A - Useful Constants, Structures and Enumerated Lists

6.1 TYPE_ServiceType

```
typedef enum
{
    SVC_TYPE_Tv,
    SVC_TYPE_Radio,
} TYPE_ServiceType;
```

6.2 REMOTE_TYPE

```
typedef enum
  {
    RT_5000,
    RT_2100,
    RT_7100PLUS
    RT_7260PLUS        //Identical to RT_2100, except that the code for the red key is RKEY_F1
  } REMOTE_TYPE;
```

6.3 TYPE_State

```
typedef enum
{
    STATE_Normal,
    STATE_Menu,
    STATE_Epg,
    STATE_List,
    STATE_Ttx,
    STATE_Game,
    STATE_FileList,
    STATE_Tap,
} TYPE_State;
```

6.4 tFlashService

```
typedef struct
  {
    byte                SatIndex;
    byte                VideoStreamType;
    bool                FlagDelete;
    bool                FlagCAS;
    bool                FlagLock;
    bool                FlagSkip;
    byte                Tuner;
    word                TransponderIndex;
    word                ServiceID;
    word                PMTPID;
    word                PCRPID;
    word                VideoPID;
    word                AudioPID;
    word                LCN;
    word                AudioStreamType;
    char                ServiceName[MAX_SvcName];
    char                ProviderName[40];
    byte                NameLock;
    word                Flags2;
    byte                unknown2[6];
  } tFlashService;
```

6.5 TYPE_SubState

```
typedef enum
{
    SUBSTATE_MainMenu,
    SUBSTATE_TimeMenu,
    SUBSTATE_TimeSettingMenu,
    SUBSTATE_InstallationMenu,
    SUBSTATE_SysRecoverMenu,
    SUBSTATE_FirmUpgradeMenu,
    SUBSTATE_TransferMenu,
    SUBSTATE_TimerMenu,
    SUBSTATE_LanguageMenu,
    SUBSTATE_RecordingSetMenu,
    SUBSTATE_PlaybackMenu,
    SUBSTATE_OsdSettingMenu,
    SUBSTATE_ParentMenu,
    SUBSTATE_ParentLockMenu,
    SUBSTATE_AvMenu,
    SUBSTATE_OtherSettingMenu,
    SUBSTATE_EditServiceListMenu,
    SUBSTATE_EditFavoriteListMenu,
    SUBSTATE_SearchMenu,
    SUBSTATE_SearchModeMenu,
    SUBSTATE_LnbSettingMenu,
    SUBSTATE_Diseqc12SettingMenu,
    SUBSTATE_USALSMenu,
    SUBSTATE_CiMenu,
    SUBSTATE_Ci,
    SUBSTATE_CiSlotMenu,
    SUBSTATE_SatSearch,
    SUBSTATE_TpSearch,
    SUBSTATE_QuickSearch,
    SUBSTATE_SmatvSearch,
    SUBSTATE_FastScan,
    SUBSTATE_Normal,
    SUBSTATE_ServiceList,
    SUBSTATE_Epg,
    SUBSTATE_Fav,
    SUBSTATE_Sat,
    SUBSTATE_Audio,
    SUBSTATE_Subt,
    SUBSTATE_Ttx,
    SUBSTATE_SystemStatus,
    SUBSTATE_InformationMenu,
    SUBSTATE_SettingMenu,
    SUBSTATE_GameMenu,
    SUBSTATE_CanalEpgMenu,
    SUBSTATE_Game,
    SUBSTATE_AntenaSettingMenu,
    SUBSTATE_PositionSettingMenu,
    SUBSTATE_PvrList,
    SUBSTATE_PvrRecord,
    SUBSTATE_PvrTimeSearch,
    SUBSTATE_PvrPlayingSearch,
    SUBSTATE_PvrRecSearch,
    SUBSTATE_Exec,
    SUBSTATE_PipList,
    SUBSTATE_EditServicesMenu,
    SUBSTATE_TtxEmul,
    SUBSTATE_ConaxMenu,
    SUBSTATE_NorConax,
    SUBSTATE_ConaxSubscription,
```

```
    SUBSTATE_EventSts,
    SUBSTATE_ConaxCaPin,
    SUBSTATE_ConaxSignPin,
    SUBSTATE_MatRating,
    SUBSTATE_CaInfo,
    SUBSTATE_RateLock,
    SUBSTATE_TokenSts,
    SUBSTATE_DebitSts,
    SUBSTATE_CreditSts,
    SUBSTATE_TokenPPV,
    SUBSTATE_MailBox,
    SUBSTATE_OtaFirmUpMenu,
    SUBSTATE_NetSettingMenu,
    SUBSTATE_IPSettingMenu,
    SUBSTATE_NetFirmUpMenu,
    SUBSTATE_LANMenu,
    SUBSTATE_LANStatusMenu,
    SUBSTATE_LANIPConfigMenu,
    SUBSTATE_ServiceCopyMenu,
    SUBSTATE_Multifeed,
    SUBSTATE_NVOD,
    SUBSTATE_GetBer,
    SUBSTATE_SimpleLang,
    SUBSTATE_SSUMenu,
    SUBSTATE_CheckSSU,
    SUBSTATE_TimerModify,
    SUBSTATE_CanalEpgRetrieve,
    SUBSTATE_QuickRecWin,
    SUBSTATE_PinCode,
    SUBSTATE_EXTSignalBar,
    SUBSTATE_DelTimerEntry,
    SUBSTATE_SysMsg,
    SUBSTATE_SysConfirmMsg,
    SUBSTATE_StopRecordMsg,
    SUBSTATE_ExtEvtInfo,
    SUBSTATE_SatTpEditMenu,
    SUBSTATE_SatelliteSettingMenu,
    SUBSTATE_DCMSearch,
    SUBSTATE_RecordingMenu,
    SUBSTATE_EntertainmentMenu,
    SUBSTATE_FrontDisplaySettingMenu,
    SUBSTATE_PhotoAlbumList,
    SUBSTATE_UCCList,
    SUBSTATE_SnapShot,
    SUBSTATE_Weather,
    SUBSTATE_Flickr,
    SUBSTATE_EpgSetting,
    N_SUBSTATE
} TYPE_SubState;
```

6.6 TYPE_PlayInfo

```
typedef struct
{
    byte                playMode;   // Playback Mode, refer TYPE_PlayMode
    byte                trickMode;  // Trick Mode, refer TYPE_TrickMode
    byte                speed;      // playback speed
    byte                svcType;    // 0 = TV, 1 = Radio
    word                svcNum;
    word                duration;
    byte                durationSec;
    byte                reserved;
    dword               currentBlock;
```

```
    dword               totalBlock;
    TYPE_File           *file;
    TYPE_TapEvent       evtInfo;
    byte                repeatStatus;
    byte                reserved2[3];
} TYPE_PlayInfo;
```

6.7 TYPE_RecType

```
typedef enum
{
    RECTYPE_None,               // No Recording
    RECTYPE_Normal,             // Normal Recording
    RECTYPE_Timeshift,          // Recording for Time shifting ( Cannot stop )
    RECTYPE_Copy                // Copy
} TYPE_RecType;
```

6.8 TYPE_PlayMode

```
typedef enum
{
    PLAYMODE_None,              // No playback started
    PLAYMODE_Playing = 2,       // Normal Playback
    PLAYMODE_TempPlaying,       // Time shifting
    PLAYMODE_RecPlaying,        // Time shifting in recording service.
    PLAYMODE_Mp3,               // MP3
    N_PLAYMODE
} TYPE_PlayMode;
```

NOTE Play Mode 8 appears to be a MediaFile (MP4)
MP3 = mode 5.

6.9 TYPE_TrickMode

```
typedef enum
{
    TRICKMODE_Normal,           // Normal Playback
    TRICKMODE_Forward,          // Fast Forward
    TRICKMODE_Rewind,           // Rewind
    TRICKMODE_Slow,             // Slow Motion
    TRICKMODE_Pause             // Paused
} TYPE_TrickMode;
```

6.10 SYSTEM_TYPE

```
  typedef enum
  {
    ST_UNKNOWN,
    ST_S,
    ST_T,
    ST_C,
    ST_T5700,
    ST_TMSS,
    ST_TMST,
    ST_TMSC,
    ST_T5800,
    ST_ST,
    ST_CT,
    ST_TF7k7HDPVR,
    ST_NRTYPES
  } SYSTEM_TYPE;
```

6.11 tRunningStatus

```
typedef enum
  {
    RST_undefined,
    RST_NotRunning,
    RST_StartsSoon,
    RST_Pausing,
    RST_Running,
    RST_ServiceOffAir,
    RST_reserved1,
    RST_reserved2
  }tRunningStatus;
```

6.12 tFlashTimer

```
  typedef struct
  {
    byte                TunerIndex;
    byte                RecMode;
    byte                DemuxPath;
    byte                ManualRec;
    byte                unused1;
    byte                SatIndex;
    byte                ServiceType;
    byte                ReservationType;
    byte                unused2;
    word                ServiceID;
    word                Duration;
    byte                unused3;
    char                FileName[131];
    dword               StartTime;
    dword               EndTime;
    word                PMTPID;
    byte                isRec;
    byte                NameSet;
    byte                unused4;
    byte                EPGMarker;
    word                unused5;
    dword               unknown1;
    dword               EventID1;
    dword               EventID2;
    word                ServiceIndex;
    byte                unused8[8];
    byte                IceTV;
    byte                unused9[13];
    tFlashTransponderTable    TpInfo;
  }tFlashTimer;
```

6.13 EPGinfo

```
typedef struct
{
  word                EventID;
  byte                DataStatus;
  tRunningStatus      RunningStatus;

  dword               StartTime;              //Local start time
  dword               EndTime;                //Local end time
  short               TimeZone;               //Offset from UTC in minutes
  word                duration;               //duration in minutes

  word                SatIndex;
  word                NetworkID;
  word                TSID;
  word                ServiceID;

  TYPE_ServiceType    ServiceType;
  int                 ServiceNum;

  byte                lang[4];

  byte                ParentalRate;
  byte                NameLength;
  byte                ShortEventTextLength;
  word                ExtEventTextLength;

  char                EventName[256];
  char                ShortEventText[256];
  char                ExtEventText[8192];

  dword               citID;

  byte                ContentIdentifier;      //if sourceFlag == 1 then the
                                              //nibbles are reversed to ETSI and
                                              //if sourceFlag == 0 then
  byte                sourceFlag;             //the nibble represent
                                              //Australia-specific descriptions
  word                unknown14;
  word                iceChannel;

  byte                unknown15[6];
} TYPE_EPGInfo;
```