Introduction

EPGInfo is a suite of functions available within the libFireBird library that provide access to the EPG data stored within the RAM of the PVR.

At the time of writing, there are three known internal memory structures used by various firmware revisions on various PVR models globally. The EPGInfo functions are aware of these internal differences and, for these three known structures, compensate accordingly presenting a standard interface to all TAPs. **Warning:** there is a chance that other undocumented internal structures exist and that the EPGInfo functions will not function correctly with PVR models utilising those structures.

A TAP demonstrating the use of these functions can be found in the _Demo\EPG folder within the libFireBird distribution ZIP file.

EPG Navigation Functions

EPGInfo_FindCurrent

```
bool EPGInfo_FindCurrent(byte SvcType, int SvcNum,
                         TYPE_EPGInfo *EPGData)
```

This function sets EPGData to the event currently playing on the service indicated by SvcType and SvcNum.

This function returns FALSE if no information is available for the selected SvcType / SvcNum combination.

EPGInfo_FindFirst

```
int EPGInfo_FindFirst(TYPE_EPGInfo *EPGData)
```

This function extracts the required subset of EPG data and sets EPGData to the first EPG entry in the data set. This function also returns the number of EPG records extracted so that a TAP can read through them in a loop.

Please note that prior to calling EPGInfo_FindFirst, the TAP writer is advised to use one or more of the helper functions to refine the search parameters limiting the amount of EPG data returned. Chief among these would be one of the filter functions plus the DST function.

EPGInfo_FindNext

```
bool EPGInfo_FindNext(TYPE_EPGInfo *EPGData)
```

This function sets `EPGData` to the next available EPG event and increments the internal event index by one.

This function returns `FALSE` if an attempt is made to read past the last EPG event.

EPGInfo_FindPrev

```
bool EPGInfo_FindPrev(TYPE_EPGInfo *EPGData)
```

This function sets `EPGData` to the previous available EPG event and decrements the internal event index by one.

This function returns `FALSE` if an attempt is made to read past the first EPG event.

EPGInfo_FindLast

```
bool EPGInfo_FindLast(TYPE_EPGInfo *EPGData)
```

This function sets `EPGData` to the last EPG event in the data set. It also resets the internal EPG event index to be the last event in the data set for subsequent `EPGInfo_FindNext` operations.

Naturally, calling `EPGInfo_FindNext` immediately after `EPGInfo_FindLast` will return `FALSE` because of a bounds violation.

EPGInfo_FindItem

```
bool EPGInfo_FindItem(TYPE_EPGInfo *EPGData,
                      int EPGIndex, bool EPGReset)
```

This function sets `EPGData` to an arbitrary EPG event based on the value of `EPGIndex`. If `EPGReset` is `TRUE`, the current position within the EPG data set will be reset for subsequent `EPGInfo_FindNext`/`EPGInfo_FindPrev` operations.

This function may be useful for referring back to the EPG data set once a sort or additional filter operation has been performed on an external data set. This saves having to sort the entire EPG data set.

Note: The EPG item index specified by `EPGIndex` uses zero-based numbering.

This function returns `FALSE` if `EPGIndex` is out of bounds.

EPG Filter Functions

There are a number of optional cumulative filters that can be applied before an `EPGInfo_FindFirst` call is made. These filters reduce the number of EPG events returned, speeding up the EPG data extraction process and reducing the programming effort for subsequent filtering operations.

EPGInfo_FilterReset

```
void EPGInfo_FilterReset(void)
```

This function clears all previously set filers.

EPGInfo_FilterTime

```
void EPGInfo_FilterTime(dword StartTime, dword EndTime)
```

This function sets a filter that will cause `EPGInfo_FindFirst`, when executed, to return all EPG events that start on or after the `StartTime` and end on or before the `EndTime`. Setting either parameter to 0 will result in the other parameter being used as a single filter.

`StartTime` and `EndTime` consist of both date and time sub-components and are expressed in local time.

FilterChannelByIndex

```
void EPGInfo_FilterChannelByIndex(TYPE_ServiceType
                                  SvcType, int SvcNum)
```

This function sets a filter that will cause `EPGInfo_FindFirst`, when executed, to return the EPG events for a specific service only.

EPGInfo_FilterDuration

```
void EPGInfo_FilterDuration(word MinDuration,
                            word MaxDuration)
```

This function sets a filter that will cause `EPGInfo_FindFirst`, when executed, to return EPG events that are between or equal to the `MinDuration` and `MaxDuration` parameters. Setting either parameter to 0 will result in the other parameter being used as a single filter.

Both parameters are expressed in whole minutes.

EPGInfo_FilterGenre

```
void EPGInfo_FilterGenre(byte *GenreArray,
                         byte GenreArraySize)
```

This function sets a filter that will cause `EPGInfo_FindFirst`, when executed, to return EPG events for one or more programme Genres or "Content Identifiers". Calling this function multiple times adds multiple genres to the filter. A list of content identifiers can be found in the ETSI EN 300 468 documentation.

Warning: This filter may not work on all firmware.

It should be noted that not all firmware versions in all regions recognise and store the content identifiers that are broadcast. On firmware versions that do not recognise and store the content identifier, the content identifier for each EPG event will be set to `0x00`, filtering on any Genre value in this case would be expected to return a null data set.

EPGInfo_FilterCallback

```
void EPGInfo_FilterCallback(void *CallbackRoutine)
```

This function sets a custom filter that allows a TAP writer to customise the EPG events that will be included in the return data set when `EPGInfo_FindFirst` is executed.

`CallbackRoutine` is a pointer to a function that accepts a `TYPE_EPGInfo` structure as the sole parameter and returns `TRUE` if the event is to be included in the returned data set. The callback is only executed if all other filters return `TRUE`. An example of such a function is included below.

CallbackFunction

```
bool CallbackFunction(TYPE_EPGInfo *EPGInfo)
```

This is a sample function and not actually found within the library. Use a function like this in conjunction with `EPGInfo_FilterCallback` to facilitate a customised filter.

EPGInfo_AbortLoad

```
void EPGInfo_AbortLoad(void)
```

This function is intended for use within a `CallbackFunction`. As the EPG events are presented to `CallbackFunction` in chronological order, `EPGInfo_AbortLoad` provides a means of stopping further EPG entries from being loaded should a target date/time be met by `CallbackFunction`.

EPG Helper Functions

EPGInfo_Free

```
void EPGInfo_Free(void)
```

After having read and processed the required EPG events, `EPGInfo_Free` should be called to free the resources used for caching the required EPG data set.

DST_SetDSTRule

```
void EPGInfo_SetDSTRule(tDSTRule NewDSTRule)
```

This function should be called before accessing the `EPGInfo` functions for the first time. It is used to instruct the library how to convert EPG events, stored by the PVR in UTC, to local time and to convert time filters to UTC. There are three possible values for `NewDSTRule`:

1. `DSTR_Firmware` Use the UTC offset maintained by the firmware.
2. `DSTR_Europe` Use hard-coded UTC offset rules for Western Europe commencing DST on "Sunday in March, 02:00 local" and ending DST on "Sunday in October, 02:00 local".
3. `DSTR_Manual` Read the "`DSTStartRule`" and "`DSTEndRule`" from "`/ProgramFiles/Settings/DST.ini`" to determine the DST offset.

The default value is `DSTR_Firmware`.

Where a rule in `DST.ini` is used, it must be expressed in local time and be in the format: "`Ordinal,Day,Month,HH:MM`" where:

Parameter	Description
Ordinal	1 - 5, representing the first to fifth occurrence of the specific "Day" within the target "Month". A value of 5 also represents the last occurrence in a month.
Day	0 - 6, day of the week on which the DST transition is to occur. 0 = Monday, 6 = Sunday. These values are the same as the "weekday" returned by `TAP_ExtractMjd`.
Month	1 - 12, the month of the current calendar year in which the DST transition is to occur.
HH	The hour that the DST transition is to occur.
MM	The minute that the DST transition is to occur.

For example, the rule for "Last Sunday in March 02:00" would be "5,6,3,02:00". If a returned date would be in the past, the function recalculates for the next calendar year.

For rules other than "`DSTR_Firmware`", this function is most useful when accessing EPG events in the period immediately before a DST transition when relying on the firmware to provide the DST offset may be inaccurate.

For example, if you receive 7 days of EPG data, and you are within 3 days of a DST transition, the first 3 days' of EPG data will be converted with the correct UTC offset, however, the 4 days after the transition will be 1 hour out. Using a European or Custom rule eliminates this firmware limitation.

DST_CalcTransition

```
dword DST_CalcTransition(byte ruleOrdinal, byte ruleDay,
                         byte ruleMonth, byte ruleHour,
                         byte ruleMin);
```

Although specifically used by the library to calculate DST transition dates internally, the `DST_CalcTransition` function may be useful for calculating the specific occurrence of a specific day in a specific month in the current calendar year.

EPGInfo_GetNrFreeEntries

```
dword EPGInfo_GetNrFreeEntries(void)
```

This function returns the number of free entries in the EPG storage pool. Monitoring the number of free entries in the pool is a reasonable indicator of the stability of the quantity of EPG entries received from an individual transponder.

For example, when switching to a new transponder after a prolonged absence, it is reasonable to expect an inrush of new EPG events that need to be updated within the EPG storage pool. Whilst these updates are occurring, the number of free entries in the pool is expected to fluctuate. Once the updates have been received and processed, the number of free entries in the pool will stabilise indicating that it is relatively safe to assume that the EPG event pool is up-to-date.