DAVID systems

DigaSystem.
The Radio Operating System

TurboPlayer TechManual

Technical Documentation

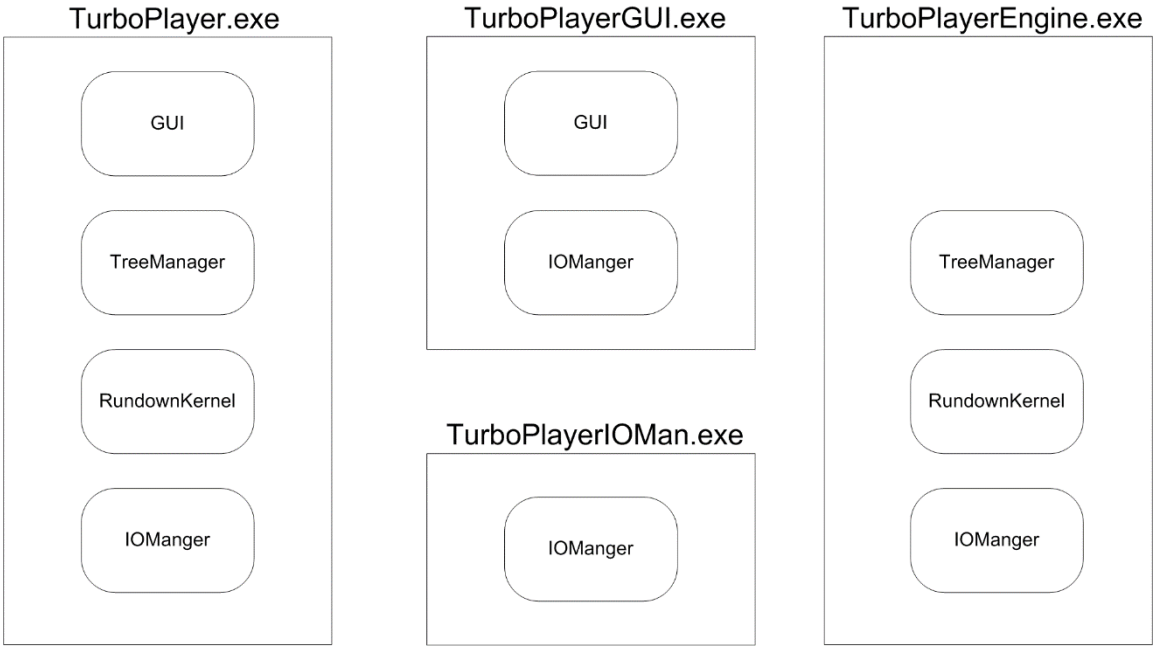
CONTENT

1 SYSTEM OVERVIEW

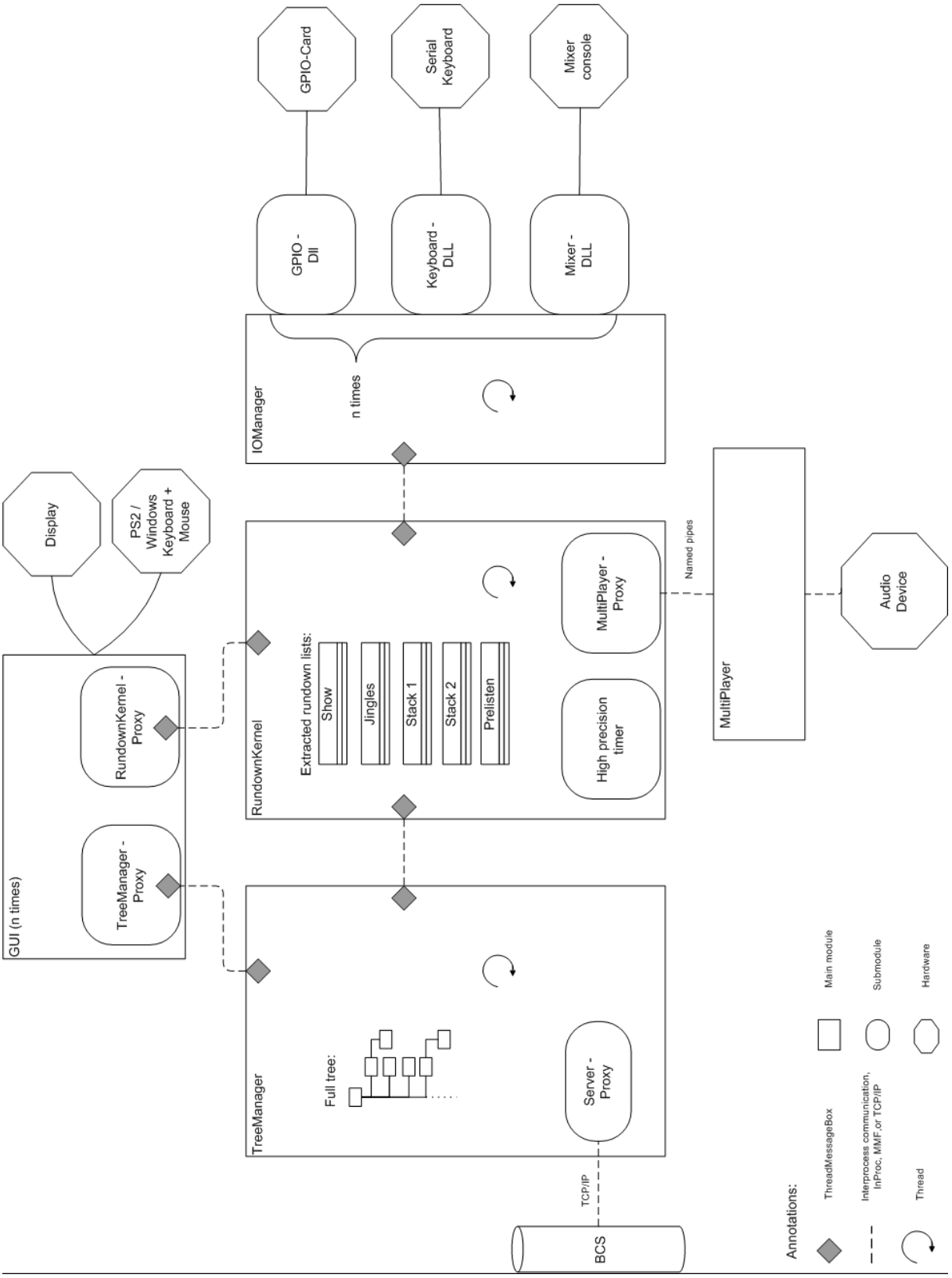
1.1 Module Overview

1.1.1 Internal modules

Internally TurboPlayer is made up of multiple modules (for the programmers: libraries), which can be linked in many ways. There are different executables available, each of them containing a set of these modules – which is shown by the following picture:



The next graph shows more details of the internal structure:



1.2 List of Files

1.2.1 Program files

Program	Directory	Description
TurboPlayer_Full.exe	Digas	The application containing all modules: GUI, engine and IO manager.
TurboPlayerGUI_Full.exe	Digas	The application containing: GUI and IO manager.
TurboPlayerEngine_Full.exe	Digas	The application containing: engine and IO manager.
TurboPlayerIOMan.exe	Digas	The application containing: IO manager.
TurboPlayerService.exe	Digas	An extra module, which can run as a Windows service. It has a built-in web server to host WebTurboPlayer and it provides a web socket server to access TurboPlayer. (Some functions are also available via REST.) This module needs many extra sub-modules, which are listed below.
MultiPlayer.exe	Digas	The multi-channel audio playout engine. Version 3.x is needed.
ProcMan.exe	Digas	A small module installed as service to give MultiPlayer the needed non-paged memory.
DVCCON.exe	Digas	A small helper program to give shared access to the audio board. Needed by MultiRec. Sometimes also called “MEX”.
StandaloneCFM.exe	Digas	A small helper program which allows to run CrossfadeMixer and MultiRec in an external app to increase stability.

1.2.2 Libraries

Library	**Directory**	**Description**
Audio32.DLL	Digas	Library for analyzing and converting audio formats/files. The DLL is optional. If it is not present, internal routines are used which can handle wave files and MPEG1 Layer-2 files. The DLL is required if OnAIR TrackMixer is used or if a waveform display in the players is desired.
BcsLoudness.dll	Digas	License DLL needed for all loudness-relevant functions.
ClipboardFormatConversions.DLL	Digas	Helper DLL needed by OnAIR TrackMixer.
CrossfadeMixer_x.OCX	Digas	Version x of the crossfade mixer.
CxImageList.DLL	Digas	Used for a display of many images (thumbnails).
DigaRTF.OCX	Digas	An ActiveX control which encapsulates the Windows rich text editor and adds a toolbar and some special functionality.
DigaSQL.DLL	Digas	This DLL provides the DigaSystem registry, and the rights system. It is essential for most of the DigaSystem programs.
DigaStory.OCX	Digas	An ActiveX control to display the content of stories as a combination of text and embedded metadata lines.

Library	Directory	Description
MFC140.DLL	Digas or Windows	Microsoft C, C++ library and "Microsoft Foundation Classes 14" for C++ modules created with Visual Studio 2022. All DigaSystem programs need these libraries at runtime. All files can be installed with the Visual C++ Redistributable for Visual Studio 2022, which can be downloaded from the link: https://support.microsoft.com/en-ca/help/2977003/the-latest-supported-visual-c-downloads Please use the x86 version called vc_redist.x86.exe.
MSVCP140.DLL		
MSVCRT.DLL		
VCRUNTIME140.DLL		
MultiRec_x.OCX	Digas	Version x of MultiRecorder. Current version is 4.
OtmControl.OCX	Digas	An ActiveX control with the OnAIR TrackMixer, the successor of CrossfadeMixer.
OtmControlLibrary.DLL	Digas	A helper DLL for OnAIR TrackMixer.
SimIO.DLL	Digas	An IO module only used for testing. It simulates an 8-channel mixer console with motor faders.
System.Windows.Interactivity.DLL	Digas	A helper DLL for OnAIR TrackMixer.
TIOGPIO.DLL	Digas	An IO module for the communication with some IO cards (Relais/opto)
TIOGeneralSerial.DLL	Digas	An IO module for a generic communication with the serial interface, (sent/received bytes) can be fully adjusted.
TIOGenericMidi.DLL	Digas	An IO module for a generic communication with MIDI devices. External controllers or some audio cards can be controlled via MIDI.
TIODHD.DLL	Digas	An IO module for a communication with DHD mixer consoles via serial cable or via IP and the RM2200D protocol.
TIODiamond.DLL	Digas	An IO module for the serial communication with LAWO mixer consoles of types: Diamond, Zirkon, Crystal and Sapphire. Since build 33 Studer OnAir consoles are supported, too.
TIOEmberPlus.dll	Digas	An IO module for a communication with mixer consoles via TCP/IP and the protocol “Ember+”. So far consoles of LAWO and DHD have been tested but with its generic part the module is also usable for other consoles.
TIOMultiCoder.DLL	Digas	An IO module which can control MultiCoder instances. Used for automatic recording.
TIOMidiMotionMix.DLL	Digas	An IO module which supports the CartMotion feature. It has a MIDI/serial interface to a hardware controller and an interface to some Solid State Logic soundcards.
TIOROAD.DLL	Digas	An IO module which can control a ROAD instance for live recordings.
TurboPlayerWrapper.dll	Digas	An assembly which wraps the interface between TurboPlayer engine and the GUI and acts as a broker between C++ and the .NET world.

1.2.3 Documentation and Help Files

File	Directory	Description
TurboPlayerTechManual.Doc	Doku	This manual: the technical manual of TurboPlayer
BCSTechManual.Doc	Doku	The technical manual of the Broadcast System
DIGPARAM.RTF	Doku	The reference of all parameters used in the DigaSystem registry. This file is used in conjunction with InfoPad to provide a help from the Administrator.
*_Changes.doc	Doku	The release notes for the corresponding module.

1.2.4 Miscellaneous Files

File	Directory	Description
*.LOG	Protocol	DigaSystem Protocol files. In the filename the month and the day are specified as two numbers.
*.PDB	Digas	Program database files. Only used for creating stack dumps when debugging.

1.2.5 Assemblies for TurboPlayerService

TurboPlayerService is a .NET based module. Therefore, you need to install the necessary .NET runtime. At the moment this is version 4.5.2.

In addition, a lot of assemblies (dlls) are required. Normally they are part of the distribution package. It should contain:

- DigaSystem.ServiceTool.dll
- Fleck.dll
- Log4net.dll
- Microsoft.Owin.dll
- Microsoft.Owin.Diagnostics.dll
- Microsoft.Owin.Host.HttpListener.dll
- Microsoft.Owin.Host.SystemWeb.dll
- Microsoft.Owin.Hosting.dll
- Microsoft.Practices.ServiceLocation.dll
- Newtonsoft.Json.dll
- Owin.dll
- Owin.WebSocket.dll
- Parameters.dll
- System.Net.Http.Formatting.dll
- System.Web.Http.dll
- System.Web.Http.Owin.dll
- TurboPlayerWrapper.dll

1.3 General Installation Hints

1.3.1 General system requirements

TurboPlayer needs a Win32 operating system. Since version 5.x the OS must be at least Windows Vista or Server 2008. TurboPlayer works on 64-bit systems, but it is not a native 64-Bit program and does not make use of more than the 2 GByte RAM a 32-Bit process can regularly obtain.

Minimum hardware requirement:	Recommended hardware:
1 GByte RAM, 1 GHz CPU	>= 32 GByte RAM, >= 4 cores, >= 3 GHz CPU

In case you want to play out many channels (surround, regionalization, multiple stacks) and / or video the requirements might increase. Especially for the MultiPlayer buffers (locally caching audio or video data) a lot of storage is needed. Processor speed is needed for video decoding or a lot of audio channels.

Important things are the hard disks. In DigaBroadcastSystem it is possible to copy all audio and text files onto the local hard disk of the OnAir client. If you use this feature, it is strongly recommended to use an extra hard-disk for the media files – see the next chapters for more details.

Another important detail is the used network adapter. In some cases, it might be useful to use two adapters (or an adapter with two ports). See the next chapter for more information.

1.3.2 Optimizing a computer for real-time audio handling

Windows is not a real-time operating system. It is a general-purpose or an office operating system. This can make it hard to do real-time processing on such a system. TurboPlayer itself is only a soft-real-time application, because he handles events, timers, etc. only with a time precision of 1 millisecond. This is far from a “hard-core” real-time system, which guarantees a reaction within microseconds or even below that. Nevertheless, the corresponding audio engine – MultiPlayer – needs to process audio data. The required precision might go down below 1 millisecond. For example, one sample in a 48 kHz file has only a duration of ~ 21 microseconds. Luckily, such small time-slots are not necessary, because MultiPlayer only handles blocks of buffered audio data. Normally, it is sufficient for MultiPlayer, too, to get computing time every millisecond.

But here is a problem on a Windows computer. Windows can be installed on a huge amount of different computers, especially, if you consider the possible hardware combinations for all components. Many of these components need a driver. Such a driver normally runs in a very privileged mode, directly below the Windows kernel. One of these privileges is to get exclusive computing time. This is normally used in conjunction with interrupts. An interrupt can be triggered by a hardware device, which needs attention by the driver. The computing time, which is needed by the driver during interrupt handling, can really block the whole computer. Therefore, Microsoft has already made a mitigation with the so-called “deferred procedure calls” (DPCs). The idea is that the developers of drivers should make the time, which is needed for the interrupt handling, as short as possible. Instead, they can use the DPCs to call longer running routines with lower privileges. Nevertheless, this does not help in case of the audio handling by a desktop application like MultiPlayer, because even the computing time within a DPC will block the time scheduler of Windows in a way so that desktop applications do not get any computing time. It should be clear that it can create audible dropouts if MultiPlayer is not able to send audio data to the soundcard because he does not get computing time.

Drivers should normally be written in a way that the time needed for handling interrupts and DPCs is as short as possible and especially each single computing block should not exceed some fractions of 1 millisecond. Most drivers comply to this rule, but some do not. They will create problems for real-time audio handling. Typical examples of these problematic drivers are network drivers (NDIS), which nowadays try to handle big blocks of data in one go – but at the same time exceeding the limit for their allowed computing time. NVIDIA drivers are also quite often affected. But in principle, any driver might create problems.

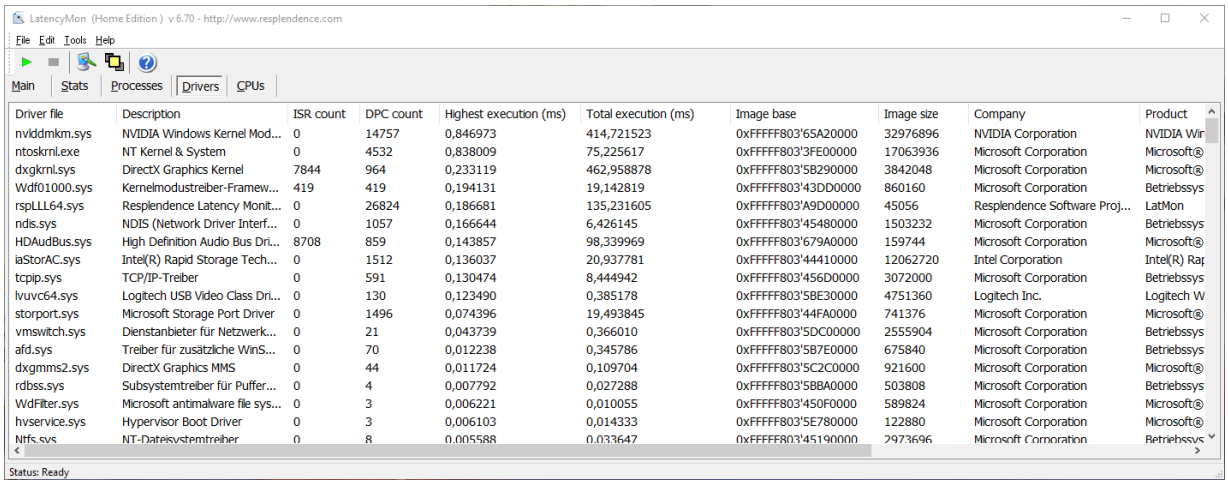
If you have a driver, which creates this problem, this is not easy to solve. Fortunately, there is a free tool out there, which displays the times, drivers are operating in DPCs and are therefore blocking desktop applications. This tool will also name the misbehaving driver, which uses the maximum time. This tool is the “DPC Latency Checker". You can find it here:

http://www.thesycon.de/deu/latency_check.shtml

This tool works only until Windows 7. For newer Windows versions, there is a different tool, the “Latency Monitor”. You can find it here:

http://www.resplendence.com/latencymon

It looks like this: (this is not the initial overview page but the page with data for all drivers)



Driver file	Description	ISR count	DPC count	Highest execution (ms)	Total execution (ms)	Image base	Image size	Company	Product
nvlddmkm.sys	NVIDIA Windows Kernel Mod...	0	14757	0,846973	414,721523	0xFFFFF803'65A20000	32976896	NVIDIA Corporation	NVIDIA Wi
ntoskrnl.exe	NT Kernel & System	0	4532	0,838009	75,225617	0xFFFFF803'3FE00000	17063936	Microsoft Corporation	Microsoft®
dxgkrnl.sys	DirectX Graphics Kernel	7844	964	0,233119	462,958878	0xFFFFF803'5B290000	3842048	Microsoft Corporation	Microsoft®
Wdf01000.sys	Kernelmodustreiber-Framew...	419	419	0,194131	19,142819	0xFFFFF803'43DD0000	860160	Microsoft Corporation	Betriebssys
rspLLL64.sys	Resplendence Latency Monit...	0	26824	0,186681	135,231605	0xFFFFF803'A9D00000	45056	Resplendence Software Proj...	LatMon
ndis.sys	NDIS (Network Driver Interf...	0	1057	0,166644	6,426145	0xFFFFF803'45480000	1503232	Microsoft Corporation	Betriebssys
HDAudBus.sys	High Definition Audio Bus Dri...	8708	859	0,143857	98,339969	0xFFFFF803'679A0000	159744	Microsoft Corporation	Microsoft®
iaStorAC.sys	Intel(R) Rapid Storage Tech...	0	1512	0,136037	20,937781	0xFFFFF803'44410000	12062720	Intel Corporation	Intel(R) Rap
tcpip.sys	TCP/IP-Treiber	0	591	0,130474	8,444942	0xFFFFF803'456D0000	3072000	Microsoft Corporation	Betriebssys
lvuvc64.sys	Logitech USB Video Class Dri...	0	130	0,123490	0,385178	0xFFFFF803'5BE30000	4751360	Logitech Inc.	Logitech W
storport.sys	Microsoft Storage Port Driver	0	1496	0,074396	19,493845	0xFFFFF803'44FA0000	741376	Microsoft Corporation	Microsoft®
vmswitch.sys	Dienstanbieter für Netzwerk...	0	21	0,043739	0,366010	0xFFFFF803'5DC00000	2555904	Microsoft Corporation	Betriebssys
afd.sys	Treiber für zusätzliche WinS...	0	70	0,012238	0,345786	0xFFFFF803'5B7E0000	675840	Microsoft Corporation	Betriebssys
dxgmms2.sys	DirectX Graphics MMS	0	44	0,011724	0,109704	0xFFFFF803'5C2C0000	921600	Microsoft Corporation	Microsoft®
rdbss.sys	Subsystemtreiber für Puffer...	0	4	0,007792	0,027288	0xFFFFF803'5BBA0000	503808	Microsoft Corporation	Betriebssys
WdFilter.sys	Microsoft antimalware file sys...	0	3	0,006221	0,010055	0xFFFFF803'450F0000	589824	Microsoft Corporation	Microsoft®
hvservice.sys	Hypervisor Boot Driver	0	3	0,006103	0,014333	0xFFFFF803'5E780000	122880	Microsoft Corporation	Microsoft®
Ntfs.sys	NT-Dateisystemtreiber	0	8	0,005588	0,033647	0xFFFFF803'45190000	2973696	Microsoft Corporation	Betriebssys

After installing a computer, run it for at least some hours to check for bad drivers ! This would be a driver, which consumes more than 1 ms of execution time in a DPC. If you encounter such a bad driver, you might solve the problem by deactivating/upgrading/downgrading/changing the driver. If that does not help, you need to change the hardware. Sometimes, it helps to use a different slot for an extension card or to change BIOS settings. Unfortunately, this is a pure try-and-error situation. If, at the end, an essential driver is affected, the system cannot be used for a playout with TurboPlayer !

Bad drivers are by far not the only problem, which can exist on a computer, and which can prevent real-time audio data handling. Many problems arise from Windows itself or from Windows settings or “optimizations”. Unfortunately, Windows was more and more optimized over time for less power consumption or improved network throughput or other stuff, but not for real-time processing. The best Windows version so far was Windows XP. Since then, the situation has worsened. Nevertheless, it should still be possible to configure Windows in a way that a computer is suitable for playout. The next paragraphs describe a bunch of settings and give tips, how you can optimize Windows accordingly. This is only a selection of common/important things. Others might be necessary. You can find other tips&tricks if you look for setting up a computer for a “digital audio workstation”. Manufacturers of these programs have manuals similar to this one.

1.3.2.1 Harddisks / SSDs

When you configure your playout computer, you should pay attention to the used hard disk. The manufacturer is not of concern, and you do not have to consider this topic if you play all audio data directly from file servers. But if you intend to cache audio files locally on the playout computer (with a corresponding media directory) then you should avoid two things:

- Do not use the hard disk, where the operating system is located, for the audio data, too! Windows is permanently reading from and writing to its system disk, and this can very well interfere with continuously and uninterrupted reading audio from the same disk.
- Do not use SSDs. This might seem counterproductive, as nowadays SSDs are the new standard. But the firmware of SSDs can optimize the data storage at any time, by moving blocks of data around or refreshing data. This could possibly interfere with reading the audio. Classical hard disks do not have this problem. As long as you do not start explicit cleanup programs, data will not be moved around. And classical hard disks are much faster than what is needed for handling one or some audio streams, so their performance is not an issue.

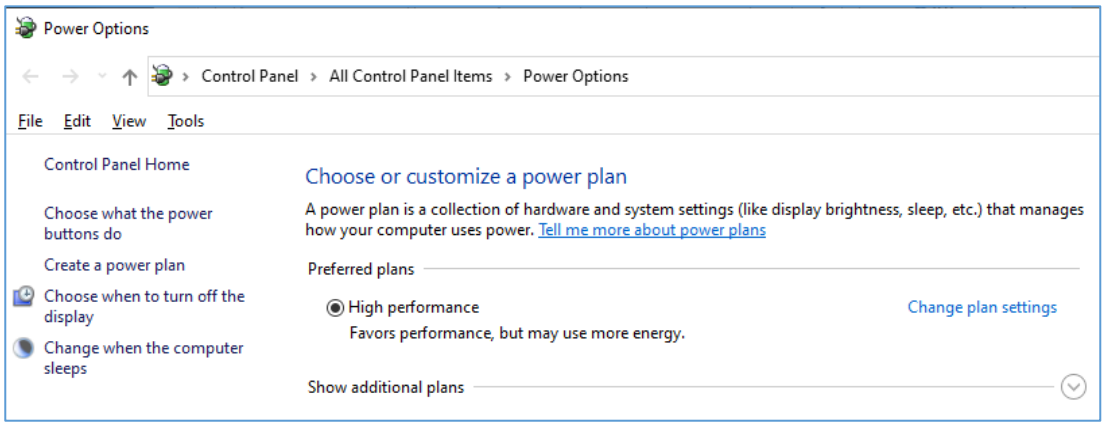
1.3.2.2 BIOS

- Generally, you should disable all sort of power management features of the CPU. These features typically imply changing the CPU speed, which is not suitable for audio handling.
- Disable any overclocking.
- Disable hyper-threading (Intel) / Simultaneous multi-threading (AMD).

- Disable “CPU power saving mode”.
- Disable “Intel Turbo Boost Technology” / “AMD Turbo CORE”.
- Disable “Enhanced Intel Speed Step” (EIST) / AMD Cool ‘n’ Quiet.
- Disable all C state support, like: C3 C6, C7, C8, C1E (CPU enhanced halt). Possibly, this option is called ‘Disable CPU Idle State for Power Saving’.

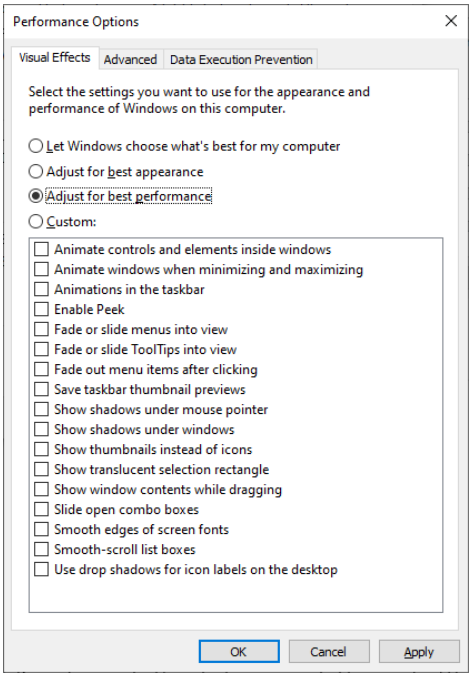
1.3.2.3 Windows power/performance settings

- Open the control panel page “Power options”:

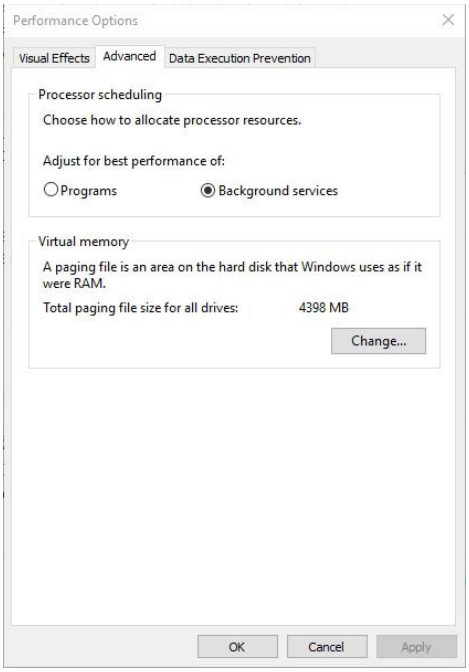


Select “High performance”, If it is not visible click on “Show additional plans”. If you have the plan “Ultimate performance” it is even better and should be selected, but it is only available on Windows10/11 Pro for Workstations. Or create a new power plan (i.e. Playout) and configure this and disable all power saving. You need to go to all the details, unfortunately.

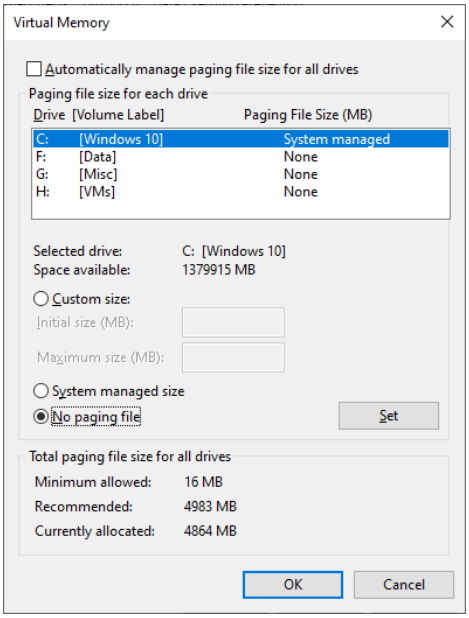
- Click onto “Change plan settings”, then “Change advanced power settings” to open the dialog “Power options” with tab “Advanced settings”. Even if you have selected a high-performance power plan, it might be that individual settings are not set to high or maximum performance. You should check especially the settings for USB, the graphic card and the network card.
- Start the settings program “SystemsPropertiesPerformance.exe”. You can enter “Performance options” in the start menu.



In the “Visual Effects” tab select “Adjust for best performance”. This should automatically disable all individual optimizations for the visual effects. Go to tab “Advanced”:



Select “Background services”. If you have a computer with 32 GByte RAM or more (which you should have), then press the “Change” button in the “Virtual memory” section to configure the paging file:



Select “No paging file”.

1.3.2.4 Installed programs and background services

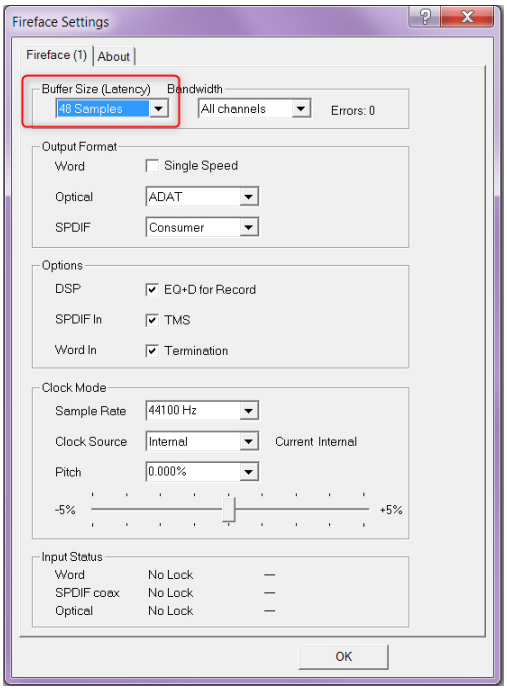
- Generally spoken: the more programs you have installed on your playout computer, the more problems can arise. Therefore, it is recommended to install only software, which is really necessary for the playout use-case. You should also remove unused hardware and deinstall corresponding drivers.
- A type of software, which is notoriously know for making trouble, are virus scanners. Nowadays, it is common to have one installed, of course, but you should at least monitor what an installed scanner is doing – especially if you encounter problems. Typically, the problems arise, because these scanners interfere with the network traffic or by running file scans in the background, sometimes in the worst suitable moment.
- Tweaking software or software for printers can create problems, too.

- You should also check for configured background processes, which can result in bursts of used computing time or network traffic. There are too many possible programs, but one type should be mentioned: background synchronization by Dropbox, OneDrive or GoogleDrive, iCloud or something similar. Use the Windows task scheduler to identify background processes, which are not essential, and which might disturb the playout.
- For your video card only install the necessary drivers. Additional software, like e.g. GeForce Experience, PhysX or AMD Settings are normally not needed (they are intended more for graphical programs, like games). Generally spoken, any software or option that boosts graphics/gaming performance can interfere with real-time audio handling.
- For the same reason, avoid using dedicated mouse and keyboard drivers and extra software coming with them. Better stick to the drivers coming with Windows.
- Do not use the software “Link Power Management” for “Intel Rapid Storage Technology”. It is known to create audio dropouts.

1.3.2.5 ASIO buffer size

- ASIO is a driver interface, which is very common for professional soundcards. Nevertheless, it has a big disadvantage regarding real-time audio handling. The problem is that with this driver model it is up to the driver to request new audio data from an application when the driver thinks it is running out of audio data. As a consequence, the application is not in control of the begin of audio playout. When MultiPlayer wants to start to play something <now>, he has to wait until the ASIO driver requests new audio data for the next time and only then MultiPlayer can start to fill in the first samples of the new audio element. It should be clear that this driver concept leads to a varying start delay, because it is a random thing how long it will take each time when MultiPlayer wants to start an element until the time when the ASIO driver requests new data.

  Fortunately, it is normally possible to configure the ASIO buffer size, which is used by the driver. It is typically configured as the number of audio samples in one buffer. Here is an example of the driver settings of an RME sound card:



  If you want to know how long the configured buffer size is in milliseconds, you need to divide the sample number through the sample rate being used. In the screenshot above it would be:

  48 samples / 44100 samples/s = 0.0010 s = 1 ms

  In order to have the lowest possible latency, you need to make the buffer size as small as possible. But if you make it too small, you will have audible audio dropouts, because the application must be able to deliver the audio data with the frequency, which results from the settings. In the example above it would be necessary that MultiPlayer gets computing time at least once per millisecond in order to not miss one or more buffers.

As a rule of thumb, you should start with a buffer size of some milliseconds, e.g. 256 samples / 6 ms and try, whether you encounter audio dropouts. If not, then reduce the buffer size. When you encounter audio dropouts then increase the buffer size again by one or max. two steps.

One additional hint: other Windows driver models, like MME or WDM do not have this problem. Nevertheless, it might not help to switch to one of these driver models. Especially for RME cards it does not help, because RME made the ASIO driver the base one, and other drivers are implemented on top of it.

1.3.3 Communication settings

As you can see from the graph about the internal structure, the GUI can be separated from the engine. Please note that there are two connections from the GUI, one to the tree manager and one to the rundown kernel. Commands sent to the rundown kernel are time critical, commands sent to the tree manager are not. In case you used only a single connection, it could happen that time critical commands cannot be sent, because the line is just occupied by other data transfer. As the GUI uses two connections (with two IP ports) it is possible to run the connection between GUI and rundown kernel on a separated cable, which directly connects the two computers. Therefore, time critical commands have an exclusive access to the network, which guarantees a fast transmission.

The inter-process communication used within TurboPlayer needs some explanation. A library with "**ThreadMessages**" and "**ThreadMessageBoxes**" is used. In the graphical overview above you can see that these messages are used for all the communication between the internal modules. These messages are convenient for the programmer, as he does not need to know whether messages are being sent within the same application or to a different application on the same or to a remote computer. It is up to the administrator to configure the communication settings properly. Only the administrator knows in which applications and on which computers his engines, GUIs, IO managers or remote TurboPlayers are running. All these settings can be found in the key "TurboPlayer\Communication". Details are described in the parameter descriptions below. Here we describe only the general part.

The thread message library can handle three ways of communication: in-process, between two applications on the same computer and between two applications on different computers:

- In-process: This can only be used if both sides are located within the same application. If you run a TurboPlayer_Full.exe all modules are included and can communicate with this method. The address format is:
  InProc-Address=<MessageBoxName>
  Each of the message boxes has a name. To connect to such a message box, the other side only needs to know the correct name. An example: the kernel opens a box with name "RundownKernel-GuiBox1" for GUI 1.
- Out-of-process on the same computer: This type is used if you have two applications which are running on the same computer, e.g. the TurboPlayerEngine and a TurboPlayerGUI. The address format is:
  MMF-Address=<MessageBoxName>
  Again, only the name of the message box must be known. It is always the same name as for the in-proc communication. The identifier "MMF" describes the inter-process communication being used: a memory mapped file. This is a means Windows provides to exchange data between two processes.
- Out-of-process on different computers: This type is used if both sides reside on different computers, e.g. a TurboPlayerEngine and a remote GUI. The address format is:
  IP-Address=<DNS name or IP address>:<port number>
  Here the knowledge of the message box name is of no help. We must specify the computer to which the connection should be established and the port number which is being opened on the remote computer. The specifier "IP" describes the communication method and needs no explanation. An example: "IP-Address=10.11.12.13:9981"

In the parameter "TurboPlayer\Communication\ComTypes" you can activate all methods you want to use. Each of these methods is optimized: InProc is faster than MMF and MMF is faster than IP. Nevertheless, you can use MMF or even IP for the in-process communication too. Especially MMF can be useful: if you specify this method you can start a full TurboPlayer or alternatively a separate TurboPlayerEngine and a TurboPlayerGUI on the same computer without changing the settings. This can be useful if you encounter crashes of the GUI. You can operate the GUI in a separate process and the engine will not be

influenced from these crashes. This is possible because a GUI can be started at any time and will automatically synchronize with the current engine state during startup.

Sometimes you will see in the parameter descriptions that you have to enter port numbers, though you do not want to use IP communication at all. This is true for the message boxes the kernel and the tree manager open for the communication with the GUIs and the IO manager. Nevertheless, real IP ports are only opened if you specify IP as method in the "ComTypes" parameter. The ports must only be specified, because the same parameter is used to deduce the number of needed message boxes which will be opened. This brings up an additional and very important point:

Every message box can only be used for exactly one connection ! You must not connect multiple clients to one message box! Notice that there is no protection against misconfiguration, there will be only some malfunctions or warnings within the logfiles. So you must pay attention to use each message box (and port !) only for one other side. Be extremely careful if you configure a network of GUIs and/or remote TurboPlayers: make a table or a graph with all IP addresses and ports to ensure one-to-one connections.

Anonther hint: with IP one side normally behaves as a server: it opens an IP port and waits for connect requests on this port. The other side will be the client: it actively connects to the listen port of a waiting server. Within TurboPlayer you can use sometimes the possibility that both sides can be server and client and the same time. Both sides will open a listen port and both sides will try to connect to the other side. Of course, one side will be faster and "wins", but this does not matter. The advantage of this solution is that you can start both sides in arbitrary order, it is not necessary to start the server side first. This sort of simultaneous client/server communication is used between kernel and IOManager and for the remote TurboPlayers. In these cases, it is possible to specify the two parameters "Port" (the local IP port for the server side) and "Address" (for the client side) at the same time.

1.3.4 Command line arguments (case insensitive):

TurboPlayer supports the following command line options. Options are not case sensitive and you can use either a minus "-" or a slash "/" to start an option. If an option has data it can follow after a space, after a colon or after an equal character. But as data must be enclosed in quotation marks if it contains spaces, a slash or a minus/hyphen, it is recommended to enclose data always in quotation marks. Example: -config"MyConfig"

Option	Data	Meaning
-config -c	configuration	Configuration to use. The configuration is a top-level registry key which is used to load all settings. Standard value is "TurboPlayer". This parameter is supported by TurboPlayer, TurboPlayerGUI, TurboPlayerEngine and TurboPlayerIOMan. You can specify the configuration after a space, a colon or an equal sign, e.g. -c=TurboPlayer_B3 This parameter is available to all modules and also for pre-v6 versions.
-install -i	-	Can be specified if TurboPlayerEngine.exe is executed from a command line which must be opened with "Run as administrator" ! The engine will try to install itself as a new Windows service and terminate afterwards. Some details can be changed in the properties of the service in Service Control Manager of Windows. Other options you can specify for the installation are: -config, -account, -password, -service, -BCS, -InitProgram, -noBCS, -noMP. The new service will be called "TurboPlayerEngine" if you do not specify a config option (meaning the default is used). If you specify a config, it is appended to the service name after a minus. For example, if you install the service with this command line: TurboPlayerEngine.exe –i –config"MyConfig" The created Windows service will use this name: "TurboPlayerEngine-MyConfig". Please have in mind that it is NOT POSSIBLE to run multiple TurboPlayer engines at the same time! It is only supported to install the engine multiple times to have multiple configurations available, but you must never start multiple of TurboPlayerEngine services at the same time! This command line argument is supported since version 6.0 by TurboPlayerEngine.exe and by TurboPlayerIOMan.exe.
-uninstall -u	-	Uninstalls TurboPlayerEngine/TurboPlayerIOMan as a Windows service. In order to uninstall a service, which has been installed with a config name you must specify the option –config, too. This command line argument is supported since version 6.0 modules.

-account	username	If you install TurboPlayerEngine/TurboPlayerIOMan as a new service, it will run with the account you specify. If you omit these options, the account “SYSTEM” will be used. Though appropriate for many Windows services, the SYSTEM account is a bad choice. The programs needs to have access to files like logfiles or the DigaSystem registry files via the DigaSQL.dll. The SYSTEM account will probably have access to all files on a local harddisk. But if files on the network, with UNC names or mapped drives must be accessed the SYSTEM account does not have enough rights or cannot see drive mappings. Therefore, it is recommended to always use a specific user to run the services. If files, which must be accessed are located on a file server of a domain, it is a good idea to use a domain user. Such an account has the format “<domain name>\<user name>”. Otherwise, a local user is sufficient. Please enclose the username and the password in quotation marks. Possibly, it is also necessary to grant the used account the right to logon as a service in the local security policies of Windows (->secpol.msc). These parameters are NOT used for the initial DigaSystem log-in. For that purpose only the registry parameters TurboPlayer\User and TurboPlayer\Password are used. This command line argument is supported since version 6.0 by TurboPlayerEngine.exe and by TurboPlayerIOMan.exe.
-password	password	
-service	-	This option is automatically used if TurboPlayerEngine/ TurboPlayerIOMan installs itself. This option signals the program that it is started as a service and therefore should never be used otherwise. The impact of this option is very limited. The programs will detect that they are running as a service automatically – even if this option is missing. Unfortunately, this happens only after some first initialization steps have been executed. If one of these steps encounters a problem, it is possible that a message box is displayed. As this would not be visible for a service, which runs on session 0, this option suppresses messages boxes during the first part of the startup process. This command line argument is supported since version 6.0 by TurboPlayerEngine.exe and by TurboPlayerIOMan.exe.

-BCS	name	Specify the name of the BCS to connect with, if empty: run without a BCS connection. You have to specify a registry key name of the keys below "Digas\PlanServer". This switch overrides the corresponding registry parameter "TurboPlayer\ServerToConnect". Connecting to a BCS is normally initiated by the first GUI, which connects to the engine. Since version 6.0 you can use the registry parameter "TurboPlayer\InitByEngine=Connect" to achieve that the connection is directly established by the engine, without waiting for a GUI to trigger it. This parameter is evaluated by TurboPlayerEngine, TurboPlayerGUI and the full TurboPlayer. The GUI evaluates it also for pre-v6 versions.
-InitProgram	name	Specify the name of the initial program/service that will be selected after startup. This switch overrides the corresponding registry parameter "TurboPlayer\InitialProgram". Selecting a program is normally done by the first GUI, which connects to the engine. Since version 6.0 you can use the registry parameter "TurboPlayer\ InitByEngine=Connect,SetProgram" to achieve that the program is directly selected by the engine. This parameter is evaluated by TurboPlayerEngine, TurboPlayerGUI and the full TurboPlayer. The GUI evaluates it also for pre-v6 versions.
-noBCS	-	Application runs without connecting to any BCS (the same as specifiying an empty string after -BCS=). Have in mind that all rundowns will be empty after start and you need a concept how to fill them. One way would be to use import buttons in the GUI together with an import module. Or the user could drag&drop from MiniDBM to a rundown. In any case it will be necessary to set the stack mode for the rundowns to "Clipboard". See parameter: TurboPlayer\RundownLists\<Name of rundown>\StackMode. This parameter is evaluated by TurboPlayerEngine, TurboPlayerGUI and the full TurboPlayer. The GUI evaluates it also for pre-v6 versions. Even if you have started TurboPlayer without a BCS connection, you can connect to a BCS by calling the macro command TP_ConnectToBCS (and perhaps setting a program and loading a show) afterwards.
-noMP	-	Application runs without using MultiPlayer. This parameter is evaluated by TurboPlayerEngine and the full TurboPlayer and also for pre-v6 versions.
-noMRec	-	TP runs without MultiRec and CrossfadeMixer/OnAIR TrackMixer (need not to be installed on the machine). No prelisten by EasyPlayer or CFM/OTM is possible. The same option is available on the general settings. This parameter is available to TurboPlayerGUI and the full TurboPlayer and also for pre-v6 versions.

-GUINR	n	For the startup of a TPGUI: the GUI number to be used. This overrides the registry parameter TurboPlayer\Communication\ GUI\GuiNumber. This parameter is only evaluated by a TurboPlayerGUI and only since version 6.0.
-TMAGUI	address	For the startup of a full TurboPlayer or a GUI only: the address to be used by the GUI for connecting to the TreeManager. This overrides the registry parameter TurboPlayer\Communication\GUI\ TreeManagerAddress. See the parameter description in DigParam.rtf for details. Important: always put the argument in quotation marks, because otherwise special characters will be interpreted in a wrong way. Example: /TMAGUI="IP=TPHOSTNAME:9962" This parameter is only evaluated by a TurboPlayerGUI and only since version 6.0.
-RKAGUI	address	For the startup of a full TurboPlayer or a GUI only: the address to be used by the GUI for connecting to the RundownKernel. This overrides the registry parameter TurboPlayer\Communication\GUI\ RundownKernelAddress. See the parameter description in DigParam.rtf for details. Important: always put the argument in quotation marks, because otherwise special characters will be interpreted in a wrong way. Example: /RKAGUI="MMF=RundownKernel-GuiBox2" This parameter is only evaluated by a TurboPlayerGUI and only since version 6.0.
-RKAIOM	address	For the startup of a TP GUI only and for the startup of TurboPlayerIOMan.exe: the address to be used by the internal IOManager for connecting to the RundownKernel. This overrides the registry parameter TurboPlayer\Communication\IOManager\ RundownKernelAddress. See the parameter description in DigParam.rtf for details. The registry parameter "InitializeIOManager" is always assumed to be true if this command line argument is given. It is only possible to use the IOManager within the started GUI as a client. Using it as the server side is not supported. Important: always put the argument in quotation marks, because otherwise special characters will be interpreted in a wrong way. Example: /RKAIOM="IP=TPHOSTNAME:9982" This parameter is only evaluated since version 6.0.

1.3.5 Application separation: GUI, engine and StandaloneCFM

The whole BCS-OnAir system is a big and complex system. Nevertheless, it could be compiled into a single application only. This might be very convenient for the installation and handling, but it has a major drawback: whenever one of the contained modules crashes the whole system is affected. For an on-air system this is not desirable.

Therefore, it was decided to separate the system into multiple applications:

1) TurboPlayerEngine
2) TurboPlayerGUI
3) MultiPlayer
4) IOManager
5) StandaloneCFM

There is also the TurboPlayer_Full.exe which contains: 1+2+4+5. In principle you can always start to use the TurboPlayer_Full.exe (together with MultiPlayer which always stands alone). As long as you do not have stability problems this is fine.

But if you encounter crashes, you can separate the applications to avoid that a crashing module affects the other modules. Be warned about two things: first, such a separation does not avoid the problems/crashes, but it makes the consequences smaller and second, the separation is not perfect, because TurboPlayerEngine and MultiPlayer must not crash: they are indispensable for on-air playout. But TurboPlayerGUI and StandaloneCFM can crash with no influence to on-air playout, remaining only the annoyance for the user. This is also the most helpful separation because there are two components in the GUI which are good candidates:

TurboPlayer makes use of the modules “MultiRec_x.ocx” and “CrossfadeMixer_x.ocx”. They are used for the GUI prelisten functionality (see Concepts/Prelisten below) and for editing crossfades visually. Both modules are very complex (MultiRec is also used as base for the MultiTrack editor family and CrossfadeMixer is itself some sort of multi-track audio editor. If one of these two modules crashes, you can make use of the **StandaloneCFM.exe**. This application loads instances of CrossfadeMixer / MultiRec and provides them to external applications like TurboPlayer. The windows of these modules appear within the main application and the user should not notice that the modules are running in a different process.

But if one of these modules crashes only the StandaloneCFM.exe is closed by Windows, TurboPlayer (GUI and engine) continues running unaffected. This allows to continue with on-air playout. When the user needs CrossfadeMixer or MultiRec the next time, TurboPlayer automatically restarts StandaloneCFM.exe.

It is very easy to switch to a separated operation: in the settings dialog of CrossfadeMixer and / or of EasyPlayer prelisten you have to check the box which says: “Run … as standalone process”. In addition, you must specify the location of the StandaloneCFM.exe in the DigaSystem registry so that TurboPlayer knows how to start it. This is done in key “Programs” in value “StandaloneCFM”.

If you use OnAIR TrackMixer (the successor of CrossfadeMixer) you cannot run it in StandaloneCFM. OnAIR TrackMixer can run only direct in the TurboPlayer GUI process.

A separation of **GUI and engine** needs a little bit more work. If the engine is not started by a batch script, the GUI can start it if its location is defined in the value “TurboPlayerEngine” in key “Programs”. In addition, the communication between GUI and engine must be set up properly (See “Communication settings” above). For the ThreadMessageBox communication between two programs on one computer “MMF” (memory mapped file) is recommended. Therefore, the value “TurboPlayer \ Communication \ ComTypes” must contain “MMF”. And values in key “TurboPlayer \ Communication \ GUI” must specify an MMF address. Typically (for the first GUI) this is:

RundownKernelAddress = MMF=RundownKernel-GuiBox1
TreeManagerAddress = MMF=TreeManager-GuiBox1

A separation of **IOManager and engine** is only necessary if you want to run the IOManager on a separate computer. So far, we did not hear of a single crash in IOManager or an IOModule in operation.

1.4 Concepts

1.4.1 TurboPlayer flavors

TurboPlayer is available in 4 so-called "flavors". They are different in their supported features and in the price.
The 4 flavors are:

- TurboPlayer_Full (=Pro)
- TurboPlayer_News
- TurboPlayer_Cart
- TurboPlayer_Lite

The optional features are:

- Showlist: Needed to display the main rundown or additional tracks.
- Stack lists: Up to 16 additional rundowns for multi-purpose use
- Jingles: The Jingles window.
- MiniDBM: Windows with an easy access to the DigaSystem database.
- Text display: Windows to display info / presentation text in the GUI.
- Time Info fields: Small windows with time information in the GUI.
- Automatic mode: Support of activation mode "automatic" for operation without presenter.
- LiveAssist mode: Support of activation mode "Live assist" for operation with presenter.
- Passive mode: Support of activation mode "passive" for stand-by operation.
- Rehearse / Standalone modes: Support of connections modes "rehearse" and "standalone".
- Remote function: Needed to establish connections to remote TurboPlayers and to perform remote starts. Includes sync-start of elements.
- Mixer source handling: Support for mixer sources.
- Faderstart: Needed to start elements with a fader.
- Prelistening: Support for prelistening via audio engine (Multiplayer) and PFL.
- CrossfadeMixer: CFM / OnAIR TrackMixer can be used to plan crossfades between elements.
- EasyPlayer: MultiRec can be used as a simple pop-up player for prelistening.
- Cart view: Rundowns and MiniDBM can display elements in a cart view.
- MusicMaster NEXUS Server Interface: Activation of this interface in the GUI.
- CartBeat: Allows a beat-exact start of elements.
- RundownMode Sequenced: Rundown playout exactly as scheduled.
- RundownMode Once: Every element can be played once in any order.
- RundownMode Random: Every element can be played in any order and frequency.
- RundownMode MultiSequenced: Like sequenced, but start with any element.
- DigaMessage: A window displaying messages from DigaMessage is available.
- Import: Import buttons for importing rundowns are available.
- Free show selection list: The list of loaded shows can be arbitrarily selected.
- Thumbnails: Thumbnails for videos/images can be displayed.
- MultiGUI: An engine allows multiple GUIs to connect.
- Recording: Allows audio recordings via MultiRec_x.ocx use in the GUI.
- Timeline: A GUI window which displays a timeline overview about a rundown.
- Embedded start: subclips triggering the start of live metadata elements (->reusing content)
- Stack mode “Endpoints” for feature “EndpointsStories for regionalized playout”

This matrix shows which feature is supported by each flavor:

Feature	TP Full/Pro	TP News	TP Cart	TP Lite
Showlist	✔	✔		✔
Stack lists	✔	✔	✔	✔
Jingles	✔	✔	✔	
MiniDBM	✔		✔	
Text display / DigaStory usage	✔	✔		
Time Info fields	✔	✔		
Automatic mode	✔	✔		✔
LiveAssist mode	✔	✔	✔	
Passive mode	✔	✔	✔	
Rehearse / Standalone modes	✔			
Remote function / SyncStart	✔			
Mixer source handling	✔			✔
Faderstart	✔	✔	✔	
Prelistening	✔	✔	✔	
CrossfadeMixer / OnAIR Track Mixer	✔		✔ [1]	
EasyPlayer	✔	✔	✔	
Cart view	✔		✔	
MM Nexus Server Interface	✔			
CartBeat	✔		✔	✔
RundownMode Sequenced	✔	✔	✔	✔
RundownMode Once	✔	✔	✔	
RundownMode Random	✔	✔	✔	
RundownMode MultiSequenced	✔	✔		
DigaMessage	✔	✔		
Import	✔	✔	✔	
Free show selection list	✔	✔		
Thumbnails	✔		✔	
MultiGUI	✔			
Recording	✔	✔		
Timeline	✔			
Embedded start	✔			✔
Stackmode “Endpoints”	✔			

1) Cart flavor supports CrossfadeMixer only in single track mode.

1.4.2 Configurations

The TurboPlayer engine can start only in a single instance per computer. But each time TurboPlayer is started, a different configuration can be chosen. A configuration is the whole set of parameters and settings which are loaded from the DigaSystem registry. This includes all BCS connection, service, GUI settings, line and rundown settings, IO modules, events, mixer sources, … everything. A typical usage of this feature is to use a single installation for multiple services which are being sent from a single studio. Depending on the started configuration TurboPlayer will connect to a different BCS, load the correct service and might appear totally different.

Each configuration is stored within a single top-level registry key (also the configuration name). The default key is "TurboPlayer". It is recommended to name further keys "TurboPlayer_xyz" so they stay together. It is not possible to have a subkey of TurboPlayer in multiple configurations, e.g. TurboPlayer\GUI1 and TurboPlayer\GUI2. If some settings should be identical for multiple configurations you have to create separated top-level keys nevertheless and you have to do the synchronization manually.

There are multiple ways to select a configuration during startup:

- If there is only the "TurboPlayer" key, this configuration is started.
- You can specifiy a command line parameter c to select a configuration. Examples: "-c TurboPlayer_B3" or "/C:TurboPlayer_Emergency". Of course, a top-level key with the specified name must exist, otherwise TurboPlayer would load default settings. This command line parameter can be used with TurboPlayer_Full.exe, TurboPlayerGUI_Full.exe, TurboPlayerEngine_Full.exe and TurboPlayerIOMan.exe.
- TurboPlayer can show a selection dialog during startup and the user can choose a configuration. Therefore, you do not specify a command line parameter but define the choosable configuration in the key "TurboPlayer\Configurations". (Hint: This is the single parameter which cannot be moved to a different top-level key, because TurboPlayer must have something to start with.) In this key you can list the configurations as sub-values, for example:
    - TurboPlayer = Standard settings
    - TurboPlayer_Emergency = Settings in case of an emergency
    - TurboPlayer_Youth = Settings for the youth service

  The value name must be identical to the top-level key of the configuration. The data is a plain text which is displayed to the user in the selection dialog.

  If the value name

  - starts with an underscore **"_"** the configuration **is ignored**;
  - ends with a star **"*"** this config is selected as **default** within the dialog and it is **automatically loaded** after a timer elapses. To stop the timer just mouse click within the config list or use the up/down keys.
  - ends with more then one star, e. g. **"**"**, this config is selected as **default** within the dialog but **no automatic load** is executed (no counter).

Pay attention to the fact that only TurboPlayer can handle multiple configurations. Other programs like DigAIRange always read from the key "TurboPlayer". Therefore some settings must stay in a key with this name (typically the events, mixer sources, channels-out or stingers). But this type of settings has to be located in the global registry anyway and for most of them there is the value "Program" to limit items to specific services.

Extension with TurboPlayer v6

Since version 6.0 TurboPlayer supports to define configurations not only in values in the DigaSystem registry (as described above), but also in subkeys of the key "TurboPlayer\Configurations". The usage of subkeys instead of values allows to define additional parameters.

The old system with the values can still be used. If you define both, subkeys and values, TurboPlayer will add all of them to a combined list of available configurations. Nevertheless, for clarity we recommend using either values or subkeys, but not both.

If you use subkeys, the name of the subkey must be identical to the top-level key of the prescribed TurboPlayer configuration. The name to be displayed to the user can be set in a value called "Label".

There are additional parameters, which allow to limit the usage of certain configurations for specified computers, or to make the default configuration dependent on the computer name. This can help when maintaining TurboPlayer configurations in the global registry. For more information about the possible parameters, please see the parameter documentation in DigParam.rtf.

1.4.3 Activation modes

There are three activation modes: Passive, LiveAssist and Automatic. Another term is "active", meaning TurboPlayer is either in mode LiveAssist or Automatic. Below you will find a description of the behaviour of TurboPlayer in all modes. In the description of the start/stop possibilities it is assumed that you did not change the "StartStop…" values in key "TurboPlayer", because with theses values you can configure for each activation mode, which start/stop possibilities are allowed.

The standard mode is **LiveAssist**. This mode is intended to be selected, if there is a presenter using TurboPlayer. Faderstarts and button starts / stops are executed. Automatic starts (= time starts, sequenced starts and relative starts) are executed as scheduled. TurboPlayer stops at elements with start mode manual and waits until the user starts the element.

The mode **Automatic** can be used for an unattended operation. Faderstarts and button starts / stops are possible normally, but you can deactivate them to avoid interruptions by user interference. Besides, you can configure different line modes (see below) in automatic mode, meaning that the treatment of lines and faders is different than in other modes. Automatic starts (= time starts, sequenced starts and relative starts) are executed as scheduled. Elements with start mode manual are treated as sequenced. There are some other differing details which can be summarized like this: in automatic mode it is most important that TurboPlayer continues playing. An example: if there is an element in the rundown which cannot be preloaded (e.g. because the file cannot be found) TurboPlayer will stop at this element in LiveAssist mode, but will skip it and continue with the next element in automatic mode.

Passive mode means: TurboPlayer is prepared for operation and perhaps it follows the actions running on another TurboPlayer for the same service, but it does not play anything. Faderstarts and button starts / stops are deactivated normally. Automatic starts (= time starts, sequenced starts and relative starts) are not executed. Instead, TurboPlayer will continue to play any elements which were just running when the activation mode changed to passive, but it will not start any new element.

Normally, the passive mode is only used if you have multiple TurboPlayers for the same service, for example one in the main studio and one in the news studio. Or you might have multiple TurboPlayers to compensate for failures. In such a setup only one of the TurboPlayers should be active at each time. This behaviour is achieved by a communication between all running TurboPlayers via BroadcastServer, called active/passive-switching. (Hint: you have to activate this feature with the parameter "TurboPlayer \ AllowMultipleActiveClients".) If a TurboPlayer wants to become active, it asks the currently active TurboPlayer for permission. Depending on the setting "TurboPlayer\ TimeoutSetActivationMode" the asked TurboPlayer either becomes immediately passive, or it shows a dialog box, so the user can allow or deny the request. If the timeout is reached and the asked user did not react, TurboPlayer will close the dialog box and will allow the activation change.

There is a registry setting "InitialActivationMode" which defines the activation mode after startup and after changing the service.

1.4.4 Server connection mode / state

You must distinguish between the connection mode and the connection state. The connection mode is something which is intentionally set by the user and which defines the behaviour of TurboPlayer. The connection state is automatically set by the system and is a statement whether the server is online or not.

The following connection modes are defined:

- **Connected:** This is the standard mode. TurboPlayer connects to the BCS and tries to re-establish the connection, if it is lost. All tree operations are executed by the BCS, TurboPlayer is a "passive" client which only displays the data it receives from BCS.
- **Standalone:** In this mode the BCS connection is intentionally cut off. It can be used, when you run into server or network problems. You can go on working with the internal server functionality, which means, all tree operations are executed by TurboPlayer. If you switch back to connected mode, a very simply synchronization is done: the tree stored in TurboPlayer replaces the corresponding tree in the BCS. This means that any changes of the original tree in the BCS, which were performed by other clients during the standalone time, are lost ! This may sound unsatisfactory, but it is of utmost importance not to interfere with the running on-air client. Therefore, this sort of data loss must be accepted. In installations with multiple TurboPlayers for a service you should always make exactly one of them standalone, to continue playing during BCS problems. Don't make multiple TurboPlayers standalone ! Due to the resulting differences in the rundown data this will produce problems when reconnecting.
- **Rehearse:** In this mode the BCS connection is intentionally cut off, too. It can be used, to simulate the sending of a show. Like in standalone mode, all operations are performed internally. The difference is the switchback to connected mode. Now the synchronization is done in the direction BCS->TurboPlayer. The whole tree stored in TurboPlayer will be replaced by the corresponding tree of the BCS. Here data loss occurs too ! But that is the idea of the rehearse mode. If you have finished your experiments with the show, you want to throw away all changes (especially the send states) and re-start with the original data. You must be aware that this switchback needs a partial internal reset, stopping all playing elements and resetting the IO modules. (The other mode changes can be executed without interfering the playout.)

The following connection states are defined:

- **Disconnected:** This state does apply during startup (as long as the connection has not yet been established), if no valid BCS has been defined in the settings, or if one of the disconnected modes has been selected (standalone, rehearse). The server is not online and TurboPlayer does not try to re-establish the connection.
- **Disturbed:** The server had been online some time ago, but at the moment the connection is lost. TurboPlayer will permanently try to re-establish the connection.
- **Connected:** The server is online at the moment.

Beside of the connection mode/state you could also want to use TurboPlayer without a BCS at all. Typically, this is useful if TurboPlayer should be a pure cart / jingle player without any rundown. There is the command line switch **/nobcs** to achieve that TurboPlayer runs without a BCS connection.

1.4.5 General TurboPlayer modes

TurboPlayer defines some “modes”. These modes change the behavior of TurboPlayer in general. General means: they are applied independent of the rundown list. (Additionally, there are modes for each rundown list.) All of them can be set by the user by using mode buttons in the GUI or they can be changed and their current state can be queried in macros with the commands TP_ToggleMode() and CurrentMode(). The following other general modes are available:

Mode	Default	Description
Activation	Passive	Passive, LiveAssist or Automatic. See 2 chapters above. If a GUI automatically loads the current show the default is LiveAssist.
AdjustLoudness	TRUE	TurboPlayer can compute a playout gain value from the results of a loudness analysis stored in the metadata of an element. Therefore, it is necessary to enter target loudness values in the DigaSystem registry for the current service and to activate this mode. Otherwise, the playout gain as it is stored directly in the metadata (in field “Fade_Gain”) is used.
AssignOverlaid	FALSE	If this mode is active TurboPlayer automatically assigns the overlaid flag to played elements if a played element is completely overlapped by a previous playing element. This convenience logic can be helpful to play short drop-ins while long music elements are running because it avoids that the drop-in messes up with the sequence of music elements. For more information see chapter 6.3. “Automatic assignment of overlaid flag”.
AutoLoadJingles	TRUE	It is possible to assign a certain jingle group to a group or a show. TurboPlayer can automatically change to the specified jingle group if the <next> element pointer is positioned into a group/show and this mode is active.
BeatExact	FALSE	If this mode is active TurboPlayer will try to delay the sequenced start of elements so that the transition between the ending and the starting elements is beat synchronized. This works only if both elements have beat markers.
ButtonStart	TRUE	This mode must be active if it should be possible that the user can start or stop elements with a button. “Button” means here: a user button in the GUI, a key shortcut or a start / stop button in the surface of a hardware connected via IO module (e.g. mixer console).
EmbeddedStart	FALSE	Embedded starts are automatic starts of live elements on a metadata track during the runtime of a live recording element, triggered by subclips. This is part of the workflow “Reusing broadcast content” and is described in chapter 10.5. with the same name of BCSSTechManual.

Mode	Default	Description
ExtendedShowScanning	FALSE	If TurboPlayer has some shows loaded and new shows are created somewhere on the timeline, this mode determines which of these shows are automatically loaded by TurboPlayer. If these new shows are created after the main loaded show, TurboPlayer will always load these shows (this cannot be changed by this mode). But if TurboPlayer has a future show loaded and a new show in the range between <now> and the start of the loaded main show becomes available, TurboPlayer will load the new show as main show if this mode is active. Please be aware that activating this mode makes it impossible to explicitly load a future show and leave TurboPlayer untouched if a show that is more recent does exist. TurboPlayer will fall back to the more recent show. If any element is playing or if any element of the main show has already been played this mode is irrelevant because TurboPlayer will not fall back to a more recent show in this case. Last hint: scanning for new shows is done with a background timer running approx. once per minute.
FaderStart	TRUE	This mode determines whether the user can start or stop elements via fader of a connected mixer console.
FaderStopsPFL	TRUE	If this mode is active opening a playout fader stops any prelistening running on the fader with PFL (pre-fade-listen) being activated.
FreeShowlist	FALSE	By default, TurboPlayer does not only load the main show (the one the user did select) but additional shows of the past and the future - depending on setting “ShowInterval”. This ShowInterval is automatically advanced to the future during playout. If FreeShowlist mode is active the user can select any combination of (consecutive or not) shows to be loaded. TurboPlayer will no longer change which shows are loaded or advance anything automatically.
Ghost	FALSE	This is a special mode for a wrong-way driver situation (such a driver is called “ghost driver” in German). If this mode is active, TurboPlayer will continue to play all elements, which are already running if the mode is activated, but it will no longer start any new elements. If playout stops the presenter / talent can read the traffic announcement. If the announcement is most urgent stopping or pausing the playout should be preferred.
InsertFiller	FALSE	Fillers are short elements in special jingle groups, which can be used by TurboPlayer to automatically fill gaps, which are detected during playout. In addition to creating at least one jingle group with filler elements and assigning a filler jingle group to the shows it is necessary to activate this mode.

Mode	Default	Description
IgnoreMarkOut	FALSE	If this mode is active TurboPlayer will not stop the playout of elements at mark-out but instead continue until end of take or until the element is stopped manually. This is intended for manual playout of transitions, which were produced for automatic playout. Hint: there is also a flag with same name and same functionality for each element. Here the general mode, which affects all elements, is meant. Please check also the special TP parameter “SequencedAtIgnoredMarkOut” described in the parameter description below.
OnAir	TRUE	This mode can reflect the OnAir state as some mixer consoles support. There is only one thing TurboPlayer changes with this mode: it is possible to configure two different tracks into which TurboPlayer writes the protocol of a rundown, depending on the OnAir mode.
Pause	FALSE	For historical reasons TurboPlayer handles the pause functionality in a special way: if the pause mode is active stop commands (by closing a fader or pressing a stop button) are no longer executed as a stop but instead as a pause. The next opening of a fader or executing a start command will trigger a restart of the corresponding element at the paused position.
PlayFadings	TRUE	As standard behavior, TurboPlayer plays in / out fades or fade curves like scheduled – by controlling motor faders or by controlling the playout level of MultiPlayer. This might be undesirable if a show scheduled for automatic playout is played manually. In this case, the presenter / talent wants to control all levels by his own. Therefore, playing fades can be switched off by deactivating this mode.
SelectSources	TRUE	This mode activates or deactivates the automatic selection of mixer console sources by their name. Normally, all needed mixer sources should be selected. This mode is intended to switch the source selection off, e.g. for reconfiguring a mixer console.
ServerConnection	Connected	Connected, Standalone or Rehearse. See 1 chapter above.
SyncStart	FALSE	This mode is deprecated! Since version 6.1 of the BCS modules a dedicated start mode “SyncStart” does exist. It replaces the usage of this mode and the TurboPlayer mode should no longer be used (and is no longer needed). See chapter 5.3 below. Here is the old text with a description of the TP mode: This mode must be activated if you want to make use of the SyncStart feature. It allows starting multiple elements in different rundowns (or even on different TurboPlayers) at the same time if one of them starts. Therefore, it is also necessary to define one arbitrary metadata field as SyncStartField (see chapter “Relative and synchronized / slave starts“ and parameter description below). The content of this field must be identical for all elements. For example, you could use the field “Music_MusicID” and start an audio element and the corresponding video element with the same music-id at the same time.

1.4.6 Rundown lists

Typically, the term “rundown” refers to a main schedule only: a list of elements played one after the other. TurboPlayer extends this term. It can load and treat multiple lists (rundowns) simultaneously. To distinguish these lists, the following names are used:

- **Show**: This is the main rundown. This is the rundown you change with the show combo box in the GUI. The content of other lists may depend on this main list. The list is called “show” (singular only), though there might be multiple BCS shows loaded to this list. Typically, this is true, because you will let TurboPlayer load at least the current and the next show in this list. Start attributes of elements (like sequenced) will be executed for this list.
- **Jingles**: One of the BCS jingle groups can be loaded to this list. Depending on the settings this can be done automatically when the show in the main list changes or when the “next” pointer changes the group. Start attributes of elements (like sequenced) will not be executed for this list, because this is not a real rundown, but a container only.
- **Stack(n)**: n=1…15. These are additional multi-purpose rundown lists. You can use stacks in different ways – see the description of the list modes below.
- **Prelisten**: This is an internal list, which is used for the prelisten feature, see below.
- **Deleted**: An internal list for elements, which are deleted from one of the other lists while they are playing. TurboPlayer will continue to play such an element and therefore it will move the needed data of the deleted element to this special list.

An important parameter for each rundown list is the number of preloaded elements, found below the key "RundownLists\<List name>". First, you should know that the rundown kernel does not hold a list of all elements, which are in a show or a jingle group. Only the tree manager stores the full information about all groups and elements. It is one of the tasks of the tree manager to build a linear list of the next n elements (for each rundown). Only this list is transferred to the rundown kernel.

The kernel will try to preload all elements it sees. For audio elements preloading means, the MultiPlayer is requested to open one of the corresponding audio files and to buffer the first n seconds of audio data. If no audio file is available, the state of the element will change to “load failed” which is visible by the element color in the GUI. TurboPlayer will regularly try again to load these failed elements. The retry interval is between 10 seconds and 3 minutes, depending on the time until the element should be started. Preloading of external elements means: the corresponding mixer source (see below) is selected. If this is successfull, the element is seen as “preloaded” too.

The size of each list can be configured with the “PreloadedElements” parameter. The sum of all list positions of all rundowns must not exceed the number of available MultiPlayer slots – something TurboPlayer checks during startup. If there are more positions than slots, a red line is printed to the error log and startup fails. Pay attention: the number of slots for the rundown must even be higher than the number of preloaded elements ! This is necessary because TurboPlayer performs preload operations faster than the unloads. This happens e.g. if an element is replaced: first, the new element is preloaded and the old element is moved to the list of deleted elements. The unload for the deleted element is performed later. Therefore, each rundown should have some reserve slots (approx. 10% but at least 3 slots). Since version 5.5 the registry parameter “TurboPlayer\CheckSlots” is available. If it is set to FALSE, TurboPlayer will not check the number of available MultiPlayer slots during start – whether the sum of configured preloaded elements exceeds the number of available slots. This setting can be used to have more preloaded elements than MultiPlayer can handle. This is only usable, if the exceeding elements are external or live elements without media file which must be handled by MultiPlayer.

The value "CurrentElements" is an additional parameter to limit the number of elements with an absolute start, which can supersede consecutive current elements - see the parameter description for more information.

1.4.7 List modes

The two most important list modes are: StackMode and RundownMode**.**

The **StackMode** can be assigned to the stack rundowns. It specifies in which way the list is filled:

- **Clipboard**: The content of the list is kept in memory only. There is no link to any node on the BroadcastServer. Initially a list in clipboard mode is empty and it can only be filled by drag/drop operations in the GUI. Changes of elements will not be saved.
- **Free**: The content of the list is linked to a node on the BroadcastServer. Nearly all nodes of the current program/service can be used; templates are excluded. The list is filled by selecting one of the possible nodes in the GUI. The content is independent of the current show and does not change when the current show advances. For example, you can load any general or personal pool, a pre-production or an additional jingle group.
- **Track**: The content of the list is loaded from a specific schedule track of the loaded shows. The track number is assigned in the registry. The user cannot change the track number, nor can he assign different nodes to the stack. The content automatically advances in time together with the shows loaded in the main show list. Additional tracks are used for regio, data or video tracks, typically.
- **Endpoints**: The content of the list is automatically being generated by TurboPlayer by picking elements from the show list, which have a certain distribution endpoint (region). The endpoint/region need to be configured for the list. The content in this mode will be a copy of some of the elements of the show list. No changes to such a list by the user are possible. For the show rundown this stack mode is not allowed, because the show rundown is always the source for all the distributed elements. For more information, please see chapter 5.4 Using EndpointStories for regionalization.

It is possible to change the stack mode for the show rundown, too, but this is only recommended for special usage (e.g. if TurboPlayer is used with pure clipboard lists).

The **RundownMode** can be assigned to any list and specifies in which way the list is played:

- **Sequenced**: The rundown is played from the beginning to the end exactly in the order the elements are scheduled. Of course, there are exceptions, like time- or mute starts, but essentially the next pointer is given by the system and the user cannot start a different element than the next one. When an element has been played, its send state is set to "sent" and the element cannot be played again. In case an element which is not the next one has to be started (e.g. a timed start) all unplayed elements above the started element are skipped. This mode is used for shows which must be played in the given order.
- **Once**: As the name says, each element can be played once. After it has been played, the send state of each element is set to "sent" and the element cannot be played again. The difference to the previous mode is that the user need not play the elements in the given order. He can set the next pointer to any element, which is (still) playable. This mode can be used for shows in which the user should have the freedom to play each element when he likes to. Nevertheless, the system obeys the start attributes and something like "sequenced" or "relative" starts are executed. Be warned that you can get multiple running sequenced sub-chains in this mode. (E.g. element 1 and 4 are started by the user and elements 2 and 5 are sequenced. Then you have two sub-chains running independently.) In automatic activation, this rundown mode results in the same behavior as mode "Sequenced".
- **MultiSequenced**: In this mode, the user can select any element he wants to start. After starting an element, sequenced or relative elements below are started automatically. In contrast to mode “once” played elements do not get the send state “sent”. Therefore, each element can be played as many times as the user likes. Time starts are executed. Like in mode “once” you can get multiple running sequenced sub-chains. This mode should not be used with automatic activation.
- **Random**: Each element can be played whenever the user wants to play it. Start attributes of the elements are ignored, no sequence is played. When an element has been played its send state is reset to its original state, typically "planned", and the element can be played again. The user can drag an element to a player assigned to the rundown to preload the element in the corresponding channel. This mode is used for jingles typically.

There are additional parameters of a rundown. They can be set in the registry below "RundownLists\..." and define the behaviour of the corresponding list. It is also possible to change all rundown modes with the macro command TP_ToggleListMode (see below). Here is a short description of the available rundown list modes:

List mode	Default	Description
AllowRearrangement	TRUE	This property can be deactivated in order to disallow user actions, which change anything in the sequence of the corresponding rundown list. This means: insert, delete, copy or move operations are no longer possible. Nevertheless, it is still possible to change the metadata of individual nodes in the rundown.
AutoDelete	FALSE	If AutoDelete is active, TurboPlayer will delete played elements from the rundown. This can be used for stacks or clipboards: an element can be played once and need not be deleted by the user after playout. Hint: with the registry parameter the behavior can be changed so that elements are appended to the end of a rundown instead of being deleted. This behavior is not available via the general mode and therefore cannot be set via macro command.
AutoPrepare	FALSE	If AutoPrepare is active TurboPlayer will automatically prepare the next n elements into the available channels / players. This is needed if the presenters / talents are used to see the next elements assigned to certain channels so that they can start an element by opening the corresponding fader. The number n of prepared elements is deduced by the number of free channels and can also be configured by the channel parameter “UseForAutoPrepare”.
ClearOnReset	TRUE	This mode is applied only for stack rundowns. If the mode is deactivated, TurboPlayer will not remove all elements from the rundown if a general reset is executed (which is normally triggered by the user by loading a new main show).
DistributionEndpoints Regions Region	<empty>	This parameter applies to a stack in stack mode “Endpoints”. It defines the distribution endpoints, which should be handled by this stack. The value can be a comma-separated list of endpoints names, but each name must be one of the subkeys of the registry key Common\DistributionEndPoints. As this feature is mostly used for regionalization, you can instead use the parameter name “Regions” or “Region”. This parameter **cannot** be changed via macro command.
PreloadAroundCursor	FALSE	This mode is only useful for a rundown being operated in a RundownMode in which the user can select the <next> element by changing the cursor position (Once, MultiSequenced, Random). In case the number of elements in the rundown can be higher than the number of available MultiPlayer preload slots, it is not sufficient to preload the first n elements of the rundown (that’s the standard behavior). If the user did select an element outside of the preload range, the element would not be startable. Therefore, this mode can be activated. TurboPlayer will adjust the range of preloaded elements around the position of the current cursor element. In the registry it can be configured how many elements before and behind the cursor element should be preloaded. This is done by a rundown list parameter which is called “PreloadAroundCursor”, too. In this table we talk about

		the rundown list mode which only activates or deactivates preloading around the cursor position.
SingleElement	FALSE	This special mode allows restricting a rundown so that the rundown cannot hold more than a single element. This mode can be used for restrictions of clipboard stacks.
Track	0 (show) n (Stack n) <none>	This property is not only active/inactive but contains a number. It is the track of a show being loaded into the rundown if the StackMode is “Track” and if the user really loads a show which then can be split up by distributing the tracks of the show on the corresponding rundown lists.

1.4.8 Logical Channels

In TurboPlayer “logical channels” are distinguished from “physical lines”. A logical channel is used for playing a single stereo audio stream. It appears in some places:

- It corresponds to one of the Windows MME / WDM audio devices which are displayed in the sound and multimedia properties of the Windows control panel. With MultiPlayerV3 you can also use ASIO and a logical channel corresponds to a range of ASIO streams.
- It is specified in the MultiPlayer settings in the Windows registry in key “Player(n)\Channels” in the values “AudioDevice”, “WaveOutDevice(n)” or “MmeDevice(n)”.
- If your multi-channel audio card supports an internal mixer, a channel corresponds to the “In” part.
- It corresponds to one visible player in the TurboPlayer GUI.
- Each logical channel is assigned to exactly one rundown list. (This means TurboPlayer does not support the use of a single channel for playout and prelistening, two different channels are necessary.)

On a logical channel only a single element can be played at each time. The number of channels you need depends on the rundown you are playing. For example, if you want to play a crossfade with an ending and a starting element and with an additional overlaid presentation or jingle, you need at least three logical channels. If there are less than three free channels, TurboPlayer will not be able to play the crossfade as intended. Instead, it will play the elements sequenced.

Each channel is of type audio or video. This is defined in the channel settings and must be identical to the real channel types in MultiPlayerAV. If you do not use video playout, the channel type doesn't bother you because "audio" is the default.

Channel restrictions:

Normally TurboPlayer will use the next free channel to play an element. There are some parameters which can limit the possible channels depending on element properties. Three restrictions are available so far: you can set the allowed classes for a channel, you can set the list positions and you can restrict the usage for neutral elements in an EndpointStory. First, the channel-class assignment is explained.

Normally, all elements can be played on all channels of a rundown list, independent of the class of the elements. Sometimes it is desirable to define that elements of a specific class can be played on a special channel only (or on any other channel with the same class assignment). Typically, this is used to assign commercials to a specific channel, which has probably an extra physical line with a special audio processing. This can be achieved with the “Class” parameter of a channel.

In LiveAssist mode elements with a class, for which a specific channel exists, cannot be played on other channels. Just in the same way, elements with a different class cannot be played on the special channel. If the presenter tries to ignore this rule, the behaviour depends on the general parameter “TurboPlayer\ShiftInvalidChannelStart”. If it is TRUE, TurboPlayer will use one of the correct lines automatically – something which can be very astonishing, if the presenter does not use this feature with intention. If the parameter is FALSE, an error message is displayed. (Hint: there is an additional parameter "ShiftToLiveChannel" which allows to start live elements with a wrong fader.) The behaviour becomes different in automatic mode. Now the channel-class assignment is treated as a recommendation only. If there are enough free channels and the corresponding lines are open or can be opened, the assignment is taken into account. But if there are closed lines, the remaining lines are used as best as possible. The extreme case is that there is only a single line available. Then, of course, the class assignment is ignored and all elements will be played on this line.

The second restriction is a list position-channel assignment. For a normal rundown this feature is not useful, but it can be used for jingles. With the channel parameter “ListPosition” you can tell TurboPlayer to use specific channels for some positions in the jingle list. For example, jingles 1-4 could be played on channel 4 only, jingles 5-8 on channel 5 and so on. An automatic shifting of invalid starts by the presenter like for the class-assignment is not done.

The third restriction is the parameter “NeutralRegionOnly” (or alternatively: “NeutralEndpointOnly”). This refers to so-called “neutral” elements within an EndpointStory. An EndpointStory can contain all elements for a regionalized leg, including one or more neutral elements, which are not played for a specific region but for the regular output for all listeners. Depending on the audio routing during the regionalized leg it might be necessary that such a neutral element must be played on a specific channel, which is reserved for that purpose. Playing a regular element on such a restricted channel or vice versa a neutral element on a regular channel would mean that the wrong listeners would hear the wrong content or they would not hear anything at all. Therefore, this is a hard restriction, which is always obeyed and also cannot be avoided by the user. A user has to start elements on the right channel type.

There might also be a temporary channel restriction: there are registry parameters which make TurboPlayer play consecutive elements of a certain class or media type on the same channel, meaning: all elements are played on the channel on which the first element was started. And in addition there is the "joined" flag of an element which has the same result: the channel is maintained during a sequence-chain.

1.4.9 Physical lines

A line is the carrier of a stereo audio stream which is available at the output connector of the audio card or the corresponding breakout box. With the typical playout audio cards like Lynx or Mixtreme it is possible to mix logical channels onto physical lines. That means a line is the “out” part of the internal mixers.

In TurboPlayer multiple channels can be assigned to a single line. (Of course, these settings should be identical to the real mixer settings of the audio card.) In this case TurboPlayer is not able to perform fadings via the mixer faders. The internal fading capability of the MultiPlayer must be used instead. On the other hand, opening and closing a line, whenever at least one of the assigned channels play, is possible.

If you are confused by channels and lines, remember the assignment: a channel is a GUI player, a line is a mixer fader.

1.4.10 Faders

A fader is a part of the mixer console which controls the level of one stereo line. It is assumed that a physical line is connected individually to a fader of the mixer console. Therefore, fader-specific settings are specified with the line parameters.

Two properties of a fader are important: does the fader report if the line is opened or closed (the line state) and: is it a motor fader which can be controlled from TurboPlayer ? These two properties are defined with the mode parameters in the registry. The following values are possible:

- None: The mixer console cannot report the line state and there is no motor fader. Also used, if there is no fader at all.
- Passive: The mixer console can report the line state, but the line cannot be opened or closed by TurboPlayer.
- Active: The mixer console can report the line state and the line can be opened or closed by TurboPlayer. This mode should be used when the user really moves a fader to open/close the line.
- Active/OnOff: The mixer console can report the line state and the line can be opened or closed by TurboPlayer. This mode should be used if there is no fader at all or if the fader is irrelevant to TurboPlayer, meaning: the user will press On/Off buttons to open/close the line.
- Active/Variable: The mixer console can report the line state and the fader can be moved by TurboPlayer.

You should note two things: first, it is not supported that a mixer console has motor faders but does not report the line state. Second, the report is only about the line state (opened or closed), not the fader position. A report of the fader position is not evaluated so far.

1.4.11 Line and channel types

So far only internal lines and channels have been mentioned. There are some additional additional types:

- Internal lines: These are lines/channels which are used for the playout of audio or video elements via MultiPlayer. Rundown elements with mixer source "System" are played on these lines. In the registry internal lines have subkeys for the assigned channels.
- External lines refer to other audio sources like CD player or microphones. These lines are used for external elements in the rundowns. Elements with a specific mixer source (like "Mic1") are played on these lines. In the registry external lines do not have subkeys because there are no audio channels assigned to them. Nevertheless, TurboPlayer holds “virtual” channels, one for each external line, so it can treat internal and external lines in the same way. These virtual channels are counted consecutively and start with the highest real channel number plus one. External lines are directly related to mixer sources, see below.
- Virtual lines are used by TurboPlayer for playing live elements, that means they have the mixer source “Live”. For such elements no audio is played, no line is opened, no fader is moved. There is only a counter for the playtime and controls assigned to the element are being executed. Like the external lines each virtual line gets exactly one virtual channel assigned. These virtual channels are counted consecutively and start with the highest virtual channel number for external lines plus one. Virtual lines do not appear in the registry, there are always 10 (32 since version 5.1) of them present.
- Remote channel: There is only a single remote channel which is not assigned to any line. This channel is used to display information about elements, running on other TurboPlayers. In the GUI you can assign a “remote player” to this channel and you will see the last element started on another TurboPlayer. TurboPlayer internally generates information about the runtime and remaining playtime, but the accuracy depends on the time synchronization between the computers, of course. The remote channel has the fixed channel number 9999.
  Since version 6.0 of the BCS modules, it is possible to have multiple remote channels with real states and time information being displayed. These channels have numbers, 10000, 10001, … For more information, please see the chapter "Play Info" below in the document.

To make this quite complicated description clearer, here is a typical example:

Rundown / type	Line	Channel
Main 1	Internal line 1	Internal channel 1
Main 2	Internal line 2	Internal channel 2
Main 3	Internal line 3	Internal channel 3
Jingle 1	Internal line 4	Internal channel 4
Jingle 2		Internal channel 5
Jingle 3		Internal channel 6
Prelisten	Internal line 5	Internal channel 7
Microphone 1	External line 6	External channel 8 (=7+1)
Microphone 2	External line 7	External channel 9
CD 1	External line 8	External channel 10
CD 2	External line 9	External channel 11
Live 1	Virtual line 10 (=9+1)	Virtual channel 12 (=11+1)
...	...	...
Live 10	Virtual line 19	Virtual channel 21
Remote	-	Remote channel 9999

With the parameter "LiveMediaTypes" you can force TurboPlayer to treat elements with specific classes and/or media types like live elements and play them on the virtual channels. This works in two ways: e.g. for audio classes (which are internal elements normally) but also for the text class (which is not played at all normally). But be aware that "Live" and "External" are two different things. You cannot declare a class or media type as live and expect that these elements will be handled as external elements with the mixer source logic (see the next two chapters).

One additional hint: do not make "Info" elements to live elements. Info elements are designed as "information for the presenter" and there is a lot of special code for handling info elements, which makes it problematic to misuse info elements as playable elements.

1.4.12 Mixer sources

Sometimes mixer consoles support a selective assignment of mixer sources (all inputs) to faders. This is needed, if there are more possible inputs than available faders. TurboPlayer supports the automatic selection of mixer sources for external elements. We distinguish three different types of mixer sources:

- Internal: These are the mixer sources for the internal audio / video lines. They must always be available and are selected during startup or with every reset. A dynamic selection / deselection is not supported.
- External permanent: These are mixer sources for external elements. They must always be available and are selected during startup or with every reset. A dynamic selection / deselection is partly supported by switching on / off the general “select sources” mode. “Permanent” means, these mixer sources are selected, if they are used by any element in any rundown or not.
- External sporadic: These are mixer sources for external elements. TurboPlayer will select them whenever an external element, which uses the mixer source, is within the preload range of any of the rundowns. When all such elements have dropped out of the preload range, the mixer source will be deselected automatically. That means the selecting is treated exactly like preloading / buffering for audio elements, which is also visible in the state (color in the GUI) of an element. These mixer sources are selected / deselected too, when you change the general “select sources” mode.

Mixer source groups:

External mixer sources can / must be grouped, each source must be of one group exactly. Typical examples are all microphones or all CD players. Groups are a useful feature for scheduling, because when the rundown is generated, the exact source is rarely known. The person who is scheduling the elements can enter the groups, e.g. “CD” though he does not know which CD player will really be used. The presenter will know the player and he can change the group into a real source. Or he can simply open a CD fader when the “next” pointer of TurboPlayer is on an element with group CD. TurboPlayer will recognize the relation, will change the source automatically and will start playing (playtime only) the element.

Sub mixer sources:

Each mixer source can have a list of assigned sub mixer sources. Only one of the sub sources can be used at any time. The idea is to have spare sources, in case one becomes unavailable or out of order. For example, the mixer source "MainMic" could be the microphone of the presenter. Of course this mic is very important. Therefore a spare mic exists. If you have both connected to your console, you could use two sub sources, a "MainMicStd" and a "MainMicSpare". Normally, MainMicStd is used, but in case of a problem you can change the sub source with a macro (see below) and MainMicSpare would be used instead. Be aware that the sub sources are only known to TurboPlayer. They do not appear anywhere in the schedule list and are not known to DigAIRange.

All external mixer sources, groups and sub sources must be declared in the registry below “TurboPlayer\MixerSources”. On the first level groups are defined, one level below are the sources, another level down are the sub sources. See the parameter description for more information. Internal mixer sources are not defined in this key. They are defined with the settings of the corresponding IO module.

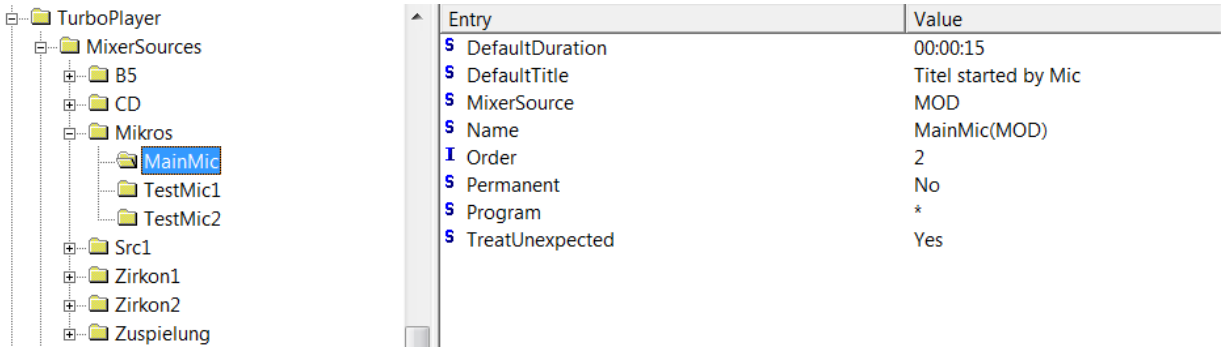
Simulated mixer sources:

In many cases you have a mixer console which does not support a source selection. Or the console has an interface which does not allow the transfer of source names (e.g. GPIO). You can use external lines and TurboPlayers logic with unexpectedly started external elements, nevertheless. But you have to simulate the source selection which would be handled by the console. You can either assign mixer sources to lines permanently in the registry - see the parameter: TurboPlayer\Lines\...\MixerSource. Or you can do the assignment with the macro command TP_SelectMixerSource. In both cases, you tell TurboPlayer that a specific line (e.g. line 9) corresponds to a mixer source (e.g. Mic1). TurboPlayer cannot check if this is correct, but it will handle line 9 as the line for the external source “Mic1”.

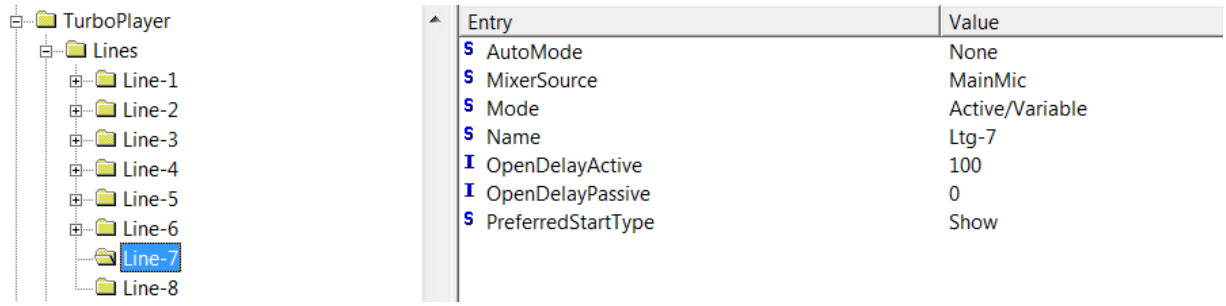
If you use mixer sources with such a console, you should also deactivate the automatic selection of mixer sources by TurboPlayer with the parameter: TurboPlayer\MixerSources\AutoSelect=false. This prevents that TurboPlayer tries to call the console when an element with an external source appears in the preload range.

Example configuration:

Here is a screenshot of the registry settings for a mixer source for the main microphone:



Here is an example how the defined mixer source is used for a line:



In the settings of the IO module you have to generate new settings for a "MoveFader"/"FaderMoved" (2 times: out and in) for the corresponding line (here line 7).

1.4.13 Playout categories

If you have read the chapters about line and channel types and about mixer sources above, you will already have noticed that TurboPlayer supports different playout categories. We can distinguish internal, external and live elements. Only elements with one of these playout categories are handled by RundownKernel and are “preloaded”. TreeManager filters out all other elements.

In this chapter we explain which metadata and parameters must be set in context with playout categories. (Hint: this chapter was introduced with version 5.x of TurboPlayer and describes the behavior of this or newer versions. Older versions behave slightly different, though most things still apply. There are some remarks below regarding old versions.)

Playout categories

Internal	Internal elements are elements with a media file (audio, video, graphic) which are played by MultiPlayer and via internal lines. In case of a mixer console with mixer sources the corresponding sources are selected during startup of TurboPlayer and must permanently be present.
External	External elements are elements without a media file, but which have a specific mixer source. They are used to control any audio or video content which is not stored in files, but which is delivered on-time during playout. Example: audio from a CD player or a microphone. For these elements the playout is only simulated: TurboPlayer can display a play- or remain time in the player window like for an internal element, but this is only a fake. TurboPlayer has no possibility to know whether the opened external line really has an audio signal on it. The term “preload” still applies: an element is handled as preloaded if the specified mixer source is available at the mixer console and has been assigned to one of the external lines.
Live	Live elements are elements, which are not really played physically. Like for external elements there is a fake playout with play- and remain time. Compared to external elements there is even less action because TurboPlayer will not control the mixer console but only simulate a playout. A typical use is for control elements, which execute macros during their simulated playout. Or you can use this playout category for text elements if the text is read by a speaker and TurboPlayer can assist by displaying a countdown. The term “preload” does not have any meaning for these elements; they are always treated like being preloaded.

Relevant registry parameters

LiveClasses	This is a deprecated parameter which was important before the introduction of the media types in May 2007. Old versions of TurboPlayer support this parameter but versions >= 5 do not evaluate it. In any case you should only use “LiveMediaTypes”, even if you are running an old TurboPlayer.

LiveMediaTypes	This is a parameter in key “TurboPlayer”. You define all media types TurboPlayer should treat as live elements. Possible media types are: Unknown, Audio, Text, Graphic, Video, Live, Line, Control, Command, Info, Multiple. The default for this parameter is: “Live, Control”. If you change this parameter don’t forget to add the two default media types “Live, Control” – they realize what is normally expected from TurboPlayer. Typically, you should only add the media type “Text” to the list if you want a virtual play counter for text elements. Only in very special cases or test situations it will be useful to declare any of the other media types as live. Hint: do not make "Info" elements to live elements. Info elements are designed as "information for the presenter" and there is a lot of special code for handling info elements, which makes it problematic to misuse info elements as playable elements. One additional hint: In versions < 5 the default for this parameter was empty but TurboPlayer handled elements with media type “Control” as live elements nevertheless.

Relevant metadata fields

SendState	If SendState=Placeholder an element is ignored and TurboPlayer does not preload it.
FileState	Elements with FileState Virtual or Requested are ignored because these states mean that a media file has been announced but is not available yet. Invalid is ignored because the analysis of the media file did detect an unusable content. Elements with other file states are handled. FileState External is intended to be used for external elements.
Source	The field Source is an important field in order to make an element internal: Digas and Harddisk are treated as internal. All other sources are intended for external elements.
MixerSource	The field MixerSource is the most important field for external elements. An external element always must have an assigned mixer source (a specific one or a mixer source group). For internal and live elements this field should be empty (though it’s ignored for these elements). TurboPlayer has a legacy function: if this field has the content “*” (which is displayed in GUIs as “<Live>”) and the source field is empty, the element will be treated as live element – independent of any other metadata fields or the LiveMediaTypes setting.
MediaType	Media types Audio, Video and Graphic are reserved for elements with a corresponding media file. They are handled as internal elements. Media type Text is reserved for elements with a text file. These elements can be handled as live elements, or they can be ignored – depending on the setting LiveMediaTypes. Media types Live and Control should be used for live elements. Media type Line is intended for any sort of external element. Media types Info, Command and Multiple are ignored.
Class	The class is not evaluated. Only in old versions < 5 it was possible to change the behavior depending on the class by changing the now obsolete setting LiveMediaClasses.
Time_StartType	Elements with start type Unused are ignored.

Time_StartMode	Elements with start mode Unused are ignored.
Time_DoNotUse	Elements with a set “DoNotUse” flag are ignored. Usage of this flag must be activated. This is useful only in some very specific situations and it should be done with great care (see parameter description).

Mandatory / recommended metadata for playout categories

Internal	SendState	=	**Planned, Cleared**
	FileState	=	**Existing, Accessible, Mirrored, Recording, Missing**
	Source	=	**Digas, Harddisk**
	MixerSource	=	**<empty>**
	MediaType	=	**Audio, Video, Graphic** (1)
	Class	=	Any class
	Time_StartType	=	Any, **except Unused**
	Time_StartMode	=	Any, **except Unused**
External	SendState	=	**Planned, Cleared**
	FileState	=	External, NoFile
	Source	=	CD, Microphone, Telephone, Tape, DAT, Minidisk, Satellite, Line
	MixerSource	=	**A valid mixer source or mixer source group**
	MediaType	=	Line (1)
	Class	=	Any class
	Time_StartType	=	Any, **except Unused**
	Time_StartMode	=	Any, **except Unused**
Live	SendState	=	**Planned, Cleared**
	FileState	=	NoFile
	Source	=	<empty>
	MixerSource	=	<empty> or * (2)
	MediaType	=	**Live, Control** (1)
	Class	=	Any class
	Time_StartType	=	Any, **except Unused**
	Time_StartMode	=	Any, **except Unused**

Metadata printed fat specifies mandatory values, all other values are only recommended.

(1) The media types are applicable as listed if you do not define other “LiveMediaTypes” in the corresponding parameter. If you change the parameter all listed media types identify a live element. Then it is mandatory to use the specified media types for all live elements.

(2) The special value * together with an empty Source field indicates a live element. This is only a legacy combination. The field content * is displayed in the GUIs as “<Live>”.

1.4.14 Prelisten and PFL

Prelisten and PFL are complicated stuff. We are going to explain many details now, but first we must point out a principle difference:

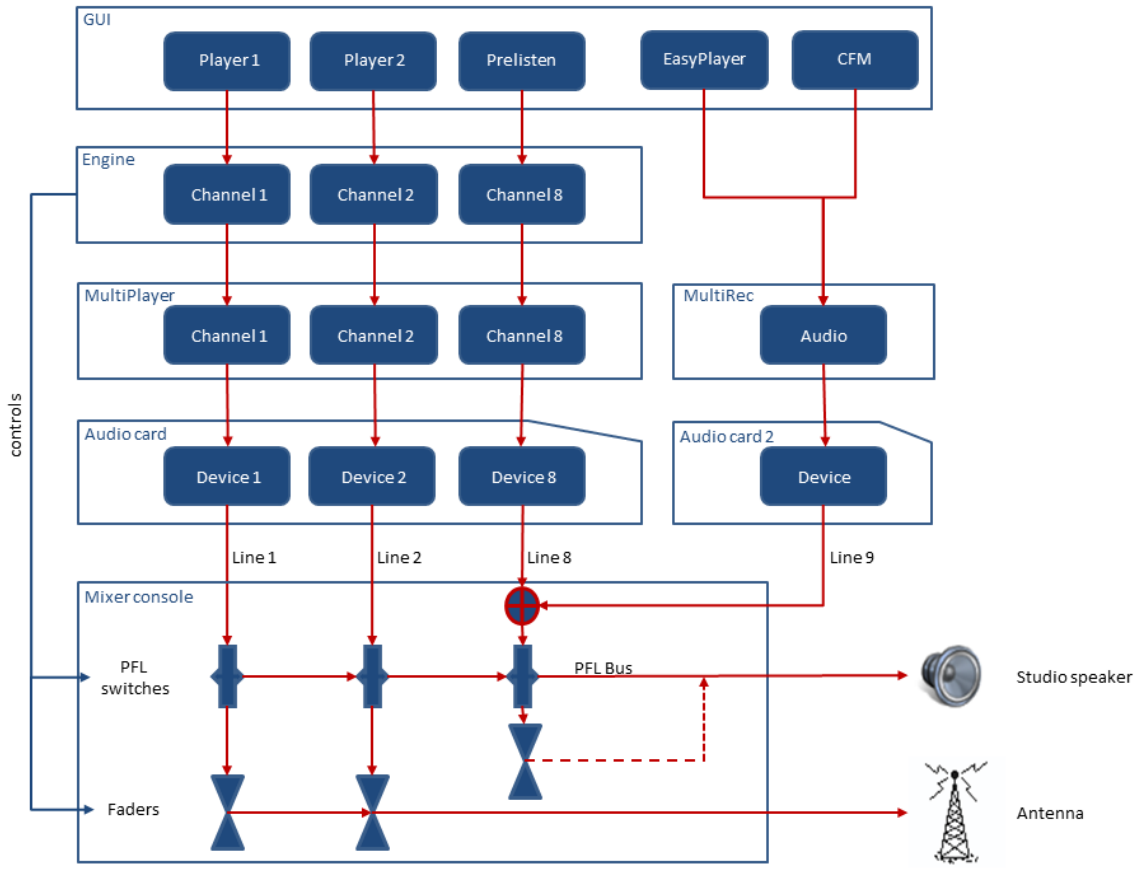
What is prelisten, what is PFL ?

The administrator should distinguish between prelisten and pre fader listen (PFL), though it seems to be only a single function to the ordinary user. If the user wants to listen to an element, he activates one of both fuctions, which is done with the same action in both cases. For example: the user presses the middle mouse key on an element. If the element is an internal audio element, TurboPlayer will perform a prelisten, if it is an external element, TurboPlayer will activate PFL for the corresponding line – if this feature has been configured.

Prelisten is a feature of TurboPlayer. It means: play an internal audio element with the purpose to give the presenter a chance to listen to the element on his head- or studio speakers which he later is going to play on-air. In contrast, PFL is actually a feature of your mixer console. When activated it performs the following action: Route the signal/input of the channel on which PFL is activated to a separate bus within the mixer console. The signal is sent to the bus pre-fader and usually “post-processing” (after Insert/EQ/Dynamics). This bus is called “PFL bus” and is usually connected to a separate monitoring source (alternate set of speakers, the presenter’s headphones, etc). Thus, the presenter is able to listen to the signal as if it was on-air but with the signal on a separate bus which is only audible in the studio. Once PFL is deactivated again, the signal will be routed back to the on-air bus and can be sent on air as usual.

Of course, your mixer console must provide a PFL feature, otherwise it is not usable from TurboPlayer. (Prelisten is always available, if you have at least one free fader / line / channel which can be used as a dedicated prelisten channel / line.)

For the further discussion here is a picture which illustrates an example of the audio routing as far as prelistening is concerned:



Prelisten without PFL

Let's assume you have a mixer console which does not have a PFL feature or that you don't want to make use of it. This is the simplest case. You need to reserve one of the channels MultiPlayer provides for prelistening. Let's assume MultiPlayer has 8 channels. If you want to use channel 8 for prelistening, all you need to do is wire the output of channel 8 to an alternate monitor source (as mentioned above, a different set of speakers or a dedicated pair of headphones, etc). Apart from that, channel 8 is just as any other channel.

A PFL mechanism is not necessary, because the audio signal of channel 8 is hard wired to the speakers. If a fader is available for the prelisten channel it can be used to start or stop prelistening. And if the fader can be controlled by TurboPlayer it can be opened or closed automatically, too. Therefore, the fader is used like the on-air faders and PFL is unnecessary.

This scenario is visible in the picture above: Channel 8 represents this concept. No PFL switch is necessary (but drawn in the picture) and the audio signal is routed via the fader directly to the PFL bus (see dashed line) which is hard wired to the studio speaker.

PFL for external mixer sources

PFL is an essential feature if you want to listen to external mixer sources before opening them on-air. An external source is an audio signal connected to the console which is not coming from TurboPlayer – e.g. a microphone (see the extra chapter). If you press the PFL button on the mixer console the input from the source will be audible on the studio speakers via a PFL bus. As this function is a feature of the console, TurboPlayer is not needed here. Nevertheless, TurboPlayer can control the PFL switching if the console provides an appropriate interface. TurboPlayer will activate PFL for an external mixer source when performing a middle mouse click on an external element in TurboPlayer's rundown.

This might seem identical with the behavior above, at least to the user, but it's fundamentally different. Prelisten is not used in this case because TurboPlayer is merely activating a function within the mixer console and no audio is present on the prelisten channel of MultiPlayer. This is also the reason why this scenario is not visible in the picture above (because it is a PFL workflow, and the picture only shows prelisten).

Prelistening with PFL switching

TurboPlayer can control the PFL switches of the console (if present) when performing prelisten. This is handled like opening or closing a fader, but instead PFL is activated before prelisten starts and deactivated after prelisten ends. This also works the other way: if the user activates PFL on the mixer console TurboPlayer starts prelistening (typically the last selected element is played) and stops if PFL is deactivated on the mixer console or within TurboPlayer. In this case a dedicated prelisten channel / device is still being used but prelistening is coupled with a PFL switching for the prelisten line. The described case can be seen in the picture above, also in column 8. Please note that this time the PFL switch routes the audio signal to the speaker and not the fader as in the previous example. Pay attention that activating a PFL switching does not automatically deactivate the move-fader commands! If you want to use PFL **instead** of MoveFader set the mode parameter of the corresponding line to "None".

Prelistening via on-air channels

TurboPlayer also supports prelistening via on-air channels. Of course, TurboPlayer cannot simply play audio on these channels because this would be audible on-air. In order to avoid this, PFL switching must be used for prelistening. As in the previous example, TurboPlayer is able to control the PFL switch in the mixer console and vice versa. The PFL switch will be toggled before and after prelistening an element when prelistening is activated on the TurboPlayer side.

When prelisten is activated, the prepared element on the corresponding channel is prelistened. If no element is prepared on the channel the <next> element in the corresponding rundown is prelistened. If the element is already being prelistened on a different channel TurboPlayer looks for the next internal element in the rundown which is not being prelistened yet. Which element is being prelistened can also be defined more exactly when prelisten is activated via the macro command “TP_Prelisten2”. See the events chapter for more details on that.

An interesting detail is the question: what happens if the user opens the fader of a channel while prelisten/PFL is active on this channel ? As the desired answer differs, TurboPlayer supports a global mode called “FaderStopsPFL”. It can either be true or false and like all other modes it can be changed during runtime. If the mode is true TurboPlayer will stop the prelistening playout, will deactivate the PFL switch and will start the correct element (which might differ from the one which was prelistened). This is the standard behaviour. If the mode is false TurboPlayer will ignore any fader moving. This behaviour is useful if you want to set the correct fader level for the upcoming on-air start during prelisten. Be aware that this mode has only an influence on fader moves. Direct start commands (e.g. from a macro) work in both mode settings.

PFL bus number

In the picture above we show only the easiest case: the mixer console has only a single PFL bus. All PFL switches route the audio signals to this PFL bus and only a single speaker set (stereo or surround) can be served. Some mixer consoles have multiple PFL busses, which means the audio signal can be routed to different busses (to one at a time). This allows serving multiple speaker-sets, one with each PFL bus. This is useful if you have multiple GUIs and each GUI needs to have a dedicated prelisten. In the registry you can specify the PFL bus number with the parameter “PFLNumber” in TurboPlayer\Line-X\Channel-X\.

By the way: this parameter also activates the PFL switching of TurboPlayer. If the parameter is missing, TurboPlayer will not do any PFL switching. For external lines a PFL number cannot be defined so far, because these lines do not have a channel sub-parameter. In order to be able to route them to a PFL bus nevertheless, the default PFL bus for external lines is the PFL bus which is defined in the parameters of the first dedicated prelisten channel of the GUI which requests the PFL action.

GUI prelisten

In the picture above you can see an independent audio routing on the right-hand side. This represents GUI prelisten. The GUI can use either EasyPlayer or the CrossfadeMixer for prelistening. Both use MultiRec.ocx as audio engine internally. Typically MultiRec is configured to use a dedicated audio device, possibly on an extra audio card or perhaps on the mainboard. Sometimes MultiRec can use one of the devices MultiPlayer already uses (in the picture device 8) but this requires that the driver of this audio card allows to open an audio device multiple times. This is not supported by all audio cards.

What is special about GUI prelisten? You will see the big difference between GUI and engine prelisten if you take a look at a system with two GUIs: one of the GUIs is running at a remote / different computer. When the remote GUI starts the engine prelisten, the playout will occur on the engine computer (which is someplace else). When the remote GUI starts its own prelisten, the playout will occur on a sound device on the GUI computer. Sometimes it is sufficient to connect the output of the GUI sound device directly to a monitor speaker. But sometimes this output is mixed onto the same line as the engine prelisten for this GUI. Also note that each GUI has only a single device for its own prelisten which means it is not possible to use multiple EasyPlayers and/or the CrossfadeMixers at the same time.

PFL switching for GUI prelisten

It might be desirable to activate PFL for GUI prelistening. This is true if you have an audio routing like in the picture above: the GUI prelisten audio is mixed to the engine prelisten audio of device 8 and therefore the same PFL switching will be necessary. To activate PFL for GUI prelisten there is an extra value in the registry: the line parameter "PFLNumberForGUIPrelisten". It is used to assign a PFL bus number to a GUI, for example if GUI 3 should use PFL bus 1 you must enter "3:1". Multiple GUI-PFL assignments can be made by separating them by commas. The line you select for this parameter is not relevant for the TurboPlayer engine. But (depending on your mixer console and the used IO module) the line number might be sent to the mixer console together with the PFL switching command. In this case you must select the line accordingly.

Prelisten state for a pure GUI prelisten

(This is a hint for advanced users who know about macros already.)

Assume you have a setup with only GUI prelistening and with no MultiPlayer prelistening and you want to evaluate the prelisten state in an internal macro. As this normally is done with a call to either DataOfLine or DataOfChannel the problem arises that neither a channel nor a line is defined for prelistening and therefore you cannot query the state in a macro. The solution is to define a "dummy" line which is not being used physically. This line can get the PFLNumberForGUIPrelisten parameter. As PFL number you use e.g. 0. This number will never be used because without a real line no PFL switching is possible anyhow. But this trick allows to assign a line to the prelistening of a GUI and then you can query the prelisten state with "DataOfLine (x, PrelistenState)".

Multiple engine prelistening

It might be necessary to have multiple dedicated prelisten channels in the engine / MultiPlayer. This can be the case if you have multiple GUIs connected to one engine and each of them should be able to do its own prelistening via the engine. This is supported by defining multiple prelisten lines and channels in the registry. Each prelisten channel has the parameter “GUI” you can use to assign one (or multiple comma-separated) GUI to the channel. If no such assigment is done for a specific GUI, this GUI uses the first defined prelisten channel in the registry. If multiple GUIs are assigned to a single channel they are not able to perform an independent prelistening. If one GUI starts prelistening the other GUIs will show this state too and each GUI can control the playout.

A single GUI can also be assigned to several engine prelisten channels. The additional prelisten channels will only be usable via macro commands which allwo to specify the playout channel (-> TP_Prelisten2). Functions which have no information about the desired channel (e.g. pressing the middle mouse button) will use the first defined channel.

1.4.15 Transition lines

A transition on this note means the transition between two music elements. All additional elements being played at the end of the first music, in-between of the music or at the beginning of the second music are the transition elements.

Since version 5.2 TurboPlayer supports transition lines. A physical line can be declared as transition line by setting the parameter “UseForTransitions” of the line to 1/TRUE/Yes. Such a line should have multiple channels assigned. The number of channels must be sufficient to play all transition elements you want to play simulatenously.

During scheduling all transition elements must be marked with the flag:
“Time_StartOnTransitionChannel”. This is not intended to be done manually. OnAIR TrackMixer can handle this flag since version 2.0. OTM sets the flag automatically in many situations but it is also possible to set or reset the flag manually with a button for each element.

TurboPlayer will identify all elements belonging to one transition. An element in between of a music 1 and music 2 belongs to the transition music 1 -> music2. For drop-in/overlaid elements the rule is: if the overlaid element starts before the middle of the master element (the music) it belongs to the transition before the master element, otherwise to the transition after the master element. For the computation of the middle and the start points, the mark-based values are used. If this automatic assignment to a transition is not correct, it is also possible to set the flag “Time_StartOnTransitionChannel” to the value 2 or 3. 2 means: the element belongs to the transition before the master element, 3 means it belongs to the transition after the master element. These two special values can only be set manually so far, OTM does not support them.

The main idea behind transition elements is that they can all be played on a single line, as if they were compiled to a single element - like it was created by CrossfadeMixer. A transition line will stay open until the last element of the transition was played. In the players in the GUI it can be configured that instead of the information for individual elements the information for the whole transition can be visualized. A manual stop of a transition (e.g. by closing the fader or by executing a fadeout) will stop the whole transition. Elements which were not played so far will be skipped. Pausing a transition will pause all elements.

Nevertheless, this analogy is not perfect in all cases. The execution of the start according to the start attributes will still be based on the evaluation of individual elements. For example: if the user pauses a transition just before an overlaid elements should start but the master element of the overlaid element is not paused, the overlaid element will start, and the pause state of the transition ends automatically.

1.4.16 Group handling

In BroadcastSystem groups do exist. Groups can contain sub-elements but no sub-groups. When TurboPlayer was designed, one of the goals was to separate the real-time critical routines from all other routines. This was achieved by creating the module called "RundownKernel". This module handles all time critical tasks. To keep it as simple as possible, the kernel handles a linear list of the preloaded elements. All other elements and the show or group structure is mostly unknown to the kernel. This design decision results in some limitation of how groups can be handled by TurboPlayer.

The main limitation is that groups can not "run" like elements. The next pointer will never be positionend on a group and a group cannot be started. As it is sometimes desirable to have a running group, such a feature has been implemented – but it is a part of the TreeManager and the GUI. A group is started implicitly by starting one of its child elements. There is no other way to start a group, which makes it a simple rule. In contrast, stopping a group can be achieved by two ways and it depends on a group property called "FixedDuration". Normally, a group is implicitly stopped, if there are no more playable elements in it. That means all elements in the group must be sent, skipped or not be playable (placeholder, no start attribute, …) A running group can also be stopped by using the macro function "TP_StopGroup". If it is called, the eldest running group is stopped. To fulfill the rule just mentioned, the function stops all playing elements within the group and skips all other playable elements.

If the group property "FixedDuration" is set, the behaviour is different. Such a group continues to run, even if all elements in it have been handled. In addition to the rule above, such a group is stopped when the first element outside of the group is being started. There is also a logic which stops the group if its stop time has been elapsed and the show in which the group is located is no more the current show. Besides, the macro "TP_StopGroup" directly stops a fixed-duration group.

In addition, there is also the concept of the "active" (or last-running) group. The active group is the last started running group. This group remains active, until an element outside of the group is being started (in contrast to the running group, this does not depend on the "FixedDuration" flag). There might be no active group at all: if the last started element is not within a group.

These concepts define the behaviour of the following features:

- Group counter: the GUI provides windows which display group times. Such a window always displays information about the eldest running group.
- Group events/controls: as a group is handled by TreeManager, event handling is limited. First, only at start and at stop of a group events will be executed. Second, the time precision of group events is not as high as for elements (which are handled by the kernel). For group events there is the question: who executes the events, if there are multiple TurboPlayers playing the rundown ? Each TurboPlayer will execute the start events when it starts an element within the group for the first time. This can be the first element on one TurboPlayer and the n-th element on the other TurboPlayer. Each TurboPlayer, which has executed the start events, will execute the stop events when the group is stopped. That means a passive TurboPlayer, which does not start any elements, will not execute any group events.
- Insert position of spontaneous elements: If there is an active group and the next pointer is not on an element within the group, new spontaneous elements will be inserted at the end of the active group. That means, if all elements of a group have been played, but the first element outside of this group has not yet been started, new spontaneous elements will be appended to the group.

Two additional hints: (1) If an element with property "MuteStart" outside of a group is started, this does not end previous groups. (2) Elements which run less than the defined minimum runtime are "reset" and can be started again. This feature does not apply to groups. If a group has been started/stopped, it will remain so, even if the element which started/stopped the group is reset.

1.4.17 CartBeat, CartMotion

CartBeat

This is the marketing name for a feature of TurboPlayer: it can perform beat-exact starts. This feature requires information about the beats of an element. Within DigaSystem this information is generated during the production of elements and is stored within the metadata. MultiTrackEditor V5 can be used to generate or modify beat markers. In the database these beat markers are stored in the subclips table, in the BroadcastSystem they are stored in the field <BeatMarkers>. In both cases they are stored in a compressed format to avoid storing each marker individually. If beat markers have been set for an element the applications can show the following icon:

When an element with beat markers is running and the user wants to start a new element which has beat markers too, he can use a special command or mode to achieve a beat-exact transition. This is either the macro command “TP_StartBeatExact” or the general mode “BeatExact”. The command can be used for example with a special button, the general mode can be activated together with the “automatic” mode to achieve a beat-exact playout for every element for which it is possible.

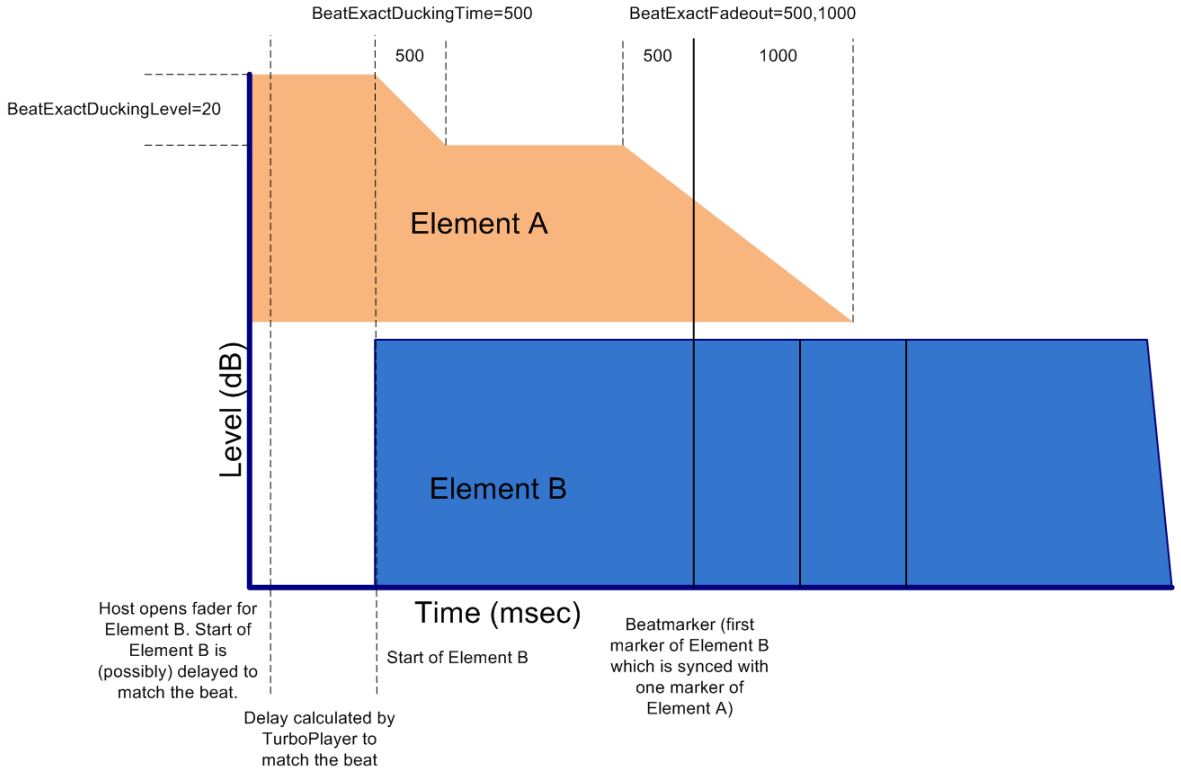
In both cases TurboPlayer will do a special form of start of the second element. When a start command is received or triggered internally TurboPlayer will first check if the transition is already beat-exact. This might be the case for preproduced rundowns if the producer has made beat-exact transitions with the crossfade mixer. Then TurboPlayer will immediately start the second element. If the transition is not yet beat-exact, TurboPlayer will delay the start. The delay time is being computed from the known beat marker positions. The second element need not have a beat marker at take start or mark-in, there might be some “forerun” till the first beat marker. TurboPlayer will compute the first possible start time for element 2 so that the first beat in element 2 overlaps exactly with a beat marker of element 1. There is a graphic below which illustrates the element handling.

Aside from the delayed start TurboPlayer will handle element 2 like always: it is started at mark-in and – if appropriate – a fade-in or fade curve is executed. Element 1 will be faded out and stopped quite fast after the overlapping beat-marker. The times can be adjusted with three parameters in the registry key “TurboPlayer\Stingers”: the value “BeatExactDuckingLevel” sets the value in dB element 1 is faded down at start of element 2. The value “BeatExactDuckingTime” sets the time for the ducking fade. The value “BeatExactFadeOut” sets the time of the fade-out of element 1.

The CartBeat feature is mainly intended for a usage with a manual start by the user. But it can also be used for a transition in which the second element has a sequenced start. This requires an activation with the parameter "TurboPlayer\Stinger\BeatExactSequenced". Of course, a sequenced transition can be prepared in CrossfadeMixer to be beat-exact. Typically, this requires loading every transition into CrossfadeMixer and to adjust the link values by moving the elements to appropriate positions. To avoid this manual scheduling, you can use the CartBeat feature for sequenced elements. Then it is sufficient to put the elements back-to-back. TurboPlayer will automatically move the second element forward or backward to achieve a beat-exact start. Of course, the original time calculation for the rundown changes because elements are started earlier or later than scheduled. Especially if you use a long forerun in the second element before the first beat marker is set, the difference time can be significant, because TurboPlayer must move the second element forward by this forerun time plus a small amount for the technical handling.

The achievable precision is the same as for every scheduled start of two elements. This value depends on the whole system. In a well-configured system TurboPlayer can achieve a start precision of approximately 1-2 ms.

The following graphic illustrates the element handling of TurboPlayer together with the possible parameters:



CartMotion

This is the marketing name for a feature of TurboPlayer / MultiPlayer: in surround audio files the sub-channels (tracks, streams, channels of a multi-channel audio) can be faded to individual levels. Typically, a special surround format is being used: "2.0 with Stereo". This is a multi-channel audio format with four (sub-)channels: front left, front right, back left, back right. These files can be produced with the MCA-file-import-dialog of the DatabaseManager 4, for example. For this audio format the applications can display the following icon:

With the macro command “TP_FadeSubChannels” it is possible to set the level of each of the sub-channels to an individual level. For example, you could set the front to -99 dB and the back to 0 dB. With the right IO module, it is also possible to use an external hardware controller to position the levels with the help of internal mixers (e.g. a Mixtreme mixer).

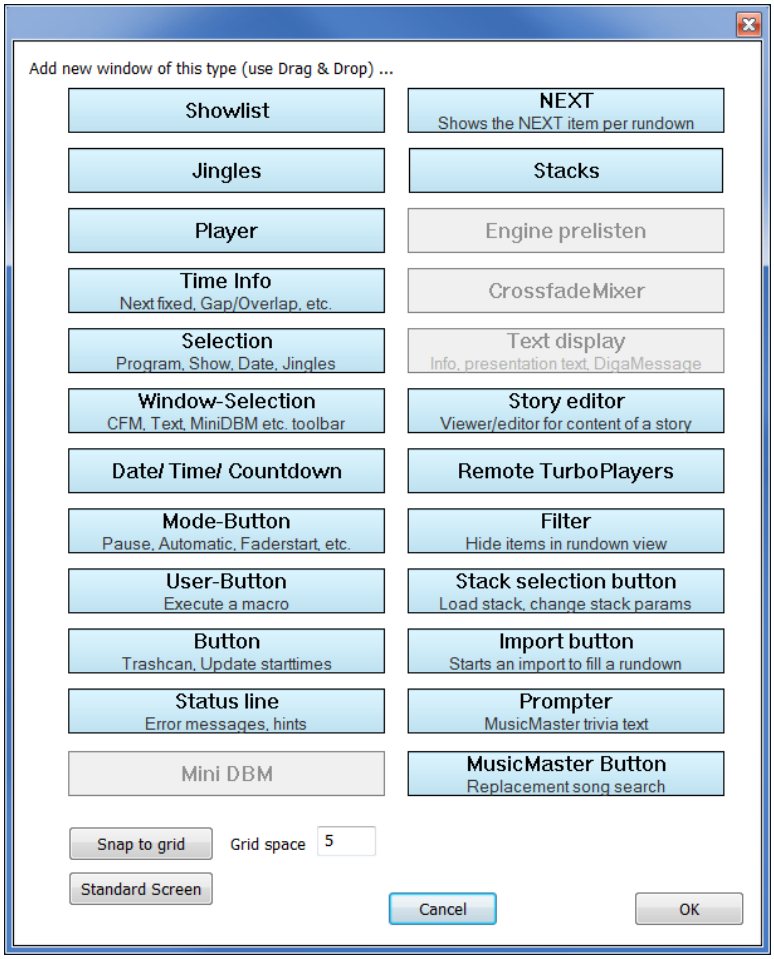
This feature can be used for a mood-shift during playout. Therefore, you produce the audio files in a way that the front and back sub-channels contain the same piece. The front could be the “full” instrumentation with all instruments playing loud and the back could be a damped version with a reduced number of instruments or the instruments playing softly. Then the presenter could start a music bed in the full version and while the bed plays, he could stagelessly switch to the soft version when he wants to say something and switch back to the full version afterwards. As the sub-channels are started sample-accurate the effect to the listener is that the mood of the element changes softly during playout. It is not noticeable that a crossfade between two variants of the same piece is being done. Of course, this feature requires that the timing of the individual sub-channels is sample-exact identical. This can only be achieved by producing the element, not by doing individual records.

2 GUI ADMINISTRATION

2.1 Standard Screen, Add Windows, Resizing windows

The GUI consists of different window types, which can be added to the GUI, and removed too.

System menu ->Add Windows…



Window Types:

- Showlist window;
- Jingles window;
- Player window;
- Timeinfo window (Next fixed starttime, Remaintime till next fixed start, Remaintime current playing group, Gap/Overlap);
- Selection window: (program, date, show, jingle selection);
- Window-Selection window (Activation buttons for: Prelisten, CrossfadeMixer, Text, Infotext, DigaMessage);
- Mode-Button (Activation, Faderstart, Pause, PlayFadings, Ghost, SelectSources, ButtonStart, OnAir mode, BCS connection, Filler, Ignore Markout);
- User-Button (linked with an event in TurboPlayer\EventsOut\);
- Button (Trashcan, Update starttimes);
- Statusline (engine and BCS connection state and error messages or hints);
- MiniDBM: a database view to search within specified databases and tables; can only be added once (greyed if already in the view);

- Stacks: the rundown can be a list (like the showlist) of any track of the main list (track 0) which follows the main track (showlist) or any node can be loaded into it (Jingles node, preproduction node, etc.)
- EnginePrelisten: the prelisten player window of the internal audio engine (MultiPlayer); can only be added once;
- CrossfadeMixer: the crossfade mixer embedded control; can only be added once; also used for OnAIR TrackMixer;
- Text: the presentation and info text of the showlist cursor element or a DigaMessage window; can only be added once per type: presentation text / infotext / DigaMessage;
- Story: this window can host a DigaStory.ocx which can display the text and metadata information for elements in a story in a combined view;
- Next: window that shows the next of the specified rundown (see settings of this window);
- Remote TurboPlayers: This window shows the connection states of remote TurboPlayers (communicating via TCP/IP via the BCS);
- Filter: filters operates on the rundowns Show and Stacks (definable) and hides specific items according to the filter settings;
- StackButton: Three modes exists (Track, Free, Switch parameter set);
- ImportButton: starts an import of data via import module into a rundown;
- Prompter: Shows the MusicMaster trivia notes;
- MusicMaster-Button: create on the fly rules/criteria for the search for replacement songs;

To add a new window of a certain type simply select the window type to add and do the D&D to the GUI. Afterwards the new window can be moved and/or resized.

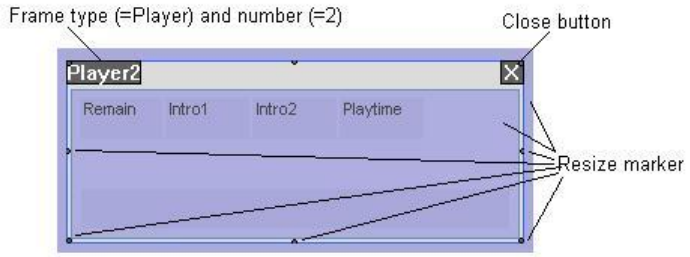
Grid spacing: grid for moving or resizing frames;

Snap to grid: all sub windows edges are aligned on a virtual grid using the adjusted grid spacing

StandardScreen: The default screen arrangement;

System menu -> Resize/remove windows - manually…

The GUI switches in the resize mode. Each window gets a title bar – with the frame type and the frame number and a close button – and a frame with markers to resize.

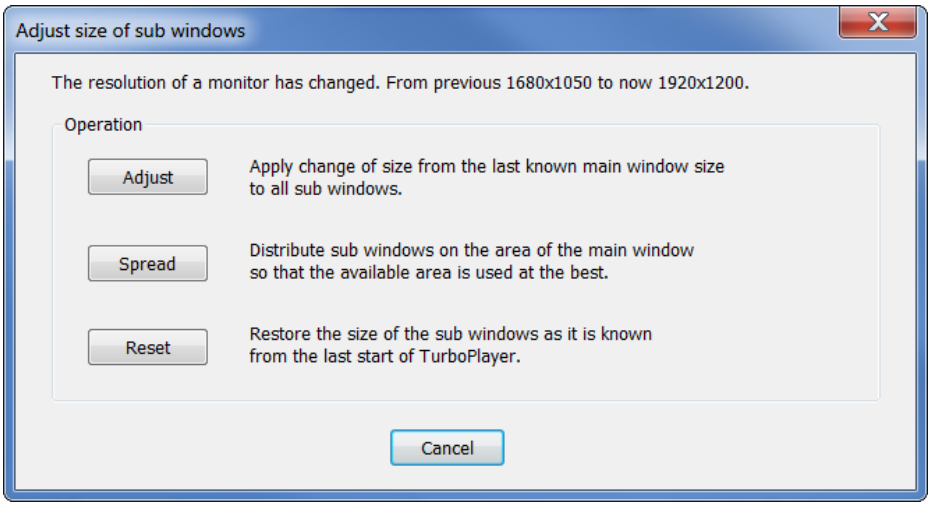


Each window (frame) can be removed by clicking the close button; it can be moved by mouse when moving the mouse inside the frame and keep the left mouse button down; it can be resized by dragging the edge lines or corner of the frame.
(Hint: To move the frame move the mouse over the title bar inside the frame).
Escape from Resize/remove frames… mode by pressing ESC or click on title bar of the application.

System menu -> Resize windows - automatically…

By default, TurboPlayerGUI does not change the position of the sub windows placed on the main window if the main window is resized. Either there will be empty space, or some sub window will be cut off or they will be completely outside of the visible area.

After resizing the main window, you can adapt to the new situation by resizing the sub windows either manually (see previous command) or you can use the function for an automatic resizing. If you select the command, this dialog appears:



The dialog allows starting three types of operations:

- **Adjust:** This operation stretches the rectangle of each sub window by a certain factor. This factor is given by the fracture of the current main window size and the last saved window size. Such a save occurs whenever TurboPlayerGUI is terminated or after one of the automatic resize operations was applied. This operation works best if all windows have been correctly positioned but then the monitor resolution changes.
- **Spread:** This operation computes a bounding rectangle around all sub windows and then performs a linear transformation for this rectangle and all contained sub windows so that the bounding rectangle fills the whole area of the main window. In order to avoid that the subwindows touch the border of the main window, a small gap on each side is inserted.
- **Reset:** This operation restores the position of all sub windows, as they are known from the last start of TurboPlayerGUI. This is independent from the current main window size. This operation is useful if previous operations did not work correctly and did destroy the layout. Pay attention: if you created new sub windows after the last start, these windows do not have a position known from startup. Therefore, these windows are kept unchanged.

A general hint: the automatic resize operations need to round the coordinates of the sub windows to full pixels. Therefore, these operations are not fully reversible, there might be rounding errors. This is especially the case if you apply the operation if the main window is very small.

Perhaps you noticed the “resolution has changed” line at the top of the dialog. This hint appears only if the dialog was automatically displayed because of a resolution change of one of the monitors of the system. Whether this dialog appears can be configured in the general settings on the tab “Miscellaneous”.

2.2 Main Settings

Main menu ->Settings

Settings | Communication:

For a description see chapter Parameters -> \TurboPlayer\Communication.

Settings | BCServer

Specify the BCServer to connect to; the user and password for the connection; the initial program, which will be selected at start up; timeouts and log file paths.
For a description, see chapter Parameters -> \TurboPlayer\

Settings | Keystrokes

At first start up default keys are assigned to the most important commands (e.g. Start, Stop).
To change a key assignment click on the command's line and enter a new key – can be a combination of Shift, Ctrl and Alt; it distinguishes between left and right ctrl keys.
To clear a key assignment push the clear button.
To change a command's parameter click on the item's column Param1, Param2, Additional Pars, Gui or Hook to change. Note: Not all parameters can be edited and not all commands need a gui number specified; it depends on the command. See chapter Command Functions in this manual.
You can build a new command by clicking the button "Create a new command". Choose the command and its parameters, which will be offered/updated in the lists to the right. Push the add button and assign a keystroke.

Normally key shortcuts work only, if an application has the focus. If you want some key shortcuts to work independent of the current focus (system-wide shortcuts) TurboPlayer offers two different implementations: a low-level Windows hook or Windows HotKeys. A hook worked well till Windows 7. Since this OS version Windows removes hooks sometimes, therefore a hook is no longer reliable and HotKeys should be preferred. However, HotKeys have the disatvantage (or simply: the different behaviour) that they cannot distinguish between left and right modifier keys (e.g. left and right shift key). If you want to use system-wide shortcuts, select the desired type in the corresponding combo box and set a check in the "Hook" column for each affected shortcut. Pay attention to make only very special key combinations work system-wide (typically only those of an additional special keyboard) in order to avoid that other programs become unusable because they do no longer receive required key messages.

All \TurboPlayer\EventsOut\ (defined in the registry) are listed at the end of the shortcuts list. Keystrokes can be assigned to them too. See also chapter Parameters -> \TurboPlayer\EventsOut\...

Prelisten keys refer to
OnAIR TrackMixer / CrossfadeMixer: The general prelisten commands usually start the prelisten on the TurboPlayer's internal audio engine (MultiPlayer). Checking this option the general prelisten shortcuts refer to the OnAIR TrackMixer / CrossfadeMixer (OTM/CFM, playout via local audio board).
EasyPlayer: Checking this option will cause the general prelisten shortcuts to refer to the EasyPlayer (playout via local audio board).

Settigns | IO-Modules

Add/ remove/ configure IO-modules; e. g. simulated IO, GPIO, etc.

Settings | Dialogs & Menus

All settings can be password protected; If protected the system menu items 'AddWindows…' and 'Resize/remove frames…" are disabled too. Settings are always enabled if no password is assigned. To change the password you must have the right permission, which is checked before. If a password is assigned you can now disable the settings by push the button or if currently disabled enable the settings by push the button and enter the right password.
Automatically disable dialogs after <n> min: check this to automatically disable settings (if tempoarely enabled by typing in the password) and activate password protection after the time elapses.

Close application without request: User won't be requested if application should really be closed;

Only Alt + F4 closes application: Application cannot be closed by mouse click on the close button; only shortcut Alt + F4 closes;

No maximize: application cannot be maximized; no maximize system menu item;

No minimize: application cannot be minimized; no minimize system menu item;

Disable all context menus:

All context menus are disabled; also those in the showlist to change certain element properties (like Sendstate, StartMode, StartType, Stinger, Class, Source, MixerSource, EventOut, etc).

Or select from the list of context menus which context menus should be disabled.

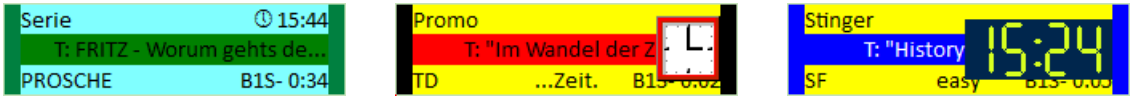
Language: Chose the language resource.

Settings | Miscellaneous

This dialog contains some miscellaneous settings, about the size dialog when screen resolution has changed, and about the display and insert/delete operations in a rundown list. It is also possible to activate the option "Disable rearrangement of rundown if filters are applied" in case is not desired to allow changes like insert, delete, copy or move in the rundown list when at least one filter is applied at the moment.

In addition, you can select details of the "clock" feature for CartElements. Such a clock is displayed on top of a cart for some time after this cart was played for the last time. The clock signals the user that the cart was recently played and should not be played again.

The appearance of the clock can be configured. You can select the type of the clock (Header text, analog, digital):



And you can set how long a clock is displayed and whether the last start time of a cart should be displayed or a countdown till the next possible use time.

Important hint: The storage of the play information is handled by the engine, you need to activate it generally in registry value "TurboPlayer\CartPlayInfo". The storage can either be in memory or you can specify a file for storing the play info. With a file it is possible to share the information among multiple TurboPlayer engines.

A played element is taken into account if it has either the "Class"="Cart" or the field "Music_Category" is being set. Elements are seen as identical if they have the same "DBRef" or (if empty) the same "Music_MusicID". Elements played on a different service are ignored. An element is treated as played if playout has exceeded the minimum runtime of TurboPlayer.

You might additionally want to activate or deactivate the usage of waveform display for items. If it is deactivated only the progress bar will be displayed during playback in the player windows.

Finally, due to the handling of overlapped windows on some custom controls or even windows that totally cover these controls, there was a handling in Drag & Drop operations, which was correct but incompatible with the usage of remote tools like Zoom or Microsoft Teams when the user was sharing the monitor and controlling the application remotely. This handling can now be deactivated checking the property "Allow Drag 'n' Drop for overlapped windows". This will allow a normal behavior in such remote control scenarios, but will cause the handling of overlapped/covering windows for Drag & Drop to be disabled, which might cause some discomfort or unintended behavior when dragging elements to control with other windows over them. So, check this property consciously and only when really needed (using Zoom, MS Teams, etc), else leave it unchecked.

One option is "Send computed times of all time info windows to the engine" (new in version 6.0). This is needed if you have WebTurboPlayer, which should display time info, too. E.g. the "remain time till the next fixpoint" or the "sum of durations". So far, the engine is not able to compute this time info, it can only relay the information form one of the regular desktop GUIs to the WebTurboPlayer GUIs.

On this subdialog you will also find a setting to disable the saving of changed parameters to the registry. (This setting is only available since TurboPlayer v6.) You can choose to save all changes, to never save anything or to save only till the next restart, which means: after a restart saving will be deactivated. This setting can be useful if you maintain your TurboPlayer settings globally and you want to prevent that TurboPlayer writes anything to the local registry.

Settings | Prelisten

Here you can select the local audio board used for prelistening by EasyPlayer and OnAIR TrackMixer / CrossfadeMixer (OTM/CFM). Chose whether to adjust the output volume by DigaSystem registry (Volume section) or set the output volume in dB explicitly.

Select the skin for the EasyPlayer.

Set the prelisten type (prelisten by internal engine – the MultiPlayer, by EasyPlayer, by OTM/CFM) for all lists to be the same.

Set the logfile section string for EasyPlayer and OTM/CFM.

If preslistening with EasyPlayer, you still have the option to Stop at MarkOut, that means prelisten will play till the mark out and stop. Per default this option is unchecked, which means prelisten will start at mark in, but play till the end of the audio.

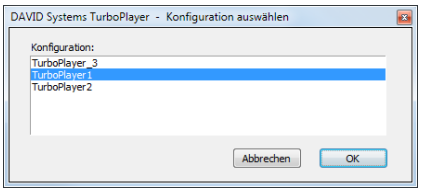
Settings | MusicMaster (MM)

The MusicMaster integration executes a search for replacement songs (triggered by cklick on the MM icon the rundown lists) and shows trivia notes in the subwindow of type ‘Prompter’.

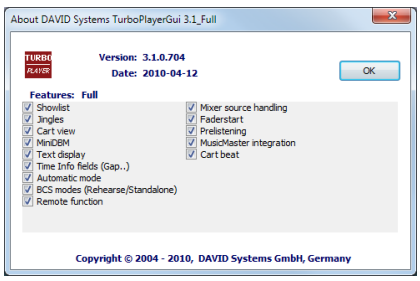
If activated define the MM database and which database is the corresponding one in the DigaSystem (this table must contain the elements of the MM database), define which database field (per default the music id) identifies the same element in both tables (MM and DigaSystem).

The ‘Perfect only’ flag applies a stricter rule in finding replacement songs. You can limit the number of replacement songs too (see settings).

TurboPlayer supports **multiple configurations in DS registry** now. These are top-level keys with a full set of settings which can be chosen during startup. See TurboPlayer-TechManual chapter "concepts" and the parameter description for "TurboPlayer\Configurations" for more information.



TurboPlayer is available now in 4 different "flavors" which differ in their feature set and in their price. See the chapter "TurboPlayer flavors" in the TurboPlayer-TechManual for more information.
If a feature is not licensed either the window type won’t be shown / cannot be added onto screen or the action won’t be executed; see About to see disabled feature range;



Settings | Video Preview

This subdialog contains some settings about previewing videos and images. You can select which type of previewer is being used (Multiplayer or MCLPlayerControl for videos, MCLPlayerControl or CxImageList.dll for images) and whether the settings for video preview should be applied to all windows. MCLPlayerControl can also be "outsourced" into StandaloneCFM.exe.

2.3 Window Types

2.3.1 Player

Shows the playing items which can be of any rundown list type. The playing channel the item plays on have to match with the channel of the player.

Settings

Player menu -> Settings

Settings can be copied and pasted to other player windows (menu -> copy settings / paste settings).

Settings | General

ChannelAssignment: Assign channel(s) to the player. You can assign more than one channel to the player. The latest started item on a player channel will be shown on the player; if it stops the former playing item (if exists and is still playing) will be shown.
For a detailed description of the channel numbers see chapter "Line and channel types".
Show intro in progress bar: shows the intro1 and intro2 remaining time in the progress bar. When intro is elapsed the progress bar switches to remaintime.
Progress bar/Waveform flashes: If checked the progress bar (or waveform) flashes last <n> seconds in the specified color.
Countdown is show for the last: Here you can configure the last seconds of a played element for which a big countdown field can be displayed.
Remain times have negative sign: Normally the remain time is displayed as a positive value if there is still time remaining till the end fo the element. If checked the sign is reversed.
Show waveform instead of progress bar: If checked instead of displaying the progress bar, the waveform that corresponds to the playing item will be shown (see left screenshot below). This option can also be toggled directly in the interface by clicking over the progress bar or the waveform holding the Ctrl key, i.e. Ctrl+LeftClick.
Show symmetric 1 channel waveform: refers to the way channels will be displayed in the waveform. You can show a single channel (levels) or a symmetric waveform (stereo channels).

Reference point during playout:

Remaintime refers to link out: shows first the remaintime till the linkout point, afterwards the remaintime till markout.

Progress bar refers to link out: progress bar shows first remain time till the linkout point, afterwards the remaintime till markout.

If element is in a group:

Here you can configure whether some of the fields display the times of the group instead of the individual element if an element is being played which is a member of a group. Be aware that the each of the settings is not applied if both fields exist in a player – first for the element and second the same content but for the group.

If element is in a transition:
Here you can configure whether some of the fields display the times of the transition instead of the individual element if an element is being played which is a member of a transition. Be aware that the each of the settings is not applied if both fields exist in a player – first for the element and second the same content but for the transition.

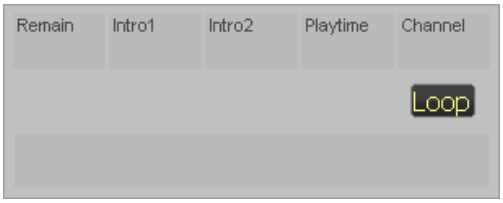
Behaviour during prelisten:

Playtime and remain time refer to: mark in / mark out: Check to display the playtime and remain time as mark-based values (like it is done during playout). Otherwise the times are based on file start / end.

Show intro and outro times: Check to display the intro/outro fields even when prelistening.

Settings | View

The position in the player to show the counters for the remain time, the intro1 and intro2/outro times, the playtime and the title and the progress bar (or waveform) can be adjusted here. To change click on the item to edit. In place edit of the caption and the horizontal alignment and the fieldname (= xml tagname, in case of additional added field).
Set position by mouse click in the player map.
For choosing which items should be displayed over the waveform do it by selecting them in the "Waveform items" group.
You can view a *Play/Stop button* (depends on the channel's play state). Press shift for Fadeout instead of Stop or for Stop instead of Play (e.g. the item is prepared and ready for play you can unload the player by pressing shift and click the Stop button (button title changes from Play to Stop if shift is pressed)).
You can view a *Loop button* which toggles the loop mode of the player (the player must have only one channel assigned, because otherwise it is unknown which channel is meant (the loop button is inactive in this case)).



Drag & Drop from any rundown (channel of the source rundown need not to be assigned for that player) onto a player; a copy of the element is inserted into the target rundown; target rundown is the rundown its channel is defined for that target player.

Drag & Drop from DigAIRange or Turbo trashcan onto a player is now possible; ditto: a copy of the element is inserted into the target rundown its channel is assigned for that player.

Settings | Color&Font

Select the field from the box and adjust colors and fonts. For adjusting waveform item colors, click the button just below the sample drawing to select the item.

2.3.2 Date / Daytime / Countdown

Shows the current data or the current daytime.

Settings

menu ->Settings

Adjust colors, fonts and the format for the date and daytime.
Countdown: Counts till a daytime fixpoint. Set a caption, specify fix daytimes in the format "hhmm" or specify minutes "mm" in an hour and the valid hours "hh" of the day.

2.3.3 Selection windows

Select the program, the date, the show or the jingle group.

Settings
menu ->Settings
Adjust colors, fonts and caption.
Settings can be copied and pasted to other selection windows.

Menu
Adjust the selection type (program, date, show, jingle group).

After the program or date was changed the new program or new date is highlighted to indicate the program or date does not suit to the current selected show. The highlighted state ends if a show of the new program or new date will be selected (and therefore loaded) or the program or date fall back to its former values after a period of time, to indicate the current program and (start-)date of the show.
Hint: Selecting a new show while any playings is not possible.

2.3.4 ModeButton

Modes are:
- Activation (three modes = LiveAssist, Passive, Automatic)
- Play fadings
- Pause
- Faderstart
- Ghost
- Select sources
- Buttonstart
- OnAir
- BCS connection (three modes = normal, Rehearse, Stand alone)
- Filler
- Ignore Markout
- Beat exact
- Sync-Start
- Free showlist composition
- Auto-load jingles
- Adjust loudness
- Extended show scanning
- Assign overlaid
- Embedded start

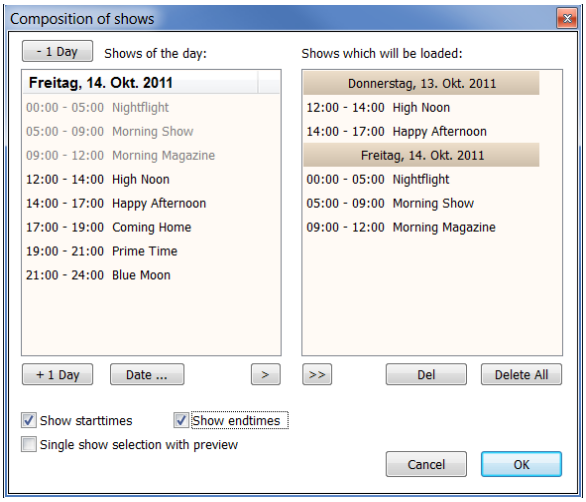
All mode states are on / off except Activation and BCS connection:

Activation:
When changing activation mode to LiveAssist and more than one TurboPlayer (TP) operates on the same program the new mode is first pending (highlighted state). The enquiring TP waits for the answer to its request. If the request was refused by another TP the old mode is shown; if it is not the new mode is accepted. At the same time the enquired TP shows a dialog with the request (“Another TP wants to become active”). If the user does not react on the request it is accepted after a timeout.

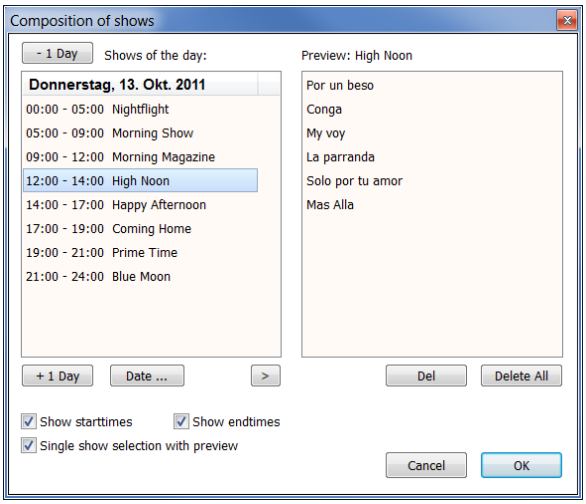
BCS connection:
Rehearse: No connection to the BCS; after reconnect takeover data from the BCS into the TurboPlayer;
Stand alone: no connection to the BCS; after reconnect to the BCS takeover data from TurboPlayer into the BCS system;

Free showlist mode:
If this mode is activated and a show will be selected -> instead of the box with the list of all shows of the day the **show composition dialog** appears.

Create a composition of shows of one day or of different dates.
Chosen shows can also be removed from composition by 'Del' key.



If you select "Single show selection with preview" the appearance changes. Only a single show can be selected on the left side and on the right side a preview (the title of the top-level elements and groups) of the selected show is displayed.



Settings

menu ->Settings

Adjust colors, fonts and captions for the active and inactive button state (=on/off).
Special case: Activation and BCS connection state mode: There exist three states: Adjust colors, fonts and captions for each state.
Settings can be copied and pasted to other mode buttons (except activation and BCS connection state mode button).

2.3.5 TimeInfo

TimeInfo can be one of these types:

- Next fixed start item
- Remaintime till next fixed start item
- Remaintime current group
- Gap/Overlap
- Crossover counter Outro -> Intro
- Duration sum
- New elements
- Live recordings

Next fixed start: shows the fixed starttime of the next fixed start element (must be of sendstate placeholder/ planned/ cleared and of start type FixedEnd, FixedStart or Fix).
Also the showend can be a fixpoint (see settings | More >) or the showend is the only one fixpoint (fixed start items are ignored).

Remaintime till next fixed start: Counts till the next fixed start item / showend.

Remaintime current group: Group remaintime refers to the first group in the specified rundown which is playing or paused. If no playing or paused group exists then group remaintime refers to the group with the NEXT element in it.
Group remaintime will be calculated as difference between group's endtime and starttime of the first element in the group that is playing or the NEXT (if group is not playing).
The remaintime will be counted if the group is playing and not paused.

Gap/Overlap: Shows a gap or an overlap in the specified rundown list. It analyzes the active show of the rundown.
The gap (or overlap) will be calculated as gap_begin + gap_end:
gap_begin: is the gap BETWEEN _either_ the showbegin time (in case the show is not yet started and the current daytime has not yet reached the showbegin time) _or_ the current daytime (otherwise) AND the NEXT item (if it is in the activce show) or the first element that is of state planned / sending / placeholder in the active show.
gap_end: is the gap BETWEEN the last element that is planned / sending / placeholder before a fixed element or before the showend (if no fixed element exists) AND the fixed element (either the showend or the next fixed element if existing in that show).

The gap counts if no element in the show plays and the showbegin time is reached or the show was already started before the showbegin time.

Crossover counter Outro -> Intro: Shows a counter for a sequenced play till the intro of the 2nd element will be reached, counter starts at outro point of the 1st element.

Duration sum: Computes the sum of audio/text duration of elements. Different types and options can be configured in the setting dialog. In principle the window displays the information how much audio / text is available for playout in the current show.

Summation is started either with the first element of the current show or with the <next> element (in rundown mode sequenced) or with the cursor element if the cursor is in the current show (for all other rundown modes). Summation is stopped either at the end of the current show or at the next fixed element. For the type "remain" only elements in state planned and cleared are counted; playing elements are counted with the maximum remain time of all playing elements, paused elements are counted with the sum of the remain times. An option is available to sum up only the times between the link points (instead of between the mark points). The many possibilities with the ignore-mark-out-feature are not treated. An ignore-mark-out flag is ignored until an element is really played behind the mark-out.

New elements: This is not a "real" counter. This info type is used to signal new elements in a rundown list to the user – especially by flashing. This window works only if you have activated the special colorization of newly inserted elements for the corresponding rundown list. When no new elements are

detected, the window appears in inactive state. When new elements are detected, the window becomes active or – if desired – flashing. This stops and the colorization of the new elements in the list is removed when the user clicks into this window. In case you want to display the counter: it is the elapsed time since the insertion of the first new element.

Live recordings: This window displays the progress / level report of a live recording. Progress is displayed as the elapsed / recorded time, the level is displayed with an optional mini-level meter on the right side of the window. So far, progress / level information is only available from ROAD. For configuration you need to know the name of the recorder as it is configured in the settings of the IO module (TIOROAD).

Settings

menu ->Settings

Adjust colors, fonts and caption.
Settings can be copied and pasted to other timeinfo windows.

Computation by engine

Since version 6.3 of TurboPlayer it is possible to let the engine compute the times and the GUIs will only display the times computed by the engine. This was introduced for WEB-GUIs so that they do not need to compute the times themselves. In order to ensure to get the same times on regular GUIs and on WEB-GUIs it was necessary to bring the computation into a central place – the engine.

In order to use the computation by the engine, you first need to configure, which times should be computed. This is done in the registry key TurboPlayer\TimeInfo. Each subkey represents one computed time and it can have a handful of settings. Please have a look into the parameter documentation (e.g. DigParam.rtf) for more information!

After the desired times were configured (and the engine was restarted), the configuration dialogs for the time info windows in the TP-GUIs will show a dropdown box with the configured times. If you choose one of them, the computation by the GUI will be deactivated and the times from the engine will be displayed instead. Please have in mind that we have tried our best to achieve that the computation by the engine and the old computation by the GUI always yields the same time value. Nevertheless, this cannot be guaranteed and there might be subtle differences. And in the future the computation in the engine might get new features/settings, which are not existing in the GUI computation.

2.3.6 UserButton

Settings

menu ->Settings

Adjust colors, font, caption and button id.
Select an eventout from the list that will be executed when button is pushed.
Button properties can also be set by command TP_ChangeUserButtonProp. Therefore a button unique id (= ButtonID) is necessary. See also chapter Command Functions (-> Events, Shortcuts Controls and Macros -> Macro programming -> Command Functions).

2.3.7 Window-Selection (Prelisten, MiniDBM, CFM/OTM, InfoText, PresentationText, Story, DigaMessage)

This window comprises the icons for the prelisten via internal audio engine 'MultiPlayer', the CrossfadeMixer (CFM) or OnAIR TrackMixer (OTM), the text windows (presentation text, infotext and DigaMessage), DigaStory and the database view (miniDBM).
Click on the icon brings the corresponding window on top (these windows overlapps after fresh installation and mostly afterwards due to lackage of free screen work area if working with one monitor).

Undock / Dock:
Windows OTM/CFM, miniDBM, Text can be undocked and free (also outside the TurboPlayer work area) positioned and docked again. Undock by right mouse button menu or the icon in the right corner of these windows (beside the hide button).

Hide:
OTM/CFM and miniDBM can be hidden by click on the hide button in the upper right corner of the window.

Prelisten window:
Progress bar which shows the audio take and points for Markin, Markout. Change play position by mouse click within the bar, also Markin or Markout if in 'Change Markin/out' mode.

OTM / CFM:
Buttons (beginning from the left) for Save, Exit, previous crossfade or item, next crossfade or item, one track mode, two tracks mode, undock, hide.
Items can be dropped onto the OTM / CFM to be loaded, can also be dropped onto the OTM / CFM icon.
Fade curves feature. Enable it in DS registry under \BroadcastSystem\FadeCurves (String) = True/False, 0/1, Yes/No.

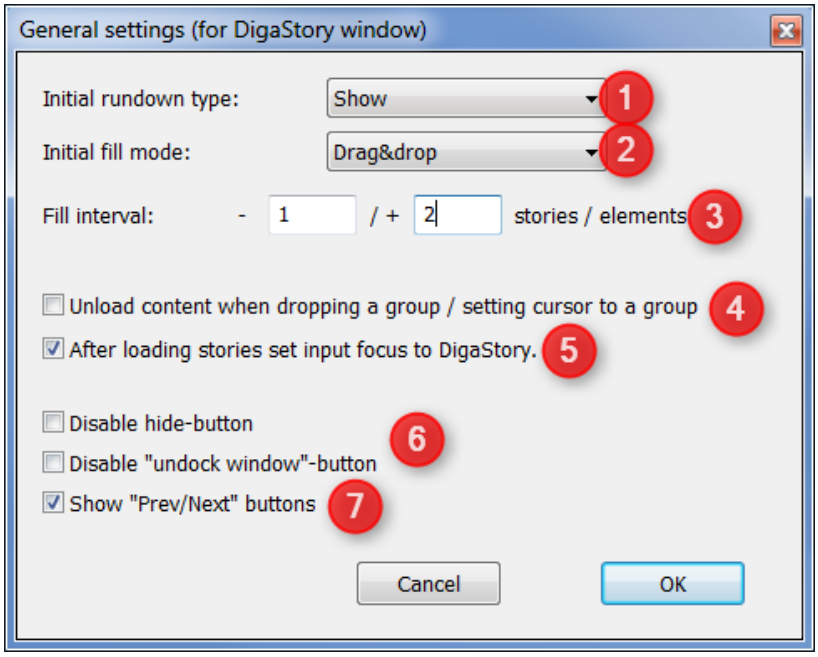
Text (presentation, infotext of the cursor selected showlist item and DigaMessage messages)
If you drag new windows of type "text" onto the surface you will notice that the type is fixed and cannot be changed: the first window will always be a presentation window, the second one a infotext window and the third one a DigaMessage window. More than 3 text windows are not possible at the moment. E.g. if you want to have only a DigaMessage window you must drag all 3 windows onto the surface and delete window 1 and 2 afterwards.
Buttons are available for:
Undock: (see above)
Edit: Text can be edited now; the cursor jumps into the text window. Attention: None hooked keys do not work till the cursor get off the text field (leave edit mode). The hold state becomes automatically active. Leave edit mode by ESC or menu or double click on title. Get into edit mode by menu or double click on title.
Hold: Activate the hold state keeps the item's text regardless whether the cursor moves
Empty: Clear the content of the window (DigaMessage only)

Settings: Specify the font used if creating a new presentation text or to show the infotext.

Text from the status line can be dropped onto the text window and error messages or hints can be read (or double clicl on status line message field).

Story window. This window type hosts the DigaStory.ocx and is only available since TurboPlayer version 4.6 and if you have registered the OCX. DigaStory displays a view of presentation text with "embedded" metadata lines and media elements. The configuration is separated into two parts / dialogs. The first one is shown by TurboPlayer and holds TurboPlayer-specific parameters, the second one is shown by DigaStory and holds DigaStory-specific parameters.

This is the dialog of TurboPlayer:



(1) Initial rundown type:

This is the initial rundown which is assigned to the window. During runtime the assigned rundown is changed by drag&drop of a story or an element from a different rundown into the window. Afterwards fill modes like "Follow cursor" or "Follow next element" are based on the new rundown.

(2) Initial fill mode:

The fill mode controls how the window can be filled. Drag&Drop of a story or an element is always possible. During runtime the fill mode can be changed with 3 buttons in the window header. Three fill modes are defined:

- Drag&drop: the window can only be filled by drag&drop.
- Follow cursor: the loaded nodes follow automatically the user cursor of the assigned rundown.
- Follow next element: the loaded nodes follow automatically the <next> pointer of the assigned rundown.

(3) Fill interval:

With the fill interval it is possible to load more than a single story or element. You can specify n previous and m next items to be loaded. An item in this sense is either a story (with all elements in it) or a single element outside of a story (top-level or in a group).

(4) Unload content…:

DigaStory does only display stories and elements but no groups. If the user tries to load a group (by dragging&dropping a group onto the window or by positioning the rundown cursor) normally nothing happens – the content remains unchanged. If you set this option, DigaStory will be emptied.

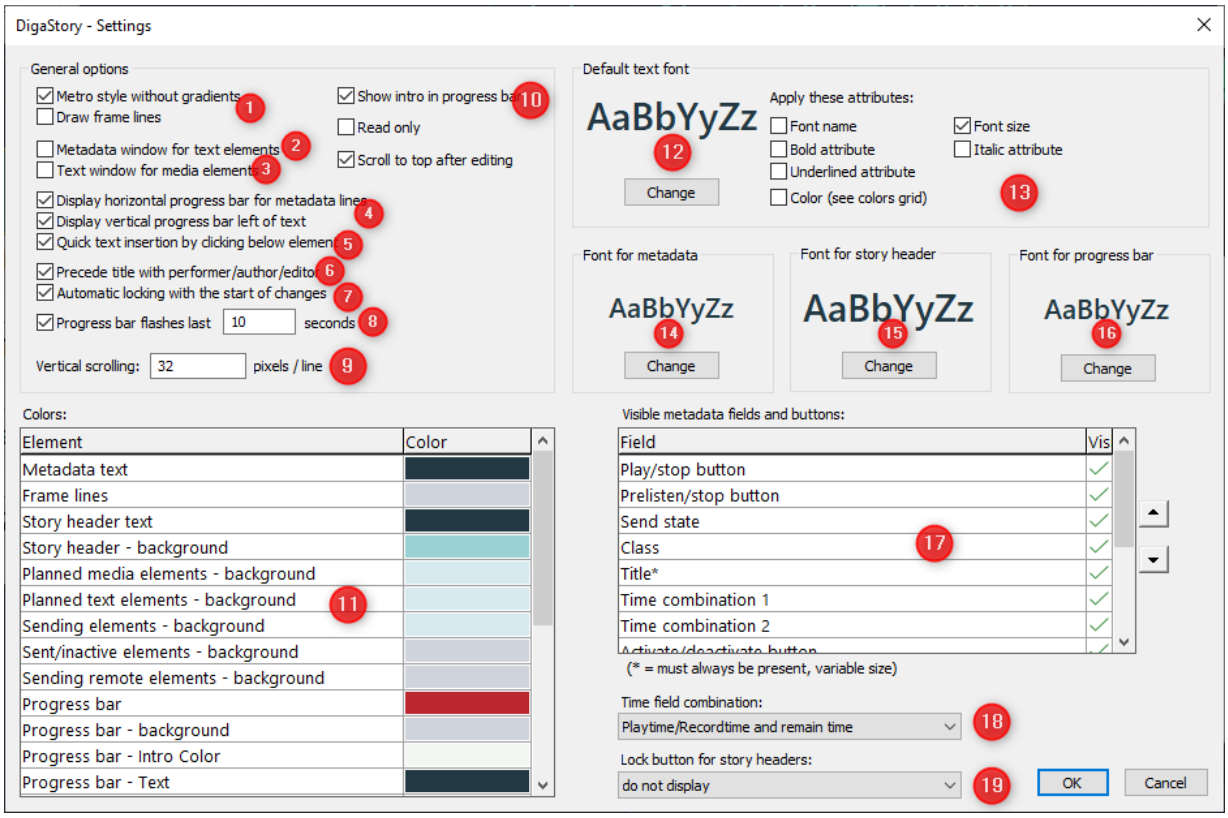
(5) Set input focus:

If you set this option the input focus of Windows will be set to DigaStory after a new content is loaded. This allows the user to immediately start writing. Otherwise he must click into the window of DigaStory first.

(6) Disable ... button: Here you can activate or deactivate the corresponding buttons in the window header.

(7) Show prev/next buttons: Additional previous/next buttons in the window header allow to change the loaded content without the need to drag&drop a new item into the window.

This is the settings dialog of DigaStory:



(1) Metro style / frame lines: Selects the general appearance with or without gradients and with or without frame lines around subwindows or –fields.

(2) Metadata window for text: If checked DigaStory displays a line with metadata fields for each text element.

(3) Text window for media: If checked DigaStory displays the presentation text of media elements too. In principle such a text appears like the text of a pure text element but be aware that at the moment TurboPlayer has no logic to "play" such a text. Only the media part of the element can be played.

(4) Progress bars: Here you can define which of the progress bars should be displayed.

(5) Quick text insertion: If checked it is possible to create a new text (element) at the end of a story by clicking into the empty area without the need to execute an extra create-text command.

(6) Precede title with P/A/E: Precedes the title within a metadata line with either the performer, the author or the editor (whichever field is found first).

(7) Automatic locking: Whenever the user starts to make changes, DigaStory will automatically set a BCS lock for the affected story. If the parameter “KeepTextFilesOpen” is set, the RTF files will be locked by holding them open, too.

(8) Progress bar flashes: Like in the player windows of TurboPlayer the progress bar can flash in the last seconds before a played element stops.

(9) Vertical scrolling: DigaStory has only a single vertical scrollbar for all text parts, metadata lines, gaps and headers. Therefore there is no real line height known to be scrolled when the user clicks the arrow up/down buttons. Here you can define a virtual line height in pixels.

(10) Show intro: If checked the progress bar first displays the progress til the first intro point in the color defined in the list below. After reaching this point the regular progress bar is shown.

(11) Colors: This is the list of all definable colors.

(12) Default text font: This font is the default font to be used to display the presentation text of elements.

(13) Apply these attributes: Here you can define which font attribute of the default font should be applied to the presentation text. You can select any of the attributes in any combination. Normally you would want to keep individual text formatting like bold/italic/underlined or the text color because it has a certain meaning and is deliberately set. But perhaps you want to make the font size identical and bigger to be easily readible. The default text color can be set in the colors list. In case the text files of an element cannot be loaded an error message is displayed instead of the text itself. Therefore the default text with all its attributes and the color "Error text" from the color settings is used. Be aware that changed text will be written back to the RTF file including the attribute changes during the load, meaning: the text is saved like the user of TurboPlayer sees it.

(14) Font for metadata: This font is used to display metadata lines.

(15) Font for story header: This font is used for headers which introduces a new story (in principle the story item itself).

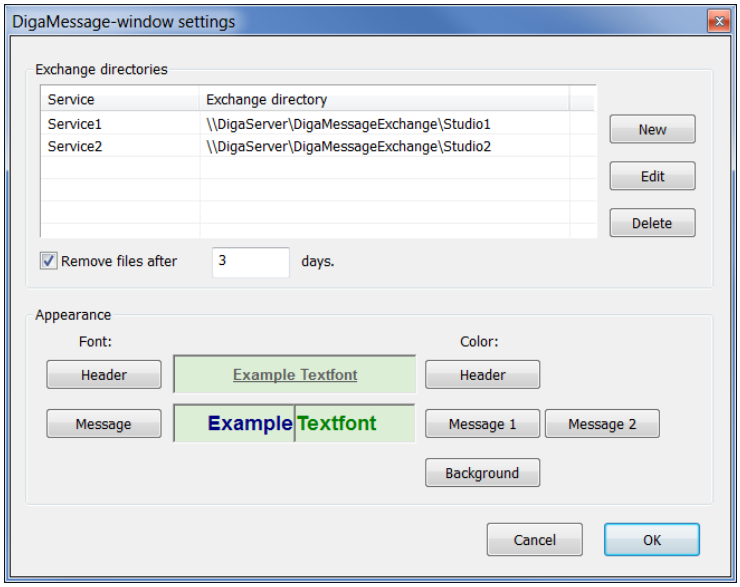
(16) Font for progress bar: This font is used for text output within the progress bar.

(17) Visible metadata fields: Check every metadata field you want to be displayed in a metadata line. The arrow up/down buttons on the right side allow to move the selected field up or down. The width of the fields cannot be adjusted and is automatically calculated from the configured font. The title field is the only one which has a variable width in order to span a big total width.

(18) Time field combination: In the metadata fields there are two time combination fields. This are not real metadata fields, but their content changes dependent on the current play state of the element. The reason for combining the fields is safing a little bit of space. Here you can configure which combination should be displayed.

(19) Lock button for story: A story header is not separated into columns like a metadata line. Therefore you cannot define a column for the lock button. Nevertheless you can enable lock buttons for story headers and make them appear either at the beginning of a line, at the end of a line or at the same position as for metadata lines.

DigaMessage window (available since version 4.2)
The window type "text" is also used to display messages received from DigaMessage. In this case the window is strictly read-only – messages can only be displayed but no messages can be sent back.
The settings dialog looks like this:



In the part "Exchange directories" you can specify for each BCS service you are going to use the corresponding message exchange directory – that is the place into which DigaMessage writes its xml files. If desired old message files can be deleted after n days. in the part "Appearance" you can configure the fonts and colors which are used to display messages.

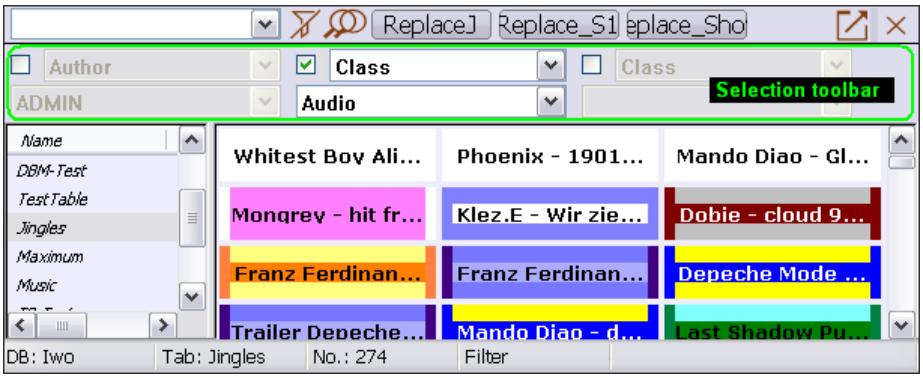
MiniDBM:
View comprises the tables and databases on the left and the content of the tables on the right hand side. In the upper left corner is the search text field: Enter words to search within the selected or specified tables. Right click in the search text box to adjust the database fields to search in.

Settings: adjust databases and their *tables* which will be shown in the left part of the view. Define tables which will be used in Multi table search mode. Adjust which database fields are to be shown in the content view (right part).
Adjust *columns* to view.
Definition toolbar: Chose which criterions can be chosen by the user, set a selection cannot be changed by user (deactivated), a selection default criteria at startup, etc.
Cart View: self explanatory

Buttons for
Select All: clears the search field and shows the whole table(s) content,
Multi table search: (indicates the tables selected for multi table search (see settings) with an loupe icon in the table view (left part),
Replace buttons: replaces the selected item in the miniDBM with the selected in the specified rundown list (right click on replace button for settings menu).
Undock button: window becomes a free popup and can be minimized on its titlebar;
Hide button: temporarely hide the window, press the corresponding icon in the window selection frame to show this sub window again;

Middle mouse button on item starts/stops prelisten (by EasyPlayer) the selected item.

If a table is defined to be shown in **cart view** (-> \Digas\Database\<Database>\<Table>\CartWallView = TRUE) the items of the table are shown as carts automatically if the table is selected. Use the CartColorManager to adjust colors, fonts and layout of the cart.



If a table is defined to have a **selection toolbar** (-> \Digas\Database\<Database>\<Table>\DefinitionToolbar = TRUE) this toolbar will automatically be shown if this table is selected. The toolbar can be switched on and off by context menu. The selection toolbar works as a filter. Each selection (filter) can be activated by setting the check box. In the upper selection box the criterion can be selected, the lower selection box shows which values are available for that table, e. g. you chose ‘Author’ as criterion, the lower box shows all authors existing for that table. The filters are AND-combined if different criteria (e. g. author = “Author1” and class = “Text”), in case of same criteria the values are OR-combined, e. g. author = “Author1” OR author = “Author2”.

Filter works only if the toolbar is visible resp. the selection is visible, in case the MiniDBM is resized and any selection cannot be displayed any more, this selection becomes inactive.

MusicMaster (MM) integration: A search for replacement songs can be executed for a rundown element by clicking the MM icon on the element. The search is performed in the MM database using the parameters adjusted in main settings | MusicMaster and the result is viewed in the MiniDBM in the corresponding DigaSystem table (mostly elements in both systems are identified by their song/music id). By using the replacement buttons the rundown selected item can be replaced by the selected item in the MiniDBM.

It is also possible to configure MiniDBM to perform the search on a predefined table. This can be configured in the Tables tab of the settings dialog from MiniDBM under the settings “Default database / table where the search happens”. It is also possible (through the option “Prefer table from selected element (from DBRef field)” to set whether the elements table present in the DBRef field will be preferred over the pre-defined table setting.

Updated to **new MusicMaster interface** (SR-18) for replacing elements and Unicode support; use DIMusicMaster.DLL >= 114.

The configuration file DIMusicMaster.xml moved to another folder (reason: restricted write permission under Vista/Win7) to %ALLUSERSPROFILE%\DigaSystem\DIMusicMaster\ (%ALLUSERSPROFILE% Win7: C:\ProgramData; WinXP: C:\Documents and Settings\All Users\Application Data\).

The xml file is automatically copied by the new DIMusicMaster.DLL.

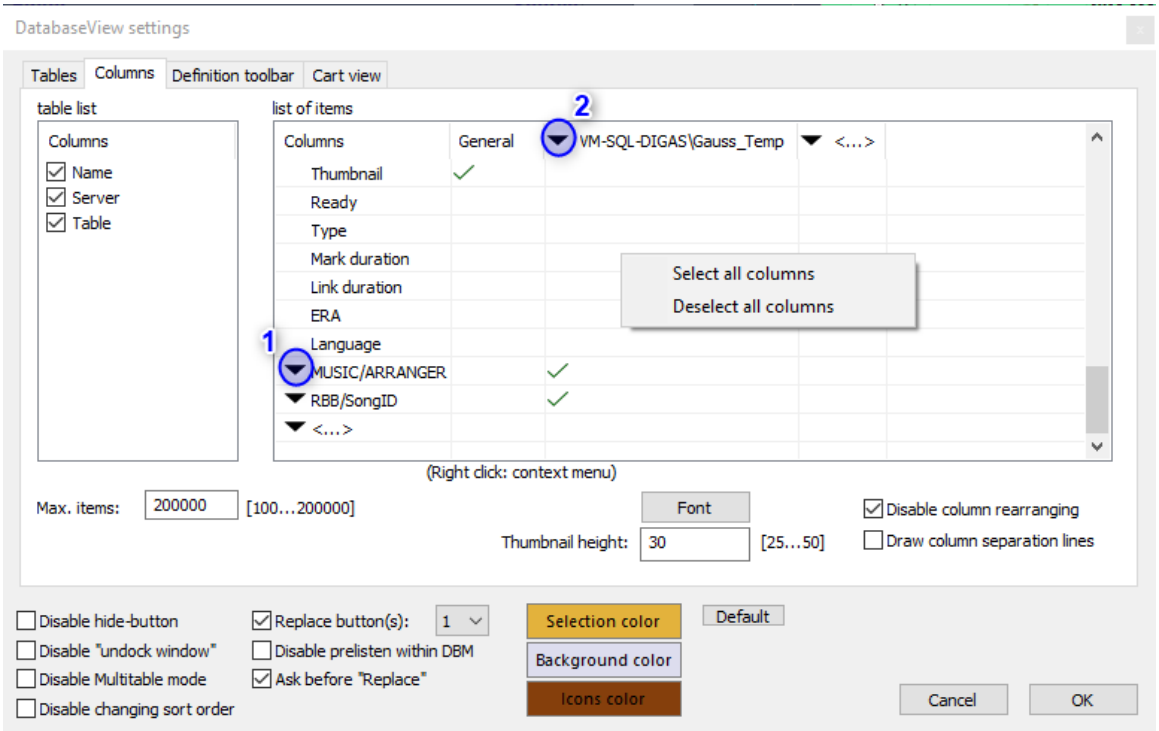
On server side use HTTPPostServer.exe (no more MMServer.exe); it supports UTF-8.

Shortcuts can be adjusted for (see main settings | shortcuts): Actualize (update content considering search field and selections), switch on/off/toggle selections (filters), copy the selected item into a rundown, show/hide selection toolbar, load into player (prepare), prelisten the selected item, change selection, scroll.

Considers the **table sort order**: defined in DS registry (Digas\Database\<database>\<table>\Order (Int.) = 0..n; if no order value is defined the table is added at the end; see settings option ‘Use ‘Order’ value for table sorting’; default sort order is alphabetical by table’s alias name (resp. table name);

Faster view of table entries in list view mode (not cart mode), noticeable if large number of table entries;

Custom fields and define **columns layout for each table**:



Below the predefined columns are the custom fields marked by an arrow which acts to (1) edit the custom field name or the custom field caption (if specified it is shown instead of the custom field name in MiniDBM) or to enter/delete a custom field. Mouse hover over the custom field name shows its caption name.
In the header you can specify the tables (2); General means the default table columns layout.

Icons for 'CartBeat' and 'CartMotion' feature

Maximum items

Turbo uses its own defined value of maximum entries to show for database queries; see settings | Columns > Max. items. Before the value \Digas\Database\<database>\<table>\DBMLines were used by default.

Results can be restricted considering the sort order, e.g. show the latest 100 results or first 50 in alphabetical order, etc.

2.3.8 Jingles

Jingletabs

The jingle window has tabbed jinglegroup access (tabs lie at the top edge of the window), jingletabs can be hidden (-> mouse menu).

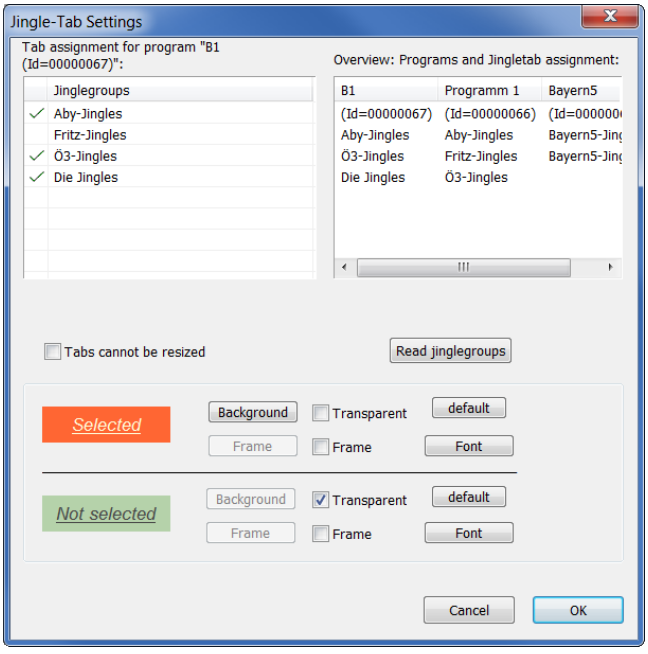
Maximum number of jingle tabs is 100.

A "Hold" button is available at the top-right corner which allows to prevent the automatic change of the loaded jingle group due to changes of the <next> pointer. (Normally the jingle group assigned to a show/group is loaded automatically.)

Menu -> Settings

For each program desired jinglegroups can be assigned to the tabs. The left list shows either the global or the service-specific jingle groups – which of them are shown must be configured in the service settings in DigAIRange.

Adjust colors and fonts for current selected tab and none selected tabs.



Jingles

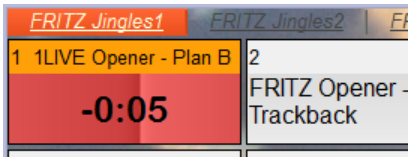
Jingles can be selected by mouseclick or key shortcuts (depends on the settings).

Middle mouse click on the jingle starts/stops prelisten.

A none playing jingle shows in the header: the jingle number, the fade option icon (if stinger or mute start), the prelisten icon (if prelistened) and the playout duration. In the body there is the title of the jingle.

A playing jingle shows in the header: the jingle number and the title of the jingle. In the body there is space for for the prelisten and fade option (stinger, mute start) icons and the remain time counts here.

Progress bar option for jingles; see settings, color of remain part adjustable.



Menu -> Settings

Call ->menu ->Settings

Adjust colors and fonts.
Set mouse click action, disable D&D, set vertical jingles order.

Menu

Hide jingle tabs;
Set prelisten type – internal engine (MultiPlayer), local audio board (CFM/OTM, EasyPlayer).

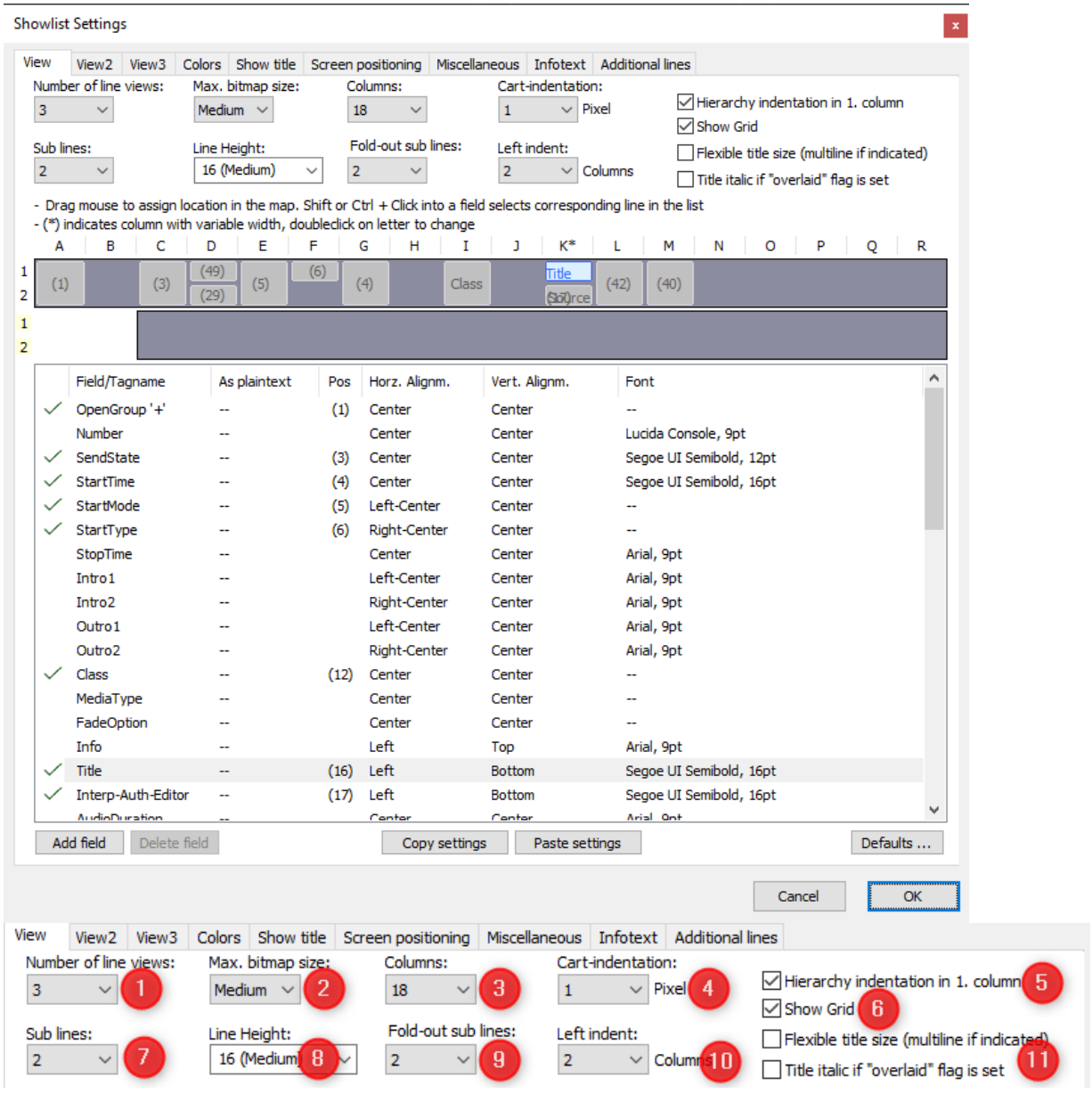
2.3.9 Showlist

Settings

menu ->Settings

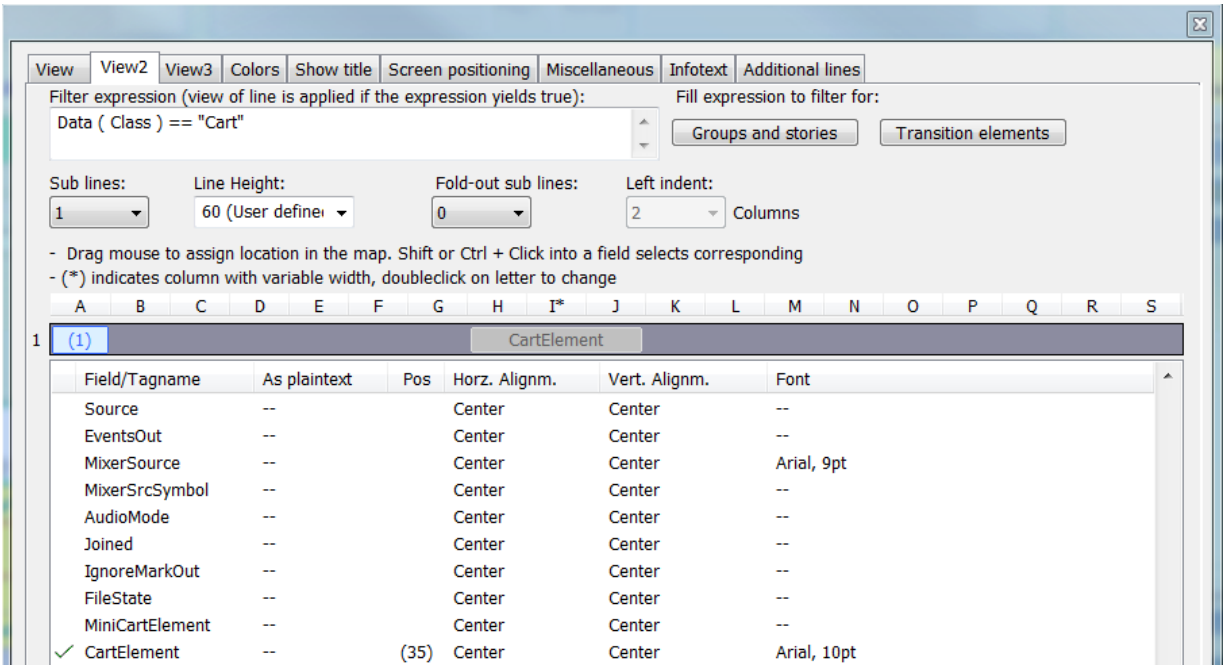
Settings can be copied and pasted to other showlist windows.

Settings | View



The view tab(s) define how nodes in a rundown list are being visualized within each line. Since version 5.2 it is possible to defined up to 10 different views and filter criteria when to apply which view. The number of views can be selected with the top left box. The individual views can be configured in the additional tabs “View2”, “View3”…, which appear.

If you select one of the additional view tabs the upper part of the tab looks different than for the first view:



On top left you can find the “Filter expression”. It defines whether the view is being applied. This is the case if the specified expression yields true for a node. If no expression is defined or if the expression yields false, the next filter is checked. The filters are evaluated starting with the highest view number down to the filter of view 2. Therefore: if a node is caught by multiple view filters, the highest view number is applied. If a node is not caught by any filter, the standard view 1 applies.

For writing an expression, you should check the chapter 4.5.1 – the general part about the syntax of macros. It is not possible to write a full macro here, only a single expression. And it is not possible to call any of the procedures or functions of TurboPlayer. Only the generic functions of the macro parser are available. In addition you can use the following 4 functions:

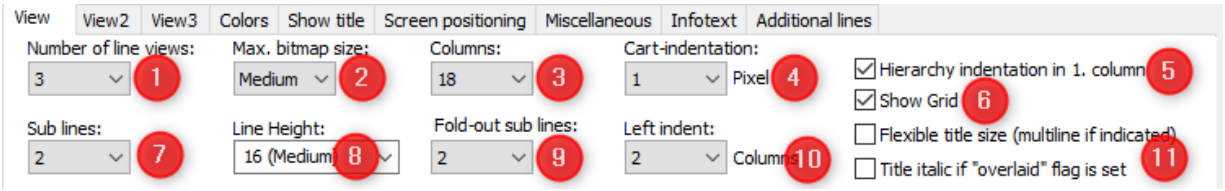
- Tagname(): the tagname of the node which is displayed in a line, e.g: Tagname() == “Group”.
- Data (<field>): the content of a metadata field of the node which is displayed in a line, e.g: Data (Time_StartMode) == “Manual”.
- TagnameOfParent(): the tagname of the parent of the node which is displayed in a line, e.g: TagnameOfParent == “Group” for all children of a group.
- DataOfParent (<field>): the content of a metadata field of the parent of the node which is displayed in a line, e.g: DataOfParent (Title) == “Commercials”.

Attention: Since version 5.4.2086.0 the Data() function returns an empty string if a queried field is not present in the metadata at all. Older versions triggered an error which resulted in an error report in the GUI log saying something like: “Failed to evaluate expression for list line view… Unknown condition function “Data”.”

The two buttons on the rights side will insert filter expressions for group/story headers and for elements which have the flag “start on a transition channel” set.

The lower part of these additional view tabs is identical to the first view tab.

Now for the details of the upper part of the view tabs:



1) The number of line view was already explained above.
2) With the maximum bitmap size you can limit how big the displayed bitmaps can become. This affects all icons like: send state, start mode and type, class, loudness ear, … The really used size might be chosen smaller than the configured maximum.
3) With “Columns” you can defined how many columns are available for the views. The number of columns is identical for all line views. Therefore you have to select a number which is high enough for all views. If a view needs fewer columns than configured, you have to spread fields across multiple columns.
4) The cart indentation affects only the display of the CartElement (see below). The cart is shrinked on each side by the configured number of pixels instead of completely filling up the specified grid cell.
5) If you activate the hierarchy indentation in the first column (which is recommended if you have groups or stories in your rundowns) the left side of each line is indented depending on the hierarchy of the node in the line. As this is difficult to explain, here is an example with a group, a story in the group and an element in the story:

6) Show Grid: if this option is activated (default), the grid will be drawn in the rundown list (or NEXT window), else no grid lines will be visible at all.
7) Each line in the list can be divided horizontally into two sub lines. If you do that, the number of available grid cells for each line is doubled. This can be configured with the “Sub lines” box.
8) The sub line height in pixels. The total line height is this height, possibly multiplied by 2 if you have 2 sub lines, plus the size of the border lines. You need not restrict yourself to the proposed numbers. You can enter any other number.
9) It is possible to unfold each line and make additional sub lines visible. Therefore you must either make the field “Open additional lines” visible which displays a “button” which will unfold or fold the additional sub lines if pressed. Or you can configure classes which are automatically unfolded without user interaction -> tab “Additional lines”.
10) The field “Left indent” is only available if you have fold-out sub lines configured. The number here is the number of columns on the left side which are not occupied with content or filled with the background color for the additional sub lines.
11) On the right side there are two options which only have an effect for nodes for which the standard view 1 is applied.
    If “Flexible title size” is checked, the title will be broken up into 2 or even 3 lines if it is too long to fit into the title cell. Therefore the font is decreased automatically.
    If “Title italic if overlaid flag is set” is checked, the attribute “italic” will be applied to the font of the title if an element has the flag “Time_Overlaid” set.

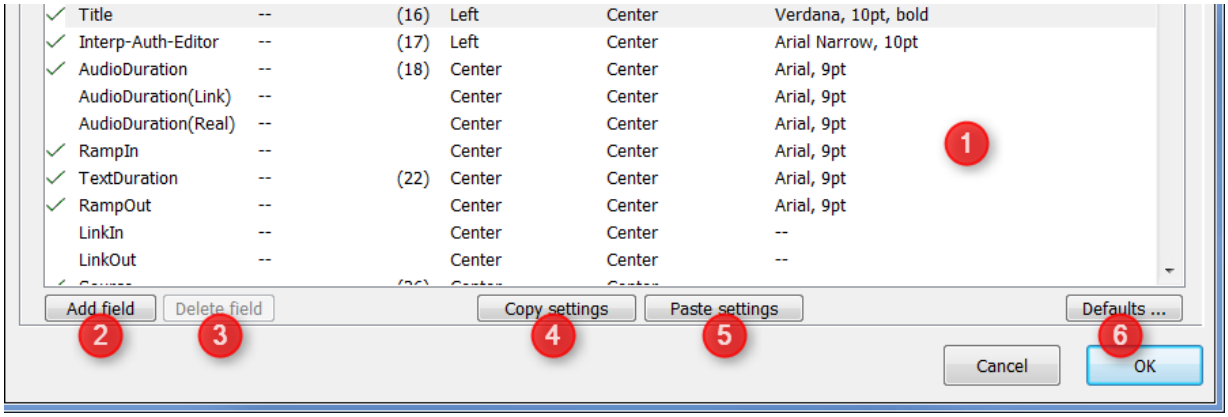
The biggest part of the view tab is filled with the preview of the cells and the list of available metadata fields. In order to make a field visible you first select the field in the list of all fields. Afterwards you click into the desired cell in the preview above of the list. If you want a field to spread multiple cells, you have to click into the top-left or bottom-right cell and expand the selection rectangle by moving the mouse while holding the left mouse key pressed. In the preview the name of the field will be drawn if the name fits into the cell. Otherwise only the number of the field is drawn and you have to look the field up in the list below. In step three you can configure details of the field in the field list, e.g. the alignment position or the font

to be used. Not all predefined fields are text and have an adjustable font. Hide a predefined field by clicking on the first check column.

These predefined fields are available:

- Number: the element number in the show (group header does not count);
- SendState: icons for Placeholder, Planned, Cleared, Sending, Sent, Skipped;
- Starttime: the planned or real starttime (if playing or played);
- StartMode: icon for Manual, Sequenced, StartOnTime, EndOnTime, Extern, Relative;
- StartType: icon for Floatig, BackwardFloating, Variable, Fixed, OffsetFloating, FixedStart, FixedEnd;
- Stoptime: the planned or real stoptime (if played);
- Intro1, Intro2: intro times;
- Outro1, Outro2: outro times
- Class: icon for the class;
- MediaType: icon for the media type (audio, video, text,…)
- FadeOption: icon for the stinger or mute-start;
- Info: infotext / comment;
- Title: ~;
- Interp-Auth-Editor: display if exist in that order: interpret, author, editor;
- AudioDuration: duration markin -> markout or the planned duration of an extern element;
- AudioDuration(Link): duration linkin -> linkout or audio duration if no link (e.g. extern element);
- AudioDuration(Real): real playout duration if an element was already played;
- TextDuration: ~;
- Dura-Start-Stop: Duration/Starttime/Stoptime. A combination column which can display one of the three fields, selecting the content is done with the macro command TP_ToggleDSSColumn, see below;
- RampIn, RampOut: the ramp string
- LinkIn, LinkOut: ~;
- Source: icon for the source (Digas, CD, etc.);
- EventsOut: icons for all fully specified generic events / events out (see macro chapter below);
- MixerSource: displays the selected source (<NoMixerSource>, <System>, <Live>, defined mixer sources in registry);
- MixerSrcSymbol: like before but an icon instead of the mixer source name is displayed;
- AudioMode: icon if audio is 5.1 or 7.1 surround sound
- Joined: icon if the flag “Joined” is being set;
- IgnoreMarkOut: icon if the corresponding flag is set and the element should not stop at mark-out;
- FileState icon;
- CartElement: the full visualization of a cart from the CartWall with all parts, fields and colors;
- MiniCartElement: a minimized version of the cart element, displaying only the colors;
- MusicMaster icon (to trigger a search for replacement songs, see MusicMaster replacement songs description)
- Open additional lines: a drop-down button for opening/closing the additional sub lines;
- Beat marker: an icon is displayed if the element has beat markers in the metadata;
- Thumbnail: if an element has an assigned thumbnail picture it is displayed;
- Overlaid: an icon is displayed if the element has the overlaid flag set (see BCSTechManual);
- Loudness: an icon with a symbolized ear in different colors is displayed, depending on the result of the loudness analysis values;
- ILK: The integrated loudness value of the analysis;
- LRA: the loudness range;
- MAXSL: the maximum short-term loudness level;
- MAXML: the maximum momentary loudness level;
- DBTP: the maximum true-peak level in dB;
- Start on transition channel: an icon if the element has the corresponding flag set;
- Distribution endpoints.

For configuring the *infotext* there is an extra tab.



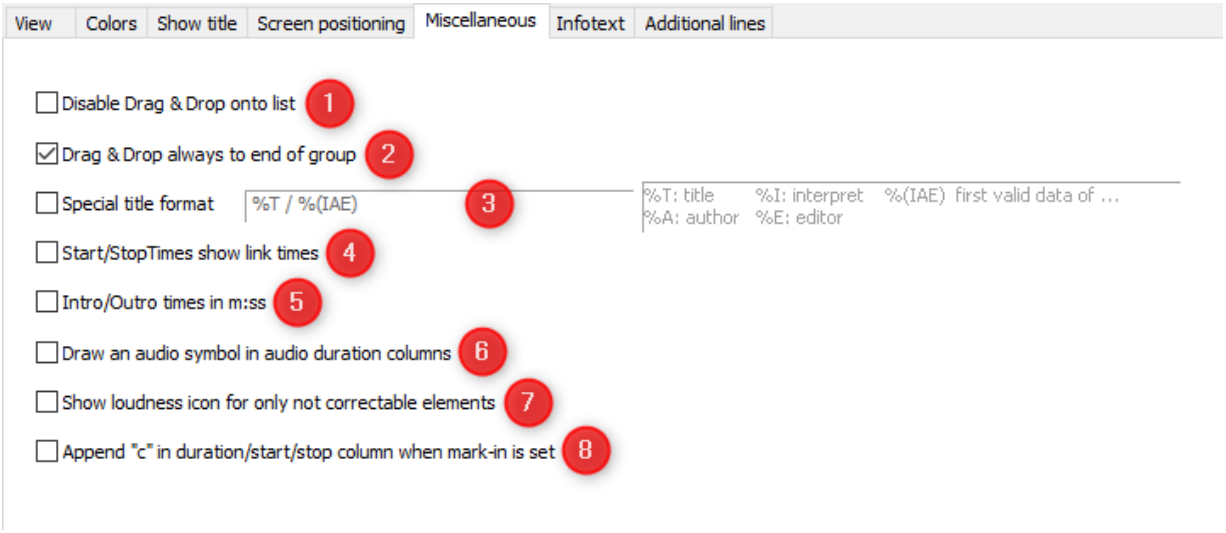
1) The list with all fields.
2) Any metadata field can be shown in the showlist by adding the xml tagname. Push the button “Add field” below the list.
3) You can also delete any of the added fields.
4) With “Copy settings” you can copy all copy the current view settings into an internal store which can be reused with:
5) “Paste settings” applies the copied settings to the current view. Be aware that this includes all settings except the filter expression and the settings which apply only for the standard view 1.
6) If you press the “Defaults” button a small menu with 3 commands appear. You can:
   - Set standard settings for a single line view.
   - Set standard settings for a double line view.
   - Remove the checks of all fields so that the view is completely empty.

Settings | Color
Adjust colors; you can specify using class dependent color for the text, as well as to colorize the class icons using UiSchemes configurations.
There is special colorization for newly inserted elements. This is only useful together with a time-info field of type "New elements" (see above) to be able to reset the colorization and it works only up to ~1000 displayed lines in the list.

Settings | Show title
Adjust colors, font for the show title.

Settings | Screen positioning
If the showlist is in standard screen mode (push corresponding keystroke) the show window aligns the first visible element and sets the cursor.
The first visible line can depend on the eldest playing item, on the latest playing item, not on playing items and/or the NEXT item. Playing items have priority over the NEXT. The item will be positioned at line <n> from top.
Example: Specify eldest playing and NEXT, and Line from top = 2: The showlist screen will always show the eldest playing or the NEXT (if no playing exist) on line 2.
The standard screen mode wil be quit if moving manually the cursor selection (by keyinput or mouse).
The cursor position has the same dependencies as the screen position.
Example: Set: Cursor depends not on playing items but depends on NEXT positions -> the cursor always is on the NEXT (if it exists).

Settings | Miscellaneous



1) If checked it is no longer possible to drop anything into the rundown list.
2) If checked, all drag&drop operations must have a group as destination or will be rejected
3) The field “Title” can be combined from multiple metadata fields. It is possible to configure a string with placeholders for:
   - %T Title
   - %I Performer
   - %A Author
   - %E Editor
   - %(IAE) the fields Performer, Author, Editor. The field being found first as filled is used.
4) Normally times are based on the mark values because this is when actions like start or stop happens. If you want to display the times based on the link values (which is how the times are stored in BCS) you can activate this option.
5) Normally intro and outro times are displayed in seconds. If you activate this option the times are displayed with minutes and seconds.
6) If checked an audio symbol is being drawn in front of the audio duration time. This would be consistent to the text duration which is always preceded with a text symbol. You should activate this option if you want to display both times: audio and text duration.
7) If checked no loudness ear icon will be displayed for elements which are compliant or correctable. Only the icon for not-correctable elements will be displayed.
8) If checked, the character “c” is displayed after the time in the special column “duration/start/stop” if the mark-in of an element is set. This is a special option to simulate the behaviour of the old NewsPlayer application.

Settings | Infotext

Here you can adjust the font, the indention, max. lines which are shown of the infotext (the comment), whether it is shown below or above its element.

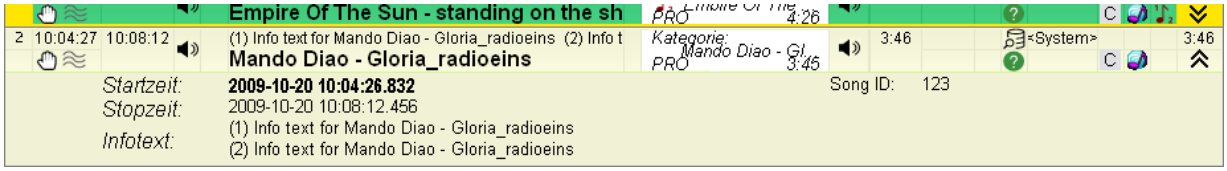
Settings | Additional lines

Here you can configure for which classes the additional sub lines (see tab “View” above) should be unfolded by default.

Menu
Resize/swap columns:
Easy to resize columns and swap columns of the showlist. Note: The variable column (the predefined field for the title per default) will automatically spread to fit the showlist window width if the sum of the columns is smaller sized than showlist window width.
Show rundown info:
Show legend:
Show infotext:
Infotext below item:
Prelisten:

Autoscroll while dragging an element over the rundown list reaching top or bottom edge (the closer the edge the faster the scrolling);

Additional lines:
See list settings | view to define the number of additional lines, an indent in count of columns. See the map to set the position of each data field. The option 'As plain text' means that the specified field name is shown – acts like a caption for a data field – and it is not interpreted as a BCS tag name.



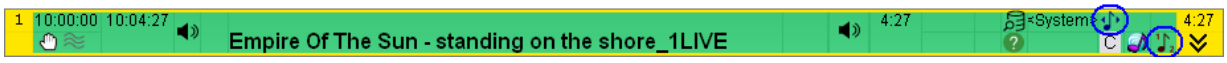
Note: Press Ctrl or Shift and click into a rectangle field in the map selects the corresponding field name in the list below. And vice versa: Select a fieldname in the list chooses the corresponding rectangle in the map.

Note: Empty lines – means its BCS data are empty – are hidden, e. g. one additional lines is designed to show the comment (info text), in case the element does not have a comment this one additional line is hidden.

Note: Do not forget to define a position in the element's grid for the open/close additional lines arrow icon; otherwise the additional lines can only be opened by a shortcut. To open/close all add. lines press Ctrl or Shift and click on the arrow icon (depends on current open state of the element).

Allow **multiple** comma separated **BCS tag names** (first valid data is shown, see list settings | view), e. g. Fieldname is 'Music_Composer, News_Author': First the data of the BCS field 'Music_Composer' is shown (if valid), second the author's name is shown.

To illustrate the new features of CartBeat and CartMotion the element can show these features; An element has the CartMotion feature (icon: ♪) if its Audio Mode is '2.0 [with stereo]', means it has two stereo audios within the audio file.
The beat marker icon 1 2 is shown if the predefined field 'Beat Marker' is on (see list settings | view) and beat markers are defined for the element (see MTE, DBM).



2.3.10 Next

This window type shows the NEXT item of all specified rundown lists (see settings).

2.3.11 Stacks

Stack windows are derived from the showlist window.

To operate on stacks what to load, which defined stack to assign to the window, the rundown info must be shown (a header line at the top edge of the window), -> menu.

Select the stack which will be shown in this window (menu), select a node to be loaded into the stack (stack mode must be free).

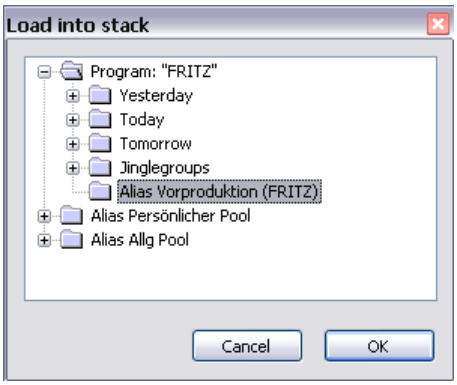
Change rundown mode by mouse right click on the rundown mode column. Rundown modes = Free (load any node into the stack), Clipboard (temporally node), Track (any track which follows the main track or the main track itself).

Change stack mode by mouse right click on the stack mode column. Stack modes = Sequenced (playout considering item order), Once (planned -> playing -> sent), Unlimited (planned -> playing -> planned)

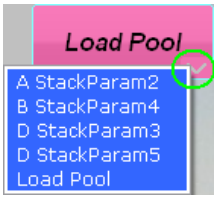
2.3.12 StackButtons

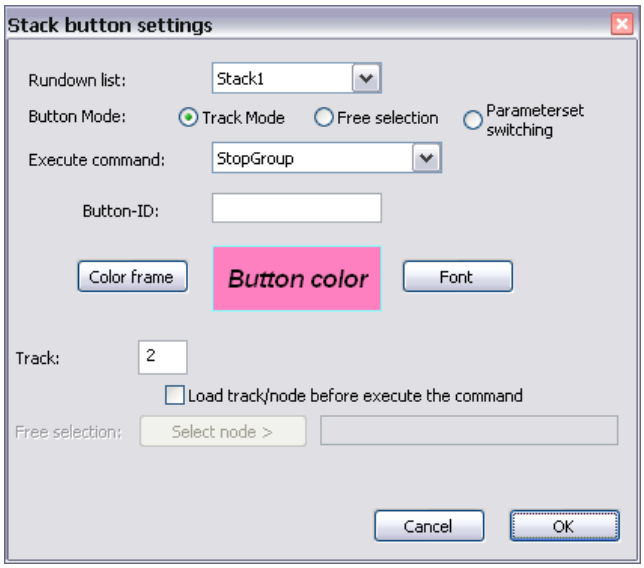
Three modes exist:

1) Track Mode: Specify the track number. Each button press this track will be loaded into the specified rundown list. The button title shows the name of the track concatenated with the current show (main track), the button title follows the main show, e. g. if the next show will become the active one, the button shows the name of the track of that show.
2) Free Mode: In the settings specify a node (Program, Yesterday, Today, Tomorrow, Jinglegroups, PreProductions, Personal Pools, General Pools). This node (and its subnodes) will be shown if the button's arrow is clicked, select the node you want to load into the rundown list. Each successive button press reloads this node.



3) Switch parameter set: If the button's arrow is clicked all events (\EventsOut\ in the registry) are shown which have the value 'RundownListType' specified and matches the button's rundown list type (or '*' to match all). This event (macro) will be executed if the button is pressed.



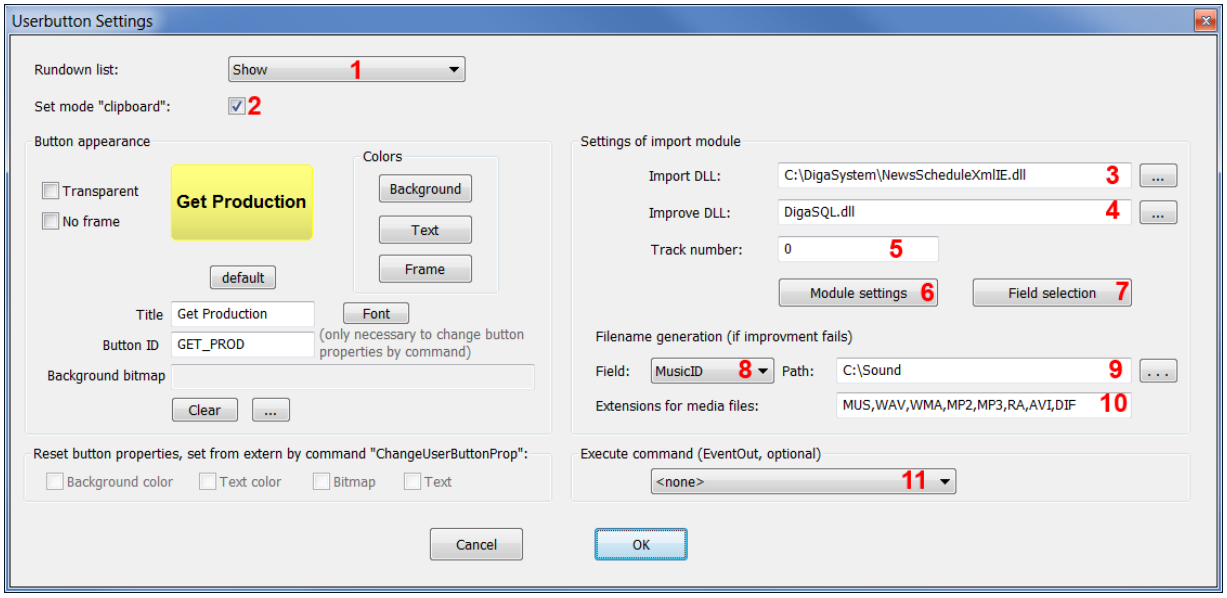


Button-ID: Define it to change button properties (color, text) by TP_ChangeUserButtonProp() macro.

Execute command: Specify a command which will be executed if the button is pressed (in track and free mode).

2.3.13 ImportButtons

Import buttons are used to replace the content of a rundown list with the output of an import module. Import modules are DLLs of the BroadcastSystem which are used mainly in DigAIRange and BUS to read data in foreign formats and convert the data into BCS-XML, e.g. the FlexIE.dll for sections-string formatted data or the NewsScheduleXmlIE.dll for Sepl-XML data. In the settings dialog of an import button you can configure the import module:



The "Button appearance" and "Reset button properties" are identical to a user button.

1) RundownList: The button is assigned to this rundown list. The import replaces the content of the rundown.

2) Set mode "clipboard" If checked, the stack mode of the rundown list is set to "clipboard" before importing. This means: the new content will only be hosted within TurboPlayer. If not checked, the current content of the list will be replaced – which might well be one of the BCS nodes. In case of a show/track being in the rundown only the current "main show" will be replaced, future or past shows will not be touched.

   Pay attention to allow the change of the stack mode if you check this option. Therefore, in the registry the value TurboPlayer\RundownLists\...\StackModeLocked must be set to "No" or "FALSE". At least for the rundown lists "Show" and "Jingles" this value is by default "yes".

3) Import DLL The filename of the import DLL.

4) Improve DLL The filename of the improve DLL. At the moment only DigaSQL.dll can be used as improve DLL and it is sufficient to specify the name "DigaSQL.dll" without a path because the DLL is loaded already. This DLL performs a database lookup to improve (complete) the element data. If you do not want to perform a database lookup leave the field empty.

5) Track number A track number to be imported. If the import data does not contain any track, the specified track number is assumed to be present. In case the destination rundown hosts shows and uses a specific track number you should use the same track number.

6)	Module settings	Click onto this button to call the settings dialog of the import module. Most of them are described in BCSTechManual.doc.
7)	Field selection	Click onto this button to call the field selection dialog. It is only needed if you perform a data improvment and allows to specify for each field whether it is taken from import data or from the database. See the chapter about import in BCSTechManual for more information.
8)	Field	These three fields are used for an "emergency filename generation". It is done if the database lookup fails and no filename is specified in the import data. The content of the field (e.g. MusicID) is taken as filename, the path and the extensions for media files are used for a file search. This file search is optional, remove the field if you do not want to use such a search.
9)	Path	
10)	Extension for media files	
11)	Execute command	In this optional field you can select one of the defined out-events which is executed during the import. In principle the event is triggered after the import but as the exact timing is not defined do not try any operations which interfere with the direct actions of the button (changing the rundown content, setting the stack mode).

Some additional hints:

- A merge operation like in DigAIRange- or BUS-import is not available. The rundown as it is yield by the import module always replaces the current content of the rundown in TurboPlayer.
- In case you want to import into a BCS node you need at least a BCS build 238 or newer. Otherwise, the root node of the import might loose its data, making it unusable possibly.
- If you want to import rundowns as they are created with the ScheduleScreen.ocx in MultitrackEditor use the NewsScheduleXmlIE.dll (build 5 or newer). In the DLL settings specify the output file of the ScheduleScreen.ocx (parameter Multitrack \ ScheduleScreen \ CreateFileSchedule) as input file, or the output path as import path.

2.3.14 Statusline

Up to now no settings.

Displays icons for the connection states (disturbed or normal) for the connection to the internal BCS and the engine of the TurboPlayer.
Logs error messages and hints which can be displayed in sum (restricted to the latest 40 messages) by dropping onto the text window or by menu ‘Report all messages’ or by double click on the message field.

2.3.15 Button

Two buttons are available. The ‘Trashcan’ button and the ‘Update start/stoptimes’ button which updated the start/stoptimes in the rundown lists.

Up to now no settings.

2.3.16 Remote TurboPlayers

The user is informed about the connection (o.k. or disturbed) state of the remote TurboPlayers, defined in the DigaSystem registry under

\TurboPlayers\Communication\RemoteTurboPlayers\Player1..n .

2.3.17 Filters

Filters act on the list content and can be a combination of the five single filters:

- Placeholder (all items with sendstate 'Placeholder' will be hidden)
- Group is empty (all group header lines of empty groups will be hidden)
- Mediatype is 'Text' (all items of MediaType 'Text' will be hidden)
- Flag 'Hide element' is set (item will be hidden if their attribute 'hide' is set)
- Matches expression (all items that match the expression will be hidden). The functions Tagname() and Data (<field>) can be used here to create filter expressions. See section Showlist for more information where this feature is also used for different views of the rundown list. **ATTENTION:** not all metadata fields are available for the function Data(), since not all of them are loaded in TurboPlayer interface. Most of the usual ones are indeed there, but if some filter criteria don't work with any "unusual" metadata field, this is probably due to inavailability of this metadata field in the interface. This can be added as a feature request.

Additionally, there is an option which allows to make children of filtered (previously collapsed) groups or stories visible if the group or story is filtered. Per default, children will also be hidden (this setting is checked) and it will only have any effect if the group or story is collapsed (else children will simply be left there).

Notice that since build 2209 it is possible to disallow rearrangements in the rundown if filters are applied. See section Main Settings, "Settings | Miscellaneous" for more details.

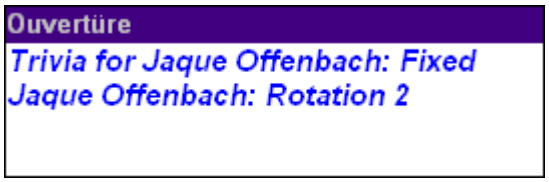
Menu - > Settings

Set filter rule, fonts and colors. Specify the rundown list the filter act on.

2.3.18 Prompter

If MusicMaster is activated (see MusicMaster Settings) the trivia text (song notes) according to the prompter settings is shown in the sub window.

Trivia notes window of type 'prompter':

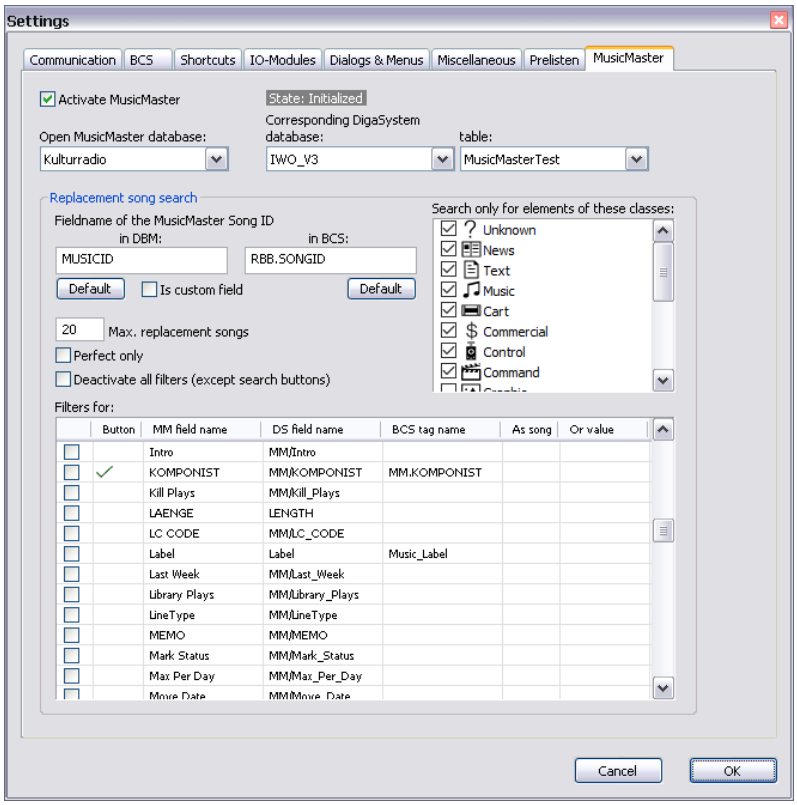


Menu - > Settings

Adjust the rundown and which element's trivia text should be views (either the NEXT or the cursor selected element).

2.3.19 MusicMaster

Define **filters** (see main settings | Music Master)



Activate MusicMaster: TurboPlayer establishes a connection to the Nexus MusicMaster (MM) server.

Open MusicMaster database: select a MM database that will be opened on initialization.

Corresponding DigaSystem (DS) database/table: An aligned DS table must exist; specify the database and table that contains the MM elements on DS side.

Replacement song search:

Fieldname of the MM Song ID in DBM and in BCS: Specify here on which field in the DS database/BCS the MM song id is mapped, e. g. usually the MM song id is mapped to the DS field MUSIC_ID (Default) and to the BCS tag Music_MusicId (Default). If the field is a custom field set this option (required for database search requests).

Max. replacement songs: Up to 99 suggestions for an element can be done.

Perfect only: Flag that instructs the search for replacement songs must hit the MM rules exactly. It restricts the search (fewer results).

Deactivate all filters: filters which are specified in the list below are deactivated, but not active criteria buttons in the GUI.

Search only for elements of these classes: Limit the search to elements of these classes; only for those elements the MM icon is shown (must be set to on in rundown list settings | view) in the element grid – a click on it starts the search for replacement songs.

In the lower list all available MM fields and their mapped DS fields are shown. Set the first column's check box state to enable this field to be a filter for the replace search, further fill in the BCS tag name and whether the filter value is the same as the current requesting song's value of this field or type in a certain value. Those checked fields (first column) define the filters used besides the criteria buttons in

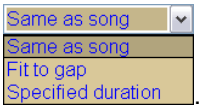
the GUI each time the MM icon of an element in a rundown list is clicked – except the option 'Deactivate all filters' is set.

In the second column you select those fields which can be chosen later in the replacement search criteria buttons (define the range of selectable fields).

Replacement song criteria buttons:

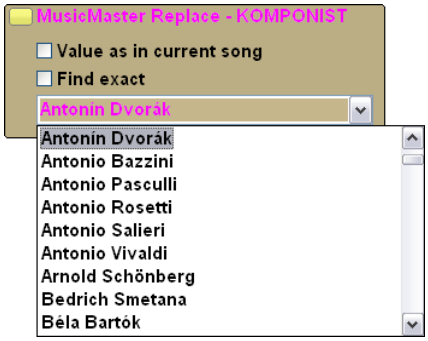
There are three criteria button types: Rule, Length and Metadata that work as filters when searching for replacement songs (in addition to the filters set in main settings | MusicMaster).
To activate/deactivate the criteria press the button in the left upper corner.
All criteria of the buttons are AND combined.

Rule: can be set to *strict* or *breakable* (more tolerable). This rule adjustment takes precedence over the 'perfect only' option set in main settings | MusicMaster.

Length: Chose on of 

. If '*Same as song*' is chosen the song's duration is shown in the right text box and this length indicates the search criteria; ditto '*Fit to gap*': Here the gap is calculated beginning from the requesting song to the next fixed element in the rundown – either a fixed element or the show end. '*Specified duration*': Here you have to enter a desired duration in the right text box.
The search for replacement songs considers the given length and sets a filter for the length.

Metadata: First set the metadata field (see criteria button settings) whose value defines the filter (in the picture above 'Komponist' resp. Composer in the DS database).
Either the value is the '*Same as in the current song*' – then the current song's value of this metadata field is displayed in the text box – or type in a value in the text box or chose one of the available* values in that table (see picture below).
(* Available values are offered for these DS metadata fields: Author, Editor, Title, Category, Class, Performer, Composer, Correspondent, Publisher, Broadcast, Speaker, Intensity, Customer, Language, Presenter, Product, Productgroup, Program, Project, Type, Carrier (the Label). For all other values you have to enter a value.)
If the option '*Find exact*' is on the value must exactly match the found replacement songs.



2.4 Layout Templates

In TurboPlayer it is possible to load GUI layout templates to accelerate changing between different GUI's for a given configuration. The idea is that DAVID may provide a package with a set of predefined GUI layouts, although creating your own layout package wouldn't be that difficult. Changing the layout requires a restart of the GUI and it will affect exclusively the GUI on which the layout is loaded.

2.4.1 Creating layout templates packages

Concerning the data structuring, the data for a GUI layout package includes the following elements:

- A single DigaSystem parameter file with subkeys for each GUI layout (can be easily done in Admin).
- Text or RTF files with a description of a given layout, one description file for each layout.
- Image files (bmp, gif, tiff, jpg, png) containing a preview for a given layout, one for each of them.
- Additional resources, especially images, as they are required by the layouts and referenced by the parameters. For example, this is the case if a button has some bitmap assigned to it. When copying layout data packages, it would be useful to concentrate every resource used by the GUI in a layouts folder.

Some rules must be observed when creating layout packages:

- There must be a single root folder containing all stuff of the package (the "package root folder").
- There must be a single DigaSystem parameter file called "TurboPlayerGUIs.par" in the package root folder. If this file is not present, no layout will be displayed or be made useful at all.
- Concerning the parameter file, all GUI layouts must be located in a top-level key called "TurboPlayer" and must start with "GUI", e.g. *GUITurboPlayerStandard*.
- There must be a parameter under the TurboPlayer top-level key called "OriginalFolder". This is a string and must be the absolute path of the root folder on the computer on which the package was created. For example, if the layout folder was created under **C:\DigaSystem\TP_LayoutsFolder**, then the original folder would have exactly this path in it.
- In each "GUI..." key the following four values must be present, so that the display of the available layout becomes more friendly and resource data may be found:
    - "Name": A user-friendly name which is being displayed in the selection dialog in the list of available layouts in a given package.
    - "Resolution": The resolution in a format like "1920x1200" with the original resolution in pixels for which the GUI was created. There is no validation concerning these dimensions, which means the one creating the package is responsible for providing correct values. On the other hand, if a GUI layout is really applied, the original values of the layout will be used, no matter what is written in this parameter.
    - "Description": The path to either a RTF or a TXT file containing the description for this layout. This will be displayed in the layout selection dialog.
    - "Preview": The path to an image file containing the preview for this layout. The image should have the original screenshot size and will be resized by TurboPlayer when being displayed.

- Besides these four values in each GUI key, there must also be all layout values that define a TurboPlayer GUI layout. These are the same ones present under “TurboPlayer\GUI” key and follow the exact same names and structure. As it will be discussed in the next section, these values will be copied from this GUI key to “TurboPlayer\GUI” key as they are stored. Thus, they must obey the same structure.

All resource files can be located directly in the package root folder or in subfolders. If such a resource is being referenced within the parameter file, it will originally point to an absolute position within the file system of the computer where the layout was created. This will be corrected by TurboPlayer before the layout is used. This is done when the configuration parameters are copied to the own local registry. All references to the original folder are replaced by references to the corresponding folder on the destination system where the layout package is installed. That’s the reason the OriginalFolder key must be present and if not, will not allow layouts to be effectively applied.

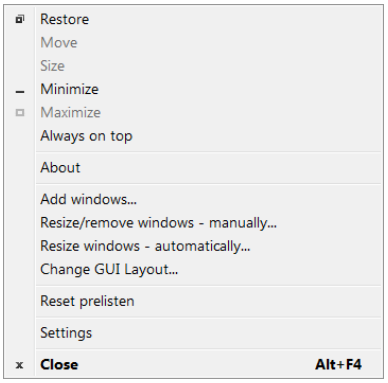
Concerning the installation of the layout package on the destination computer, the administrator must simply put the whole package into a single folder of his choice - probably by unpacking a ZIP file delivered by DAVID and point to this folder in the layout selection dialog in TurboPlayer (covered in the next session).

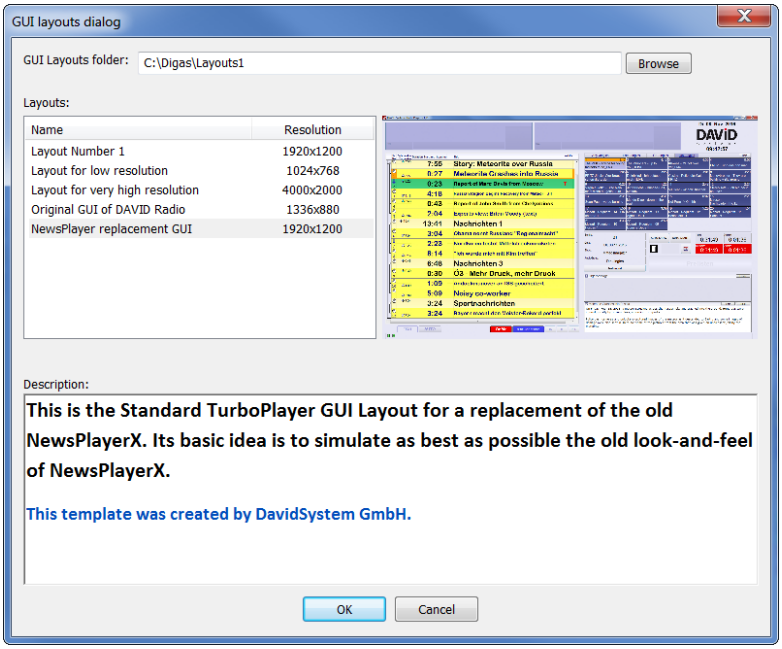
2.4.2 Using layout templates packages

In order to use a layout package, first of all it must be uncompressed/copied to a new folder. Normally it would be under DigaSystem folder, but this is not a rule and it can be chosen as desired.

After copying the layout folder to the desired location, right click on TurboPlayer’s title bar and choose “Change GUI Layout…” in the menu (picture below). The GUI layouts selection dialog will be displayed.

If it is the first time you are opening this dialog (or if the layouts folder was not configured yet), prior to displaying the dialog, a folder selection window will be shown querying the user about the root layouts folder location (where the TurboPlayerGUIs.par file, as well as every other used resource, are supposed to be found). Since the dialog itself has nothing to offer without a layout folder, if the user doesn’t choose something, the dialog will be closed. If a layout folder has been previously defined, it will be saved as any other setting and will be used on the next time. Notice that the layout folder will only be saved if the dialog is closed with the OK button. Cancelling the dialog will discard also any changes made to the layouts folder location.





The main elements shown in this dialog are the ones already presented in the previous session. First of all, we have a control where the currently selected layouts folder is displayed, as well as a button to open the folder selection again, allowing the user to change the layouts folder location. Once more, if this is changed, it will only be saved if the dialog is closed with the OK button.

On the left there is a list showing all layouts found in the parameter file. On the first column of this list the user-friendly name is displayed (notice that if a name was not given, we will have a white field there) and on the second column the resolution for this layout as configured in the parameter file. On the right a preview image is displayed for the selected layout, if this image is available and readable. This image will be resized to fit the dialog. Below in the dialog a text control is displayed showing the description for the currently selected layout, once more if available and readable.

When the dialog is opened, the settings from the specified root folder are read and everything found is loaded and prepared to be displayed. The list on the left shows the available layouts. When the user selects a layout in the list the image and description file will be shown, if possible.

Pressing the OK button will show the user a message, informing that in order to apply the selected layout, TurboPlayer will be restarted, and asking for a confirmation. If the user chooses “Yes”, the dialogs will be closed and TurboPlayer will be restarted to apply this new layout. If everything succeeds, the new layout settings will be copied to the local registry and the new controls configuration will be displayed. It is important to notice that prior to copying the new settings, the old “GUI” key will be renamed to “GUI_Old”, so that the old settings are still accessible - if desired. If you are sure, you won’t need it, feel free to delete this key, it is just a backup. Applying new layouts in sequence will make this “GUI_Old” key always be overwriten by the current configuration before applying the new layout. If you have a non-standard configuration running (meaning: the current configuration key is not “TurboPlayer”) the new layout will be copied into the “GUI” subkey of the currently active configuration.

If the user chooses “No”, TurboPlayer won’t restart, and the selected layout won’t be applied. However, the dialog will be closed, and any possible change of layout folder will be saved. Choosing “Cancel“ will abort the operation, closing the message box and the dialog keeps being displayed.

2.4.3 GUI Layouts and global settings

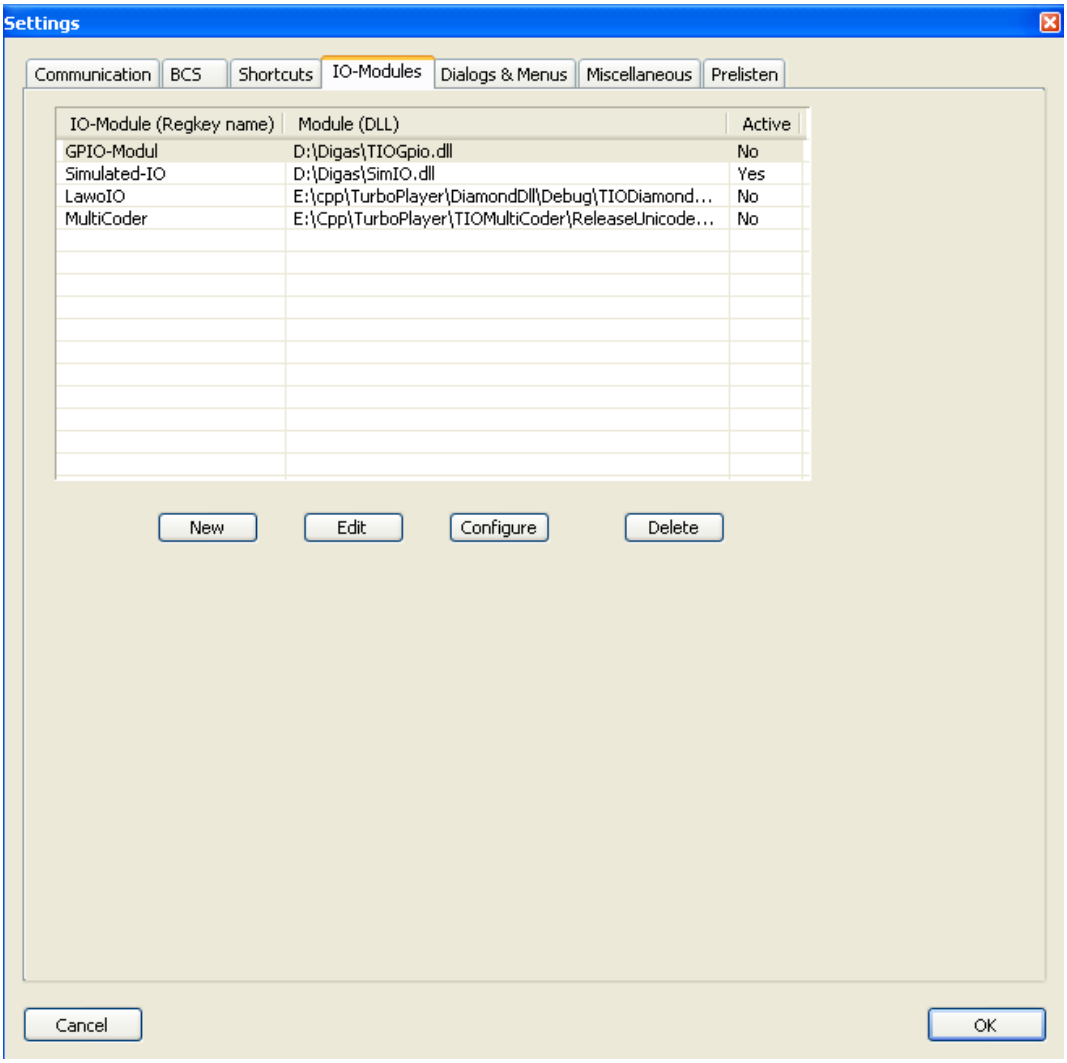
Pay attention when you want to use GUI layouts when TurboPlayer GUI settings have already been set in the global DigaSystem registry. This might not work! The mechanism for GUI layouts will create the copy of the template parameters in the local registry. If you have already some settings in the global registry, the sum of all settings from both locations will be applied. This can create a funny mixture of settings. We recommend to not use GUI layouts when TurboPlayer GUI settings have been defined globally; or at least only with care and a good selection of global parameters.

3 IO MODULES

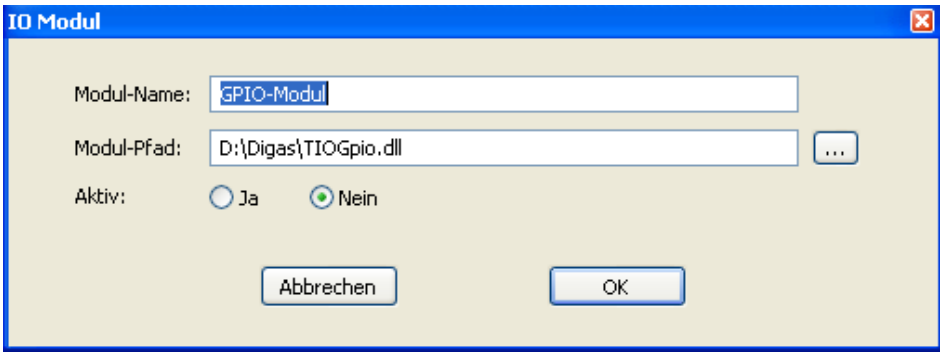
3.1 I/O Modules – General Information

There are some different kinds of hardware, working together with Turbo Player. I/O modules (DLLs) exist to act as "driver" between TurboPlayer and the hardware. There are some specific modules (like for LAWO mixing consoles) or for the MultiCoder. Also a general module for GPIO (General purpose input / output, relays and optocoupler) is available: The TIOGPIO.DLL. Most mixing consoles can be controlled via this module.

All I/O modules provide a settings dialog. It can be called from the "I/O-Modules" tab of the TurboPlayer settings dialog which can be called by a right click on the menu bar in Turbo Player. A roll-down menu appears where you have to click on “Settings”.



The "Edit" button calls the general settings dialog (independent of the I/O module).



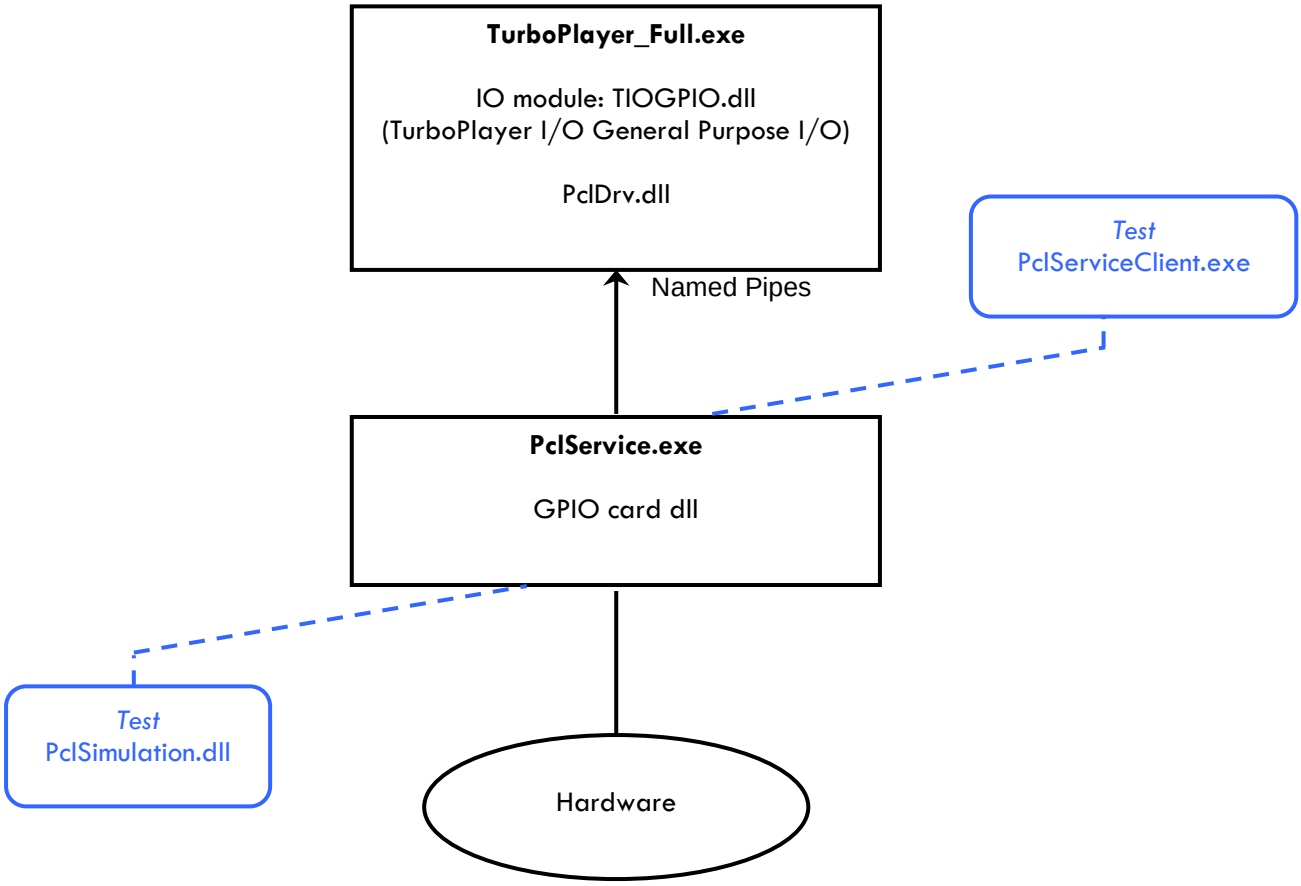
Here you can enter the displayed name, the filename of the I/O module and whether the module is active. To deactivate a module, you can simply click on “No” in the “Active” column, too.

The "Configure" button calls the specific settings dialog of the I/O module. Some modules are described in the chapters below.

TurboPlayer stores all settings in the DigaSystem registry in the key "TurboPlayer\IO\<Regkey name>". Please do not edit the I/O module specific settings within the registry.

3.2 GPIO Module

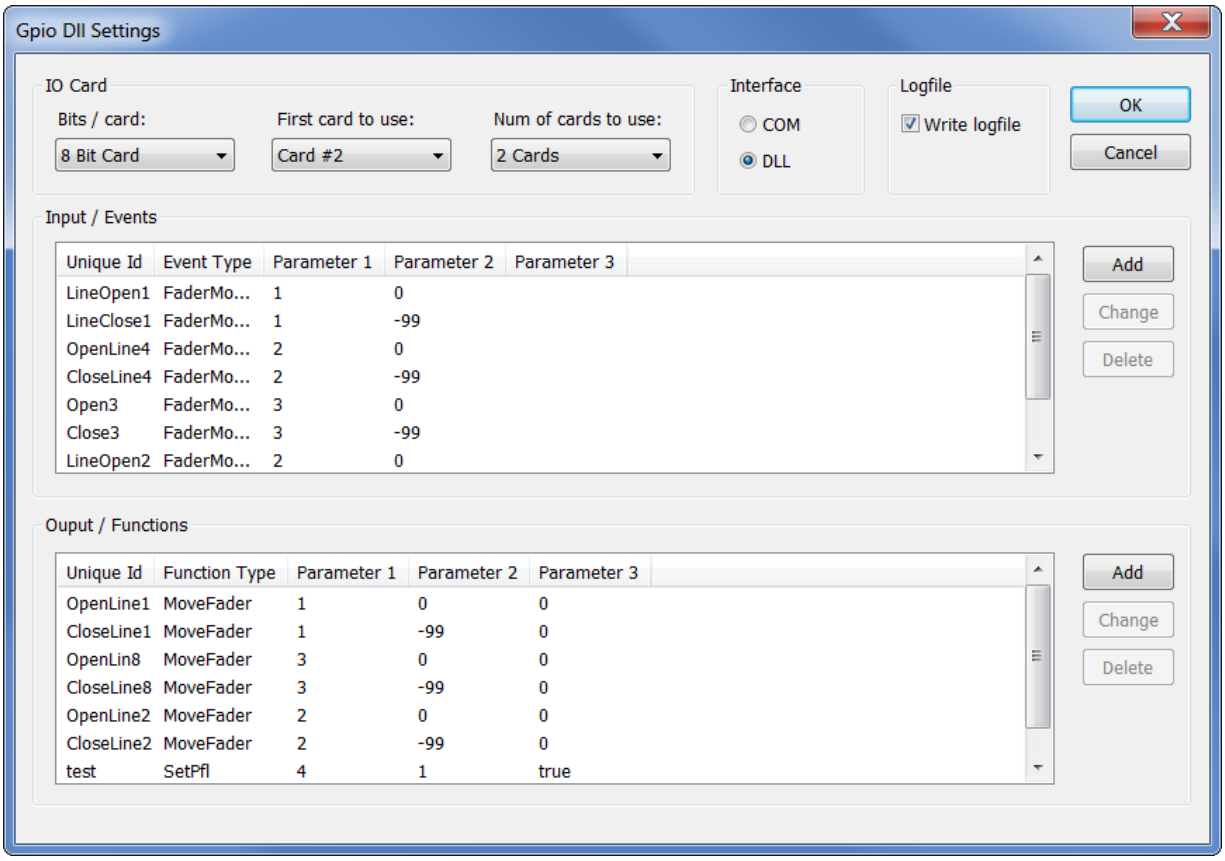
3.2.1 I/O Modules – TurboPlayer, working with GPIO – Overview:



The “GPIO card dll” is one of the dlls needed to operate a specific GPIO card, e.g. the “PCI-1760.dll” or the “PclAxiaLiveWire.dll”.

3.2.2 GPIO Settings dialog

Here is the dialog:



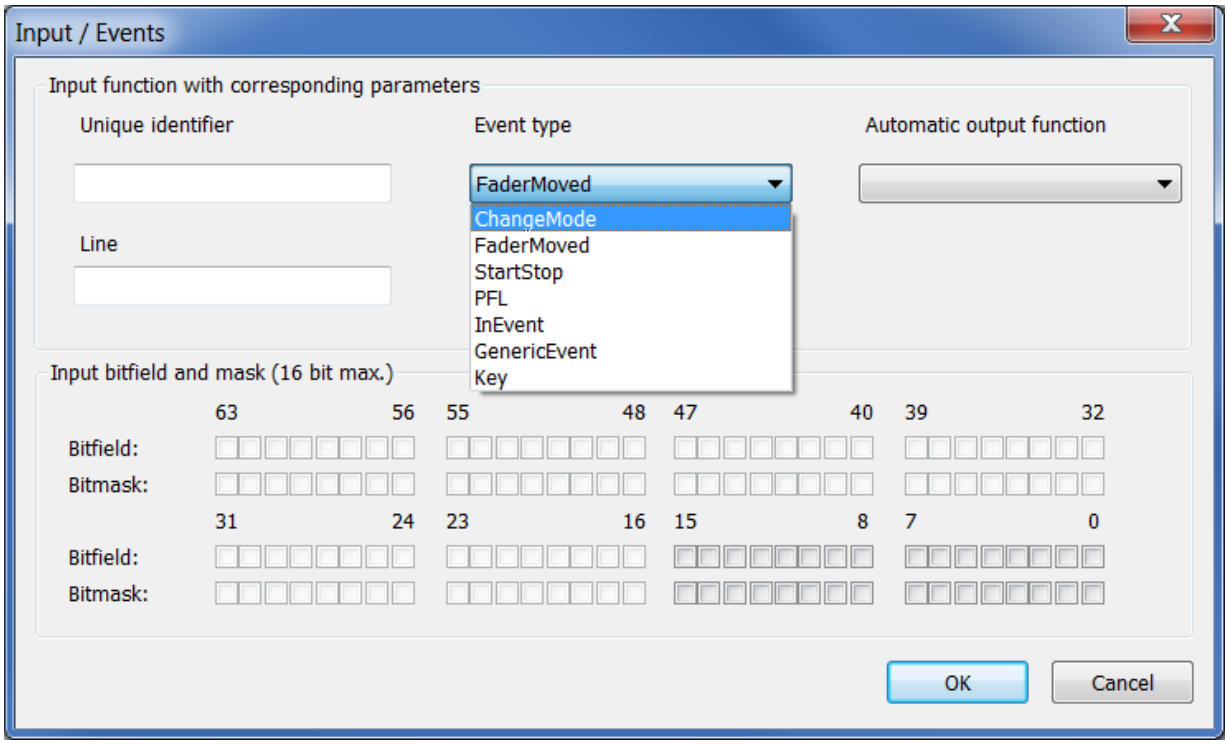
In the “IO card” group you can select the GPIO card(s) to be used. The type of card is selected within the configuration dialog of PclService.exe, here you have to specify how many bits are available on the card(s). If you want to use multiple cards they must all be of the same type! It depends on the manufacturer of the cards how they are enumerated - or in other words: how they get their device IDs. Here in the GPIO settings dialog (and in PclService) the cards are simply enumerated, starting with “1” for the first card in the system. In the settings dialogs for the input/output functions you will see that the available bits of each card are added. The bits of all cards are presented as a consecutive list of bits.

For the interface "DLL" should be used always. "COM" is only for backward compatibility with old PclComServers which could communicate only via COM. “DLL” means that the TIOGPIO.DLL will make use of the “PclDrv.dll” to communicate with the PclService via named pipes. Therefore, this dll must be available in the program directory or in the Windows system directory.

Choose “Write logfile” to write a protocol. The file is written to the temp directory of the current Windows user (e.g. C:\Users\<Username>\AppData\Local\Temp).

We distinguish between input and output events. Input events are triggered by the hardware and are processed by TurboPlayer. Output events are triggered by TurboPlayer and control the output relays of the IO card.

Click on “Add” to create a new Input/Event or Output Function.



A unique identifier must be given to each event. This is only the registry key name and is not used anywhere.

Some different types of events can be created: ChangeMode, FaderMove, StartStop, PFL, InEvent, GenericEvent and Key (see below). Key is not used at the moment.

Every event can be linked to an “Automatic output function”. This is necessary when the connected device expects an answer to an input signal. For example, when someone opens a fader on the console the console signals a "line is open" (=FaderMoved), but the line is not really open and no sound is audible. The console requires an explicit command "open the line" (=MoveFader) to really open the line.

Something similiar is sometimes neccesary for the output events. For example, if TurboPlayer sends the signal "MoveFader" it expects an answer from the mixer console that the fader has been moved (=input event "FaderMoved"). If the mixer console does not generate such a signal, or if the signal is not connected to the IO card, it is necessary to generate a fake signal for TurboPlayer to be happy. This is done by creating an input event "FaderMoved" and linking the output event to this fake input event. This means: the IO module simulates the event which is not really received from the console.

If you forget to configure such an automatic input/output event strange behaviour will be the result. If you forget the "FaderMoved" in the example above the behaviour is quite clear: TurboPlayer sends the "MoveFader" command but as no answer is received the start of the current element does not work because TurboPlayer thinks, it was not possible to open the line. If you encounter such a problem, it might help to look through the technical logfile of the engine or to print messages 70002 (=MoveFader) and/or 70010 (=SetPFL) to the message log. You should see the typical ping-pong between TurboPlayer and the console - e.g. Turbo sends "SetPFL" and the answer is "PFL was set". TurboPlayer expects an answer to the commands "MoveFader" and "SetPFL". The answers your mixer console requires must be looked up or evaluated.

One important hint regarding the line mode of TurboPlayer (registry parameters "TurboPlayer\Lines\Line-X\Mode" and "...\AutoMode"): for the mode "Active" (meaning TurboPlayer can control the line) a specialization of this mode exist: "Active/OnOff". It should be used when there are no motor-faders and the mixer console can only report an open/close (or on/off) state for the line. For an GPIO interface this is naturally the case because only single bits are used to transport the information. Therefore, you should always use the OnOff specialization in conjunction with GPIO (as long as "Active" is used). This changes the behaviour of TurboPlayer in some details, e.g. it decides wether fades are executed though an element was started manually by opening a fader.

3.3 Events

Hint: This chapter is written for the GPIO module, but it does apply for all IO modules.

3.3.1 In Events:

The first possible event is called **ChangeMode**. It can be called from the I/O module to change one of the internal modes of TurboPlayer. There are two fields to be completed: *ModeName* and *NewMode*.

Mode Name is the name of the mode to change. *NewMode* describes the new mode, which is switched into by the event. The following fillings are assigned to the respective fillings for Mode Name:

Mode	Possible values
Activation	Passive, Live assist or Automatic
Pause	On or Off
PlayFadings	On or Off
SelectSources	On or Off
FaderStart	On or Off
ButtonStart	On or Off
Ghost	On or Off
OnAir	On or Off
ServerConnection	Connected, Rehearse or Standalone
InsertFiller	On or Off
IgnoreMarkOut	On or Off
BeatExact	On or Off
SyncStart	On or Off
FreeShowlist	On or Off
FaderStopsPFL	On or Off

Note: Some of the fillings are explained in detail in chapter “concepts”.

Another event is called **MoveFader**. This event moves a fader, as the name says. The two fields to be completed here are *Line* and *Level*

Mode	Explanation
Line	describes the Line number, the defined line has to be filled in.
Level	describes the end level in dB, whereas -99 mean, that the fader is closed.

Note: With GPIO only On and Off states can be signalled. This might be a "fader is open/closed" or "line is opened/closed". In the communication with TurboPlayer level values are used.
The assignment is: 0 dB is open
-99 dB is closed

Note 2: A mixer console should answer with this event onto the output event "FaderMoved". If the mixer console does not answer with this in-event, this event must be defined nevertheless, and it must be specified as "automatic input event" within the setting of the output-event of "MoveFader".

The next event, called **StartStop**, signals a start or stop command. The input fields are called *Line, Channel* and *Command*

Mode	Explanation
Line:	fill in the line number
Channel:	fill in the logical channel number
Command:	fill in one of the string literals: Start, Stop, or TogglePlay

Note: The module should send either command for lines or channels, but not for both. The unknown parameter should be set to -1.

Attention: You will notice that there is also a FaderMoved function. The start/stop function should only be used by keyboards which have a real start/stop button. The FaderMoved function should be used by mixers. The engine will decide whether to start/stop an element automatically, depending on its internal states.

In the following, the event **PFL** will be described. It signals a change of the PreFaderListening state of a line. *Line, PFL Number,* and *PFL On (Not Off)* are the required fields in this case.

Mode	Explanation and possible values
Line:	fill in the line number
PFL:	fill in the Number of PFL bus, starting with 1
PFL On (Not Off):	shows, if PFL was activated or deactivated, fill in true for on and false for off

Note: A mixer console should answer with this event onto the output event "SetPFL". If the mixer console does not answer with this in-event, this event must be defined nevertheless, and it must be specified as "automatic input event" within the setting of the output-event of "SetPFL".

Another event to choose is called **InEvent**, which simply signals an in-event. Just an *Event Name* is needed, besides the “Unique identifier” and if necessary the “Automatic output function”.

Mode	Possible values
EventName	define the name of the event

Note: In-events are used in conjunction with the external start mode of elements only (see chapter "Generic events". The administrator must edit the list of possible events in the registry (in the key "TurboPlayer\EventsIn"). Here the parameter "EventName" is one of those registry keys.

The last type of in-event, that is currently used, is called **GenericEvent** (see chapter about events and macros below). Similar to the “InEvent”, this event signals a generic event. In principle generic events can be either internal functions (with known names, starting with “TP_”) or out-events listed in the registry. Nevertheless we recommend to use only out-events as they are listed in the registry and to define the executed commands (macro) within the registry.
EventName and *Parameter* fields can be filled in.

Mode	Explanation and possible values
EventName	define the name of the event
Parameter	can be any parameter string, but should be empty. Define the parameters within the registry.

3.3.2 Out Events:

The first kind of out-event to be described is **Reset**.

The *unique identifier* must be given. This event is signaled when TurboPlayer executes a reset. Possibly you want to reset the device the I/O module controls.

ModeChanged has already been described at the In events list. But here it is called to inform the I/O module about changes of the internal modes of the engine.

MoveFader has already been described, but there are two more parameters, when Move Fader is an Out event: *Duration* and *Relative*.

Mode	Possible values	Explanation
Duration		means the duration of the move in milliseconds
Relative	TRUE	means that means the fader is moved relative to its position in the moment of this call
	FALSE	means that the fader is moved to an absolute position

The next event, **PlayStateChanged** shows, that the play state oft an element has changed. The *line* number and the *new state* you want to evaluate must be given.

Mode	Possible values	Explanation
Line	the line number	
State	BeforeStart:	This special play state change is sent immediately (the defined "OpenDelayActive" time) before a start. You can execute anything which must be performed before audio is being played.
	Start	
	Stop	
	Pause	
	Prepare:	The state prepare means, the element has been loaded into the player for the specified channel and is ready to be played.

SetSignalProcessing is used to change the signal processing (typically music/speech) for a line. The *line* number and the *Processing* are required.

Mode	Explanation and possible values
Line:	The line number
Processing:	A processing identifier can be defined in the DigAS registry in the key "TurboPlayer\ChannelsOut" as required.

GenericEvent and **SetPFL** (as PFL) have already been described at the In Events list.

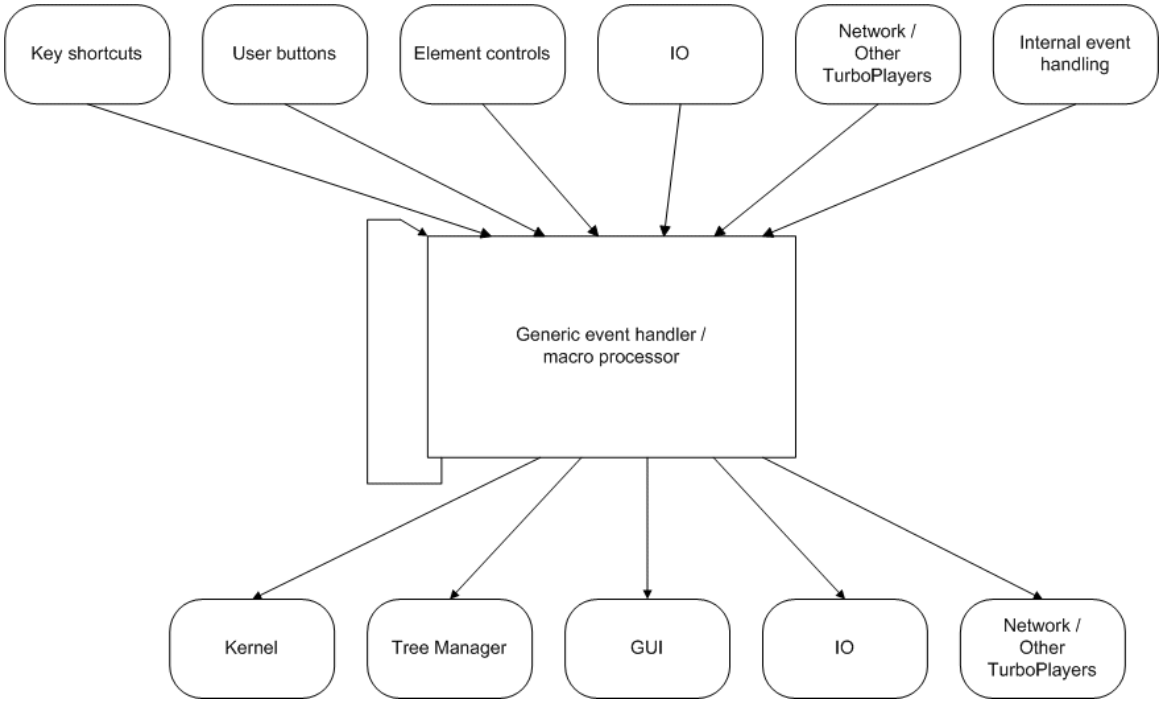
3.4 Other IO Modules

For the modules "Diamond", “DHD”, “EmberPlus”, “GenericSerial”, "GenericMidi" and "MidiMotionMix" there is an extra documentation.

4 EVENTS, SHORTCUTS, CONTROLS AND MACROS

4.1 Overview of event handling

The event concept is a fundamental part of TurboPlayer. Many features are realized with the help of events. Events can be triggered at multiple places, and they can be handled in multiple places. But there is always a central module called generic event handler or macro processor which interprets the event commands and triggers the programmed actions in other modules. The following graph illustrates this fact:



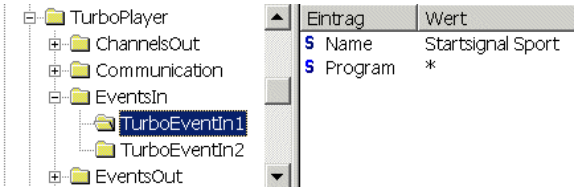
In the top part you can see all "modules" which can trigger generic events, in the middle is the macro processor and at the bottom are modules which execute necessary actions. Each arrow symbolizes a generic event. In the following chapters you will find an explanation of all these expressions.

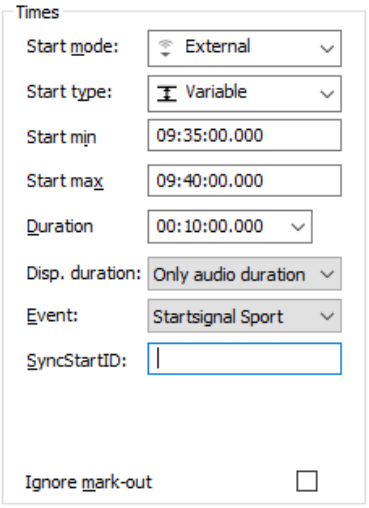
4.2 Event types

There are four different event types: in-events, internal events, generic events and timed events. This differentiation exists mainly for historic reasons, but it is still visible in the administration, because each event type has its own subkey below “TurboPlayer” in the DigaSystem registry.

4.2.1 In-events

In events have a single purpose: they trigger the start of elements with start mode “external”. These events are generated by IO modules only. Example: if an external contact is closed, the GPIO module can be configured to send a specific in-event to the kernel. If there is an element with start mode “external” and corresponding start-event, it is started. Normally (if the element is floating) the element must be the “next element”. For elements with fixed start type there is the registry parameter “ExternalLookAheadTime” which defines the time interval, in which the element can be started by the in-event. And for elements with variable start type the Start min/max fields define the time slot in which an incoming in-event triggers a start. The next two screenshots show the place where you can define the event in the registry and the general submask in DigAIRange where the user can select the in-event:





4.2.2 Special in-events

There is a special in-event which will be triggered by TurboPlayer automatically if a condition is fulfilled. The name of the in-event is "**TP_AllRegioChannelsStopped**". It is triggered when the last element in all regio channels is stopped. Or more in detail: whenever an element is stopped in a channel, which is assigned to a regio-track rundown, TurboPlayer looks through all other channels assigned to these regio rundown lists. If all channels are free, the in-event is triggered. This also works if some of the regionalized elements of an EndpointStory are playing in a remote TurboPlayer. You can use this to continue with the next element on the main track (the show) when the last regionalized element has been played. Therefore, you have to create a key:

TurboPlayer\EventsIn\TP_AllRegioChannelsStopped

in the registry, specify a program/service and a name. Then you can assign an external start mode together with the new in-event for the first element after the regio block / the EndpointStory.

4.2.3 Internal events

Internal events are triggered by TurboPlayer internally, in some cases. For each internal event, you can define one or multiple macros, which will be executed when the corresponding event is triggered. Typical internal events are: TurboPlayer is being reset, an element is starting or: the “next pointer” is set to a new element. This gives you the possibility to perform some custom-specific operations in these cases. All existing in-events are listed below in the chapter about macro programming.

4.2.4 Generic events

Generic events are only text. A text with one or multiple commands or conditions which should be executed by the macro processor. In the overview graph above each arrow symbolizes such a text string being sent. As in every programming language there is a prescribed syntax for these text strings, which is described in detail in the chapter about macro programming.

In the registry generic events can be found in the TurboPlayer subkey “EventsOut”. (This name is used for reasons of backward compatibility only. In TurboPlayer1 these events were always sent from the kernel to IO modules, that’s why they have the “out” in the name.) But only a part of the generic events is defined in this key. Mainly this key is used, to store macros. That means, you create a new subkey, define the text string of the event in the parameter “Command”, specify a user-friendly name in the parameter “Name”. Now you have a new macro, which can either be called from other macros, or you can place a user button into the GUI which triggers this macro or you can assign a key shortcut which triggers the macro. If you want to use the macro within the control list of an element in the rundown, some additional parameters are needed, see below.

Generally spoken you call a macro or a procedure by writing its (key-) name with round brackets behind the name, e.g. “DoSomething()”. Some of the procedures expect parameters, which are written as comma-separated list within the brackets. We distinguish three types of procedures. If the macro processor interprets the name, it must decide, which type it is.

The first type are internal procedures. This are procedures, which are executed by one of the internal TurboPlayer modules. E.g. a start command will be executed by the rundown kernel. Most internal functions begin with a “TP_” in the name, so its quite easy to identify internal procedures. In the overview graph internal procedures are symbolized by the arrows which point from the processor to one of the internal modules: kernel, tree manager or GUI. All existing internal procedures are listed in the chapter about macro programming below. Do not use the prefix "TP_" for your own events/macros, this prefix is reseved for the internal procedures !

The second type are defined macros. That means, if the macro processor does not recognize a name as an internal procedure, it looks through all defined out-events, whether the name matches a key name. If so, it executes the corresponding macro in the “Command” parameter. This feature is symbolized in the overview graph by the arrow, which forms a loop on the left side of the macro processor. Hint: this allows to construct recursive procedure calls, but the processor prevents a recursion depth higher than 32. By the way: there is also a protection against endless loops: If a macro needs more than 1 second for its execution it is terminated. If you need a longer maximum execution time you can increase the value with the registry parameter TurboPlayer\MaxMacroExecutionTime, but you should think many times about doing this, because as long as a macro is running, the kernel is blocked and cannot do anything else.

If the processor does not find a key name that matches the procedure name, it assumes the command will be executed by an IO module. This is the third type of procedures, so to speak the “real out events”. The command will be sent to all IO modules, and you must have configured at least one of them to do something when it receives the command string. Some IO modules can handle commands without the need to configure them explicitly (e.g. DHD_SetLogic command). Therefore, you need to check the documentation of the IO modules. This event type is symbolized by the arrow which points from the processor to “IO”.

Possibly you missed an arrow: the one which points from the processor to “Network / other TurboPlayers”. There is a nice feature: if you write the special internal command “TransferExecution (Computername)” as first line of a macro, the macro will be executed on the specified computer (if there is a TurboPlayer running). This feature allows a handing over between two TurboPlayers. For example, a group is running on TurboPlayer1 but the next group should run in TurboPlayer2. Then you could program a start macro which is executed at the end of the last element of the first group, and which starts the first element in the second group on TurboPlayer2. Be warned that this feature is not highly reliable. The macro will only be sent once to the receiving computer. If the TurboPlayer at the remote computer is not ready to execute the macro, of if the network connection is interrupted for some time, the command might be lost.

4.2.5 Special generic events

At the moment there is one special event name which must not be used as generic event, because it will be treated differently than an event. If you create a fully specified event with the registry key (!) "**TP_Record**" it can be used as record flag for an element or a group. If set, TurboPlayer will try to record the playout of this element/group. This is done by sending ControlRecording commands to the IO modules. The "TIOMultiCoder" IO module can handle these commands and can control a MultiCoder via SOAP interface. Multiple instances of MultiCoder can be controlled. They are distinguished by the coder/recorder name. In the settings of the TIOMultiCoder.dll you can configure a list of controllable MultiCoders. Each entry gets a coder/recorder name, an IP address and an IP port. The names can be used either as first parameter of the TP_ControlRecording command (which directly sends control commands to the recorders) or with the special TP_Record event by appending the name like this:

"**TP_Record_[RecorderName]**", e.g. "TP_Record_Coder2"

If no coder name is specified, the coder with the internal name "<Default>" is being used. This allows to define a single coder in the dll configuration and to use the event "TP_Record" without any coder/recorder name. One additional hint: the coder name is not used as parameter of the TP_Record event (which would be possible) but instead it is appended to the event name. This makes it possible to use the event as a fully specified event which is then available in the context menu of the rundown list.

Since version 6.0 of TurboPlayer a new IO module is available: TIOROAD.DLL. It allows to control an instance of ROAD instead of MultiCoder (which is deprecated). In this case the name of a recorder is configured in the TIOROAD settings dialog and there you can assign a recorder name to a recording template of ROAD, which will be instantiated as a new job when the recording is started.

An element/group being recorded is recorded form the beginning to the end. If multiple consecutive elements/groups have the flag set, only a single record will be created. If you configure MultiCoder/ROAD to put the recording into a database, the new entry will get some metadata of the first element/group with the record flag. Not only the title and some other fields but also the fields "service" and the "show" will be set. The info field will contain a generated description about the TurboPlayer which initiated the recording. Note that the two BUS tasks: AutoInsert and AutoReplace can be used to copy such recordings into the BroadcastSystem.

Attention: there is also a macro command "TP_ControlRecording". Do not mix both names ! TP_Record is only used for registry key names below EventsOut and allows to set a record-flag for an element by the user. TP_Record does not need a command value and must never occur within a macro. On the other side, TP_ControlRecording can only be used within a macro to control the recordings programmatically.

4.2.6 Timed Events

Timed events are defined in the registry in the key "TurboPlayer\EventsTimed". As for generic events there is a "Command" parameter with a macro which is executed when the event is triggered. As the name says, these events are triggered by a timer.

At the moment two timer types exist. First you can specify a "Period" parameter with a period in millisecond. Such a timer will trigger the event periodically. The second timer type is selected if you specify the parameter "HourPoints". There you can specify a comma-separated list of times (only minutes:seconds, without hour specification). The timer will trigger its corresponding event whenever such a point is reached, which is the case once per hour.

4.3 Event sources

Some sources for generic events have been discussed in the previous chapter: the internal events and the execution transfer via network. Other sources are described now.

4.3.1 Key shortcuts

Key shortcuts are listed below the TurboPlayer subkey “Keystrokes”. For each combination of keys there can be a value. The value name is a description of the key combination pressed, e.g. “Shift-Ctrl+112”. This is the key code with additional modifier keys. The value data is the command string to be executed. In most cases it is an internal command like “TP_StartStop (Channel,1)” but it can be the name of an event-out too, or even a whole macro. The latter is not recommended. Instead of writing a macro within the data of a shortcut, you should create a new event-out in the registry and assign a shortcut to that event-out. In the main settings dialog there is a tab to edit the key shortcuts (see chapter about GUI administration), it is not necessary to add values manually.

If TurboPlayer receives a message from Windows that a key combination has been pressed, it looks through all defined keystrokes to find a matching value. If such a value exists, the corresponding function or macro is executed – if it is known.

4.3.2 User buttons

User buttons are a type of GUI window you can put onto the surface as you like. Each user button can trigger an event-out. In the settings dialog you can choose one of the events-out you have defined in the registry, but it is not possible to assign a command string directly. If a user button is clicked, the corresponding event is executed.

Normally, a user button is stateless, of course. Each time it is clicked, it simply triggers the specified event. Therefore, it cannot reflect any changes which take place when the macro is executed. If you want to make a state change visible in the button, you have to use a special internal function: “TP_ ChangeUserButtonProp”. It changes some of the button properties, like the displayed text or the colors. As an example, let’s assume, you want to create a button, which opens or closes a microphone fader. If the fader is open, the button should be red. This can be achieved by writing a macro like this: if the fader is closed, open the fader and make the button background red, otherwise close the fader and make the button background grey. See the examples in the “Macro programming” chapter for a listing of this macro.

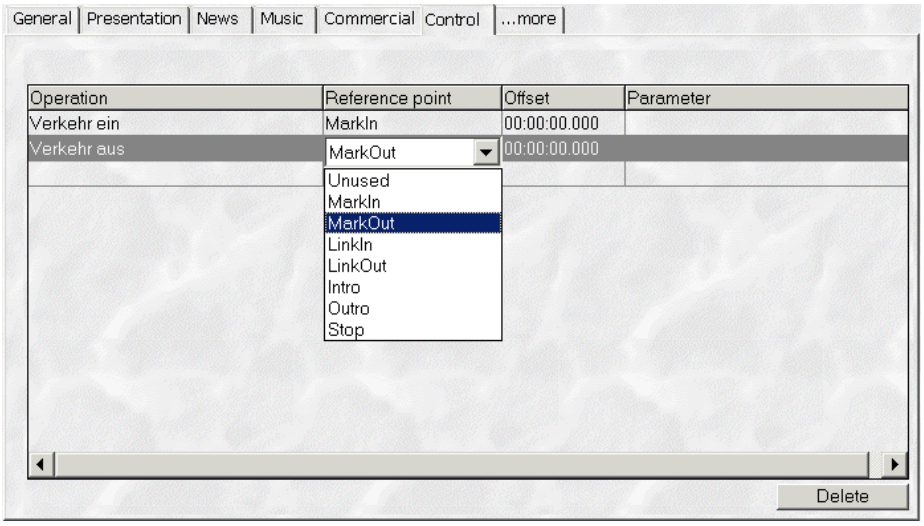
4.3.3 Subclips

Since version 5.5 of TurboPlayer subclips can be the source of a macro command. If the EmbeddedStart mode is active, a subclip with an EmbeddedStartID triggers the special macro command TP_StartEmbedded. This is intended to play live elements in a metadata track during the playout of a live recording. This workflow is described in BCSTechManual chapter 10.5. “Reusing broadcast content / Metadata track”.

4.3.4 Controls

A control is an element media type within DigaSystem. This media type indicates elements which do not have any audio, video or text assigned, but which trigger a predefined action when they are “started” by the on-air application. Aside from this special media type, you have also the possibility to assign an event list to regular elements. In this case, the events are called controls, too.

As controls are something like “scheduled events” they can be edited with DigAIRange. There is a submask called “Control”:



In the first column you see the event name. In the second column you find the reference point which - in combination with the offset - specifies when the event will be executed while the element is being played. In the “parameter” field you can optionally specify a comma-separated list of parameters. These parameters are submitted to the macro being called and can be queried from this macro by the special local variables %1, %2, %3… Be aware that it might be necessary to put parameters into quotation marks or even in an escape sequence (see below) if the parameter contains special characters or might be misinterpreted by the macro processor as an arithmetic expression. Also note that DigAIRange evaluates the “Program” parameter in the registry that lists for which services the corresponding control is available.

For some controls it might be clear, at which position they are executed, e.g. always at the beginning, meaning: reference point = mark-in, offset = 0. If this position is predefined, you can create so-called “fully specified” events. You enter the three parameters “ReferencePoint”, “Offset” and “Bitmap” in the registry and it is possible to activate or deactivate this event with a context menu in the “Event” column of a show window in DigAIRange. The bitmap parameter is needed, because DigAIRange will display this bitmap in the event column when the event has been activated for an element.

While TurboPlayer is playing an element, the control list of the element is processed. Whenever the time position of an event is reached, the corresponding event is executed. This is also true for control elements. These elements can have a duration greater than zero and they are played like regular elements, though playing means a simple playtime counting. In case an element is stopped before the intended duration has been reached, all remaining events will be executed when the element stops.

Pay attention to one **important** thing: though it is very common to use control elements with duration zero this can make a problem: if the next element in the rundown behind the control element is a sequenced element, the second element cannot be started at the same time as the previous control element. There is always a little delay of about the time you have configured as "OpenDelayActive". This behaviour of TurboPlayer is a result of the design and to avoid it, you must give the control elements which are used in a sequence chain a little virtual duration of the OpenDelayActive time plus some dozen milliseconds. E.g. if the delay time is 100 ms use 120-150ms. This avoids a wrong scheduling which cannot be executed by TurboPlayer.

For a group it is also possible to specify controls, but there are only two valid reference points: the beginning and the end (all possible points can be set, but they are interpreted as beginning or end, e.g. link-out=end). For groups the offset does not mean that the event will be executed before or after the specified reference point. But the offset is used to determined the order in which events are executed. For example you can assure that an event is activated first, before you want to execute it in the second step.

4.3.5 IO

Some IO modules can trigger generic events. A typical example is the GPIO module, which handles GPIs (general purpose inputs), e.g. a closed contact. You can configure the module in a way that a generic event is sent when a specific GPI is triggered. In the module settings you can specify a string, which – like always - can be a command string, though it is recommended to specify one of the events-out and to put the macro definition within the event-out.

4.3.6 Timer

Timers are used only to trigger timed events, see above.

4.4 Control creation

Since Release 2012.1 the programs of the BroadcastSystem can create controls when elements or groups are inserted. the idea is that you can use any metadata field to fill in special code strings. If such a code string is recognized, a corresponding control can be created automatically.

For example a database element might have a title like ".....[TA-ON]", meaning: the traffic announcement should be activated when the element is started. When the element is dragged&dropped into a rundown, the title can be analyzed, if "[TA-ON]" is found it can be removed and instead the corresponding control will be set for the element.

To make this work you need fully specified events. That means: EventsOut with defined ReferencePoint, Offset and Bitmap. A correct Program parameter might be necessary, too. Now you can add the registry parameter "Creator" in the event key. It has the format:

```
field(position)!/:code
```

Here is an explanation of the parts:

field	mandatory	The name of the metadata field in which the search is performed. It must start with a letter and can contain letters, numbers or the chars: . _ - It must be a fieldname as it is used within BroadcastSystem.
position	optional	The position where to look for the code. It can be: B begin of the field E end of the field * anywhere in the field 1..n a number with the exact position, starting with 1 In case no position is specified * is assumed.
!	optional	String comparison must be case sensitive. Without ! it is case insensitive.
/	optional	Remove code if found in field.
:	mandatory	Separator. All the text after it is the code.
code	mandatory	The special code string to look for. Pay attention: every character after the : counts, including spaces. So be careful not to enter any spaces if you do not want to.

Here are examples:

```
Title(E)!/:[TA-ON]
```

```
CUSTOM.CONTROLS(*):TA
```

4.5 Macro programming

4.5.1 Syntax

TurboPlayer supports the execution of simple batch commands (aka macros), which can be defined e.g. in the "Command" parameter of out-events. Here is a description of the language syntax, which follows the C syntax:

Case sensitivity:
In principle, this language is case insensitive. This means you can mix lower and upper letters as you like. Nevertheless, we strongly recommend to write everything as it can be found in the documentation, because the implementation of some individual function might handle things case sensitive.

Statement:
A statement is an executable sub part of a macro, for example a procedure call or a variable assignment. Each statement is either ended by a semicolon ; or a line break. Example:

```
Func1()
```

Comment:
A comment is introduced by the two characters "//". The remaining of the line will be ignored by the macro processor.

```
// This is a comment
```

Escape sequence:
Function parameters or right sides of assignments which contain special characters like "=(){}//... can be enclosed in an escape sequence with the start \{{ and end \}} to be treated as a unit. Example:

```
$var = \{{a"b"c"d"e\}}
```

Block:
A block is delimited by { and }. It is a sub-unit for ifs or loops with own local variables. Example:

```
{
    f1 ( )
    f2 ( )
}
```

Types:
Each literal is either a string or a number. This language has no explicit types, the processor decides itself whether a literal is a string or a number. In most cases both types can be used in the place of each other. If a conversion is necessary and possible, it is done automatically. It is also possible to use the cast functions: string(...), integer(...) and bool(...). The bool cast converts to either number 0 or 1.

Numbers:
Always a decimal integer. Examples:

```
123
-10
```

The cast integer(…) can be used to enforce that an expressions is used as a string. Example:

```
integer("10")
```

Strings:
Any text which is not a number. Can be delimited by quotation marks, but need not. Example:

```
"Text"
```

The cast string(…) can be used to enforce that an expressions is used as a string. Example:

```
string(10)
```

Booleans:
There are no booleans in this language. Within a condition or expressions the number 0 yields false and every other number yields true. An empty string yields false, any other string yields true. The cast bool(...) can be used to ensure that an expression is either 0 or 1. The cast will convert usual strings like "true", "false", "yes", "no", "on", "off" case-insensitive to either 0 or 1.

Global variables:

Variables are either of type integer or string. The type is chosen automatically and can change whenever a new value is assigned. Each variable must have been assigned a value to exist. Afterwards it can be used in expressions. If you try to access a variable which does not exist, macro execution is terminated with an error. You can use the intrinsic function <varexist> to check whether a variable is defined. Global variables are available among multiple macros. Example:

```
$var = "Hello"
```

Local variables:

Local variables are available only in the block they are defined or its sub blocks. Example:

```
%var = "Hello again"
```

Special state variables:

TurboPlayer defines some special variables which are only set sometimes. They appear as additional local variables and represent some internal state. For example, if just an element is being started, the folloing variable can be used to retrieve the channel on which the element is starting:

```
%TheChannel
```

Procedure:

A procedure is a subroutine which performs some action. A procedure call consists of the procedure name and a parameter list in round brackets. Procedure names can be composed of letters, numbers or the underscore. They must not have any other character – especially no spaces - in their names. This is true for all internal procedures which start with a TP_ (see below) but as each defined out-event can be used as a procedure name, this rule applies for event names too. The list of parameters is separated by commas. If there are no parameters, brackets must be used nevertheless. Spaces before and behind procedure names and/or parameters do not matter. If you have a parameter which contains a comma, you must enclose the parameter in an escape sequence:
\{{ ….,…. \}} Examples:

- `Proc1()`
- `Proc2 ( Show, 1000 )`
- `Proc3 ( Start, \{{ Text, comma, or something \}} )`

Hint: A procedure does not return anything and therefore it cannot be used within an expression.

Procedure parameters:

To evaluate parameters which were specified when calling a procedure you can use special local variables:

```
%1, %2, %3, ...
```

Function:

A function is a subroutine which returns a value. Therefore, a function can be used within an expression. You can define your own functions and a lot of internal functions are also available. Example:

```
DataOfChannel ( 3, PlayTime )
```

Defined macro function:

It is possible to define your own macro functions. This can be done on each block (=stack level). Globally defined functions can also be set in the registry key "TurboPlayer\Functions". Neither function parameters, nor the return value have a type declaration. This is given implicitly when the parameter/return value is set with a value. The function parameters are accessible as local variables with the name of the parameter. In the parameter declaration in parentheses a preceding % is optional. A defined macro function **must** return a value, though the return value need not be evaluated by the caller.
Example:

```
function IsNextElementNews ( rundown )
{
    return DataOfElement ( %rundown, "Next", "Class" ) == "News"
}
```

If you want to define macro functions globally in the registry, create subkeys below "TurboPlayer\Functions". Each subkey should have either a value "Command" or "CommandFile". The content of the Command value or the referenced text file must contain only function definitions (like in the example) or comments. You can put multiple function definitions into one value/file. The name behind the statement "function" defines the name of the function, not the registry key. In order to get not confused, it is strongly recommended to make the name of the registry key and the name behind the function statement identical. Alternatively, you can put all defined functions into one file and create only a single registry key, e.g. "AllFunctions".

Defined macro functions are available in TurboPlayer version 6.0 or higher.

Expression:

An expression is a text with nested literals (text strings or integer numbers), variables, functions, operators and round brackets which can be evaluated to a single result. Strings can be enclosed in quotation marks but this is not necessary if there are no spaces within the string. Like in C boolean values are treated like this: true = 1 and false = 0. String operands used as booleans are interpreted like this: not empty = true, empty = false. An expression is used within conditions or as the right side of variable assignments. Example:

```
strlen ( "123" ) * ( %i + 1 )
```

If you have to evaluate strings like “true” or “false” within conditions use the bool(…) cast. Example:

```
if ( bool ( $someString ) )
```

Operators:

The following operators are defined:

==	Equal	Compares two subexpressions, numbers or strings must be identical to yield true. If one of both sides is a string, a string comparison is done.
!=	Not equal	The inverse of the equal operation
~=	Equal no case	Both operands are converted into strings. Yields true if a case-insensitive comparison says that both strings are equal
$=	Contains	Both operands are converted into strings. Yields true if the left string contains the right string. The search is case-insensitive. An empty string on one of both sides always yields 0 / false
<	Smaller	Yields true if the left side is smaller than the right side. For strings a case-sensitive comparison is performed
>	Bigger	Yields true if the left side is bigger than the right side. For strings a case-sensitive comparison is performed
<=	Smaller or equal	Like smaller but both sides can be equal
>=	Bigger or equal	Like bigger but both sides can be equal
!	Not	Inverts the following boolean operand
||	Or	Yields true if either the left or the right side are true
&&	And	Yields true if both the left and the right side are true
+	Addition	Adds both sides. Strings are concatenated
-	Subtraction	Subtracts both sides. Not defined for strings.
*	Multiplication	Multiplies both sides. Not defined for strings.
/	Division	Divides both sides. Not defined for strings.

The operators are evaluated with the following hierarchy:

1	( )	Function evaluation
2	!	
3	* /	
4	+ -	
5	== != ~= $= < <= > >=	
6	|| &&	

Condition:

The expression within the parentheses of an if or while statement. Example:

```
if ( %i > 0 )
```

ATTENTION: some programming languages have the rule that the right part of a && or || operator are not evaluated if the left part of the condition already determines the result of the whole condition. This is not true for this macro language. The macro processor will first evaluate all subexpressions within a condition and only in a second step the result of the applied operators will be computed.

if-statement:

A conditionally executed sub part. Additional "elseif" or "else if" and "else" can follow. Each sub part must be a block in curly braces, direct statements are not allowed. Example:

```
if ( ClassOfNext(Show) $= "Music" )
{
   SwitchLightOn(5)
   DoSomething()
}
else
{
    SwitchLightOff(5)
   DoSomethingElse()
}
```

while-statement:

A loop. The sub part (which must be a block) is executed as long as the condition is fulfilled. Example:

```
while ( %i < 10 )
{
   func ( %i )
   %i = %i + 1
}
```

continue-statement:

Omits the remaining part of the sub part of a while statement and continues with the next loop run (when the condition is fulfilled). Example:

```
while ( %i < 10 )
{
   ...
   if ( isTrue ( ) )
   {
      continue
   }
   ...
}
```

break-statement:

Like the continue statement used within a loop. Ends the execution of the subpart of a loop.

return-statement:

Ends the execution of a sub macro. If used within a procedure, which cannot return a value, do not put a value or an expression behind the return statement.

If used within a defined macro function, a value must be returned back! (The caller can ignore this value.) Therefore, put a value or an expression behind the return statement. Example:

```
return %i * 10
```

Attention: as usual in this language, the end of a line terminates a statement. For the return statement (like for an if-clause) there is an exception if you enclose the value to be returned in brackets.

terminate-statement:

Ends the execution of the current macros. This means: the current sub macro and - if applicable - all surrounding macros.

4.5.2 Intrinsic functions

The macro processor implements already some intrinsic functions which can be used like the TurboPlayer-specific functions (which are listed in the next chapter). The following intrinsic functions are defined:

`abs ( n )`	Returns the absolute value of n
`min ( ... )`	Returns the minimum value of the specified numbers
`max ( ... )`	Returns the maximum value of the specified numbers

`makeguid ( )`	Returns a string with a new GUID (globally unique identifier) created by the operating system. The string does not include curly braces at the beginning or the end. Example: 6052001d-fdf1-4a2e-a1cb-a15903e7bc1e
`pow ( x, e )`	Returns the power: x^e
`random ( a, b )`	Returns a random number in range [a,b] (including a and b). a and b must fullfill the condition: 0 <= a < b
`regex_match (s, r, o)`	Regular expression match. Returns 1 if the string s fully matches the regular expression r. The regular expression must be formulated with the syntax of ECMAScript. o is an optional parameter and can be the string “icase” to perform a case-insensitive matching.
`regex_search (s, r, o)`	Like before, but the string s need not fully match r, it is sufficient if any partial string of s matches r.
`replacechr ( s, f, r )`	Returns the string s but all characters f which are found in the string are replaced by character r
`replacestr ( s, o, r, f )`	Returns s but all occurrences of string o are replaced by string r. F can be optional flags, currently only "icase"
`strcat ( s1, s2, ... )`	Returns the concatenated strings. If one of the parameters is an integer it is automatically treated as a string.
`strchr ( s, c )`	Returns the index (starting with 0) at which the character c can be found first in s, otherwise -1
`strcmp ( s1, s2, o )`	Returns 1 if the strings s1 and s2 are identical, otherwise 0 is returned. o can be “icase” to performs a case-insensitive comparison.
`strlen ( s )`	Returns the sum of characters in the specified string
`strrchr ( s, c )`	Returns the index (starting with 0) at which the character c can be found last in s, otherwise -1
`strsearch ( s, u, o )`	Returns 1 if the substring u appears somewhere in string s, otherwise 0 is returned. o can be “icase” to performs a case-insensitive search.
`strstr ( s, u, o )`	Returns the position within string s (starting with index 0) at which the substring u appears in string s. If the substring does not appear in s -1 is returned. o can be “icase” to performs a case-insensitive search.
`strupper ( s )`	Returns s converted into upper letters
`substr ( s, f, n )`	Returns from string s n characters, starting with character f n can be “left”, which returns the string left of f, excluding position f. n can be “right”, which returns the string right of f, including position f.
`trimstr ( s, f )`	Trims (removes spaces) string s. f is an optional flag, which can be "left", "right" or "both". Default is both.
`varexist ( ... )`	Returns 1 if the specified variables exists, otherwise 0
`string ( n )`	Cast to string: ensures that n is interpreted as a string
`integer ( s, d )`	Cast to integer: ensures that s is interpreted as an integer. The second parameter is optional and defines a default value to be used when s cannot be converted into an integer.
`bool ( x )`	Cast to bool (=integer 0 or 1): ensures that x is interpreted as a bool. The following results are defined (all comparisons are case-insensitive): • 1 for: “true”, “yes”, "ja", “on”, non-empty string, any number != 0 • 0 for: “false”, “no”, "nein", “off”, “0”, empty string, 0

Regular expression:

A regular expression is a way to formulate a text search or text match patter. It is available via the intrinsic functions `regex_match` and `regex_search`. Different idioms of regular expressions exist. Here the type “ECMAScript” is being used. For a description, please check these web pages:

http://www.cplusplus.com/reference/regex/ECMAScript/

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions

4.5.3 TurboPlayer functions

General hints: If you make an error when specifying a function, the behaviour depends on the seriousness. For example, if you specify a not existing rundown, the event execution is stopped, because the event will never execute correctly. In contrast, if you try to retrieve element data of a channel, but there is no element loaded to the channel, only an empty string will be returned, because this is only a temporary problem – the next time the event is being executed, there might be an element in the channel.

Many functions of the kernel accept a parameter “position”. This means the rundown position of the kernel. As the kernel handles only the preloaded elements it is the position within the range of preloaded elements, starting with 1. Example: the first preloaded element has position 1. As the range of preloaded elements floats, it is sometimes difficult to determine the correct position for a function call. Please check the function DataOfElement which can return the kernel rundown position for some input parameters, e.g. for the node ID. Some GUI functions also have a “position” parameter. Pay attention: this is not the kernel position! Please read the documention of these functions carefully.

Some of the functions know special terms like "TheRundown" or "TheElement" (see "special state variables" below). Within these functions these terms can be used directly, meaning NOT as a variable "%TheElement" or as a sub function call "TheElement()". Of course, you can use a variable too, but there is a difference: In case you specify a variable and the variable is not set when the macro is executed the whole macro will be terminated. If you specify the term directly it will be evaluted by the implementation of the function, which will act differently: the function will return an empty string as the result. Another possibility to prevent a macro termination would be to use the function "varexist (%TheElement)" to test if a variable like "%TheElement" is defined before using it.

CurrentMode (Type)

Returns the current state of the specified mode

Parameter:	Type	One of: Activation, AdjustLoudness, AssignOverlaid, AutoLoadJingles, BeatExact, ButtonStart, EmbeddedStart, ExtendedShowScanning, FaderStart, FaderStopsPFL, FreeShowlist, Ghost, IgnoreMarkOut, InsertFillers, OnAir, Pause, PlayFadings, SelectSources, ServerConnection, SyncStart
Return	A string	For the activation mode possible values are: LiveAssist, Automatic, Passive. For the server connection possible values are: Connected, Standalone and Rehearse. For all other modes possible values are: 1 (=on), 0 (=off)
Comment:	See the chapter “Concepts” above for an explanation of the available modes.	
Executed by:	Kernel	
Examples:	CurrentMode (Activation)	

CurrentListMode (Rundown, Type)

Returns the current state of the specified mode of a rundown list

Parameter:	Rundown	The rundown list type, one of: "Show", "Jingles", "Stack1".."Stack15", "TheRundown"
	Type	One of: AllowRearrangement, AutoPrepare, AutoDelete, ClearOnReset, PreloadAroundCursor, Rundown, SingleElement, Stack, Track
Return	A string	For the rundown mode possible values are: Random, Sequenced, MultiSequenced, Once For the stack mode possible values are: Clipboard, Free, Track For track it is the active track number. For all other modes possible values are: 1 (=on), 0 (=off)
Comment:	See the chapter “Concepts/List modes” above for an explanation of the available modes.	
Executed by:	Kernel	
Examples:	CurrentListMode (AutoPrepare, Stack1)	

CurrentDate ( )

Returns the current date as a number

Return	Integer	Internal date format, the year, month and day are written as a single number, e.g. christmas 2009 is: 20091224. To convert such a date into a string use the function ConvertDate.
Executed by:	Kernel	

CurrentTime ( )

Returns the current time as a number

Return	Integer	Milliseconds after midnight. To convert such a time into a string use the function ConvertTime.
Executed by:	Kernel	

ComputerName ( )

Returns the computer name

Return	String	Windows computer name on which the macro is being executed
Executed by:	Kernel	

ConvertDate (Date, Format)

Converts a date between the internal number format and a string (both directions).

Parameter:	Date	Either a number or a string. If a number is given the internal date format is assumed. This is the year, month and day written as a single number, e.g. christmas 2009 is: 20091224. In this case the function returns a string with the formatted date. If a string with a date is given, the function returns a number with the date in internal format.
	Format	An optional format string. You can use the fields YYYY MM DD or separator characters. If no format is specified, the format “YYYY-MM-DD” is assumed for the conversion number->string. For the conversion string-> number the function tries to analyse the used format. Recognized formats are: “YYYY-MM-DD”, “MM/DD/YYYY”, “DD.MM.YYYY”.
Return	Integer/ String	Either a number or a string, see above.
Comment:	If no valid conversion is possible, e.g. because no valid date string is given, the function fails with an error (the macro execution is terminated).	
Executed by:	Kernel	

ConvertTime (Time, Format)

Converts a time between the internal number format and a string (both directions).

Parameter:	Time	Either a number or a string. If a number is given the internal time format is assumed. This is the milliseconds after midnight. In this case the function returns a string with the formatted time. If a string with a time is given, the function returns a number with the time in internal format.
	Format	An optional format string. You can use the fields hh mm ss fff or separator characters. If no format is specified, the format “hh:mm:ss.fff” is assumed for the conversion in both directions.
Return	Integer/ String	Either a number or a string, see above.
Comment:	If no valid conversion is possible, e.g. because no valid time string is given, the function fails with an error (the macro execution is terminated).	
Executed by:	Kernel	

StationName ( )

Returns the station name

Return	String	DigaSystem station name on which the macro is being executed See registry parameter: Digas\Settings\StationName
Executed by:	Kernel	

IsEventActive (Name)

Returns whether the specified generic event is active

Parameter:	Name	The name (=registry key below "EventsOut") of the generic event. String comparison is case insensitive.
Return	An integer	Possible values are: 1 (=active), 0 (=not active)
Executed by:	Kernel	
Examples:	IsEventActive (FollowEvent)	

IsFaderStartActive (Line)

Returns whether fader starts for the specified line or mixer source are active

Parameter:	Line	Either the line number or a known mixer source name
Return	An integer	Possible values are: 1 (=active), 0 (=not active)
Executed by:	Kernel	
Examples:	IsFaderStartActive (Mic1)	

IsTimerActive (Name)

Returns whether the specified periodic timer for a timed event is active

Parameter:	Name	The name (=registry key below "EventsTimed") of the timed event. String comparison is case insensitive.
Return	An integer	Possible values are: 1 (=active), 0 (=not active)
Executed by:	Kernel	
Examples:	IsTimerActive (BlinkEvent)	

NumberOfConnectedGUIs (Options)

Returns the number of currently connected GUIs.

Parameter:	Options	This parameter is optional, you need not define it. If no parameter is given, the function counts only GUIs which are currently connected, a GUI with connection state “Disturbed” is not counted. The options can be a comma-separated list of keywords “Disconnected”, “Disturbed” and “Connected”. In this case the function will count all GUIs with the specified states.
Return	An integer	0, 1, 2, 3 …
Executed by:	Kernel	
Examples:	NumberOfConnectedGUIs (“Connected, Disturbed”)	

IsGUIConnected (GuiNo)

Returns, whether the specified GUI is currently connected.

Parameter:	GuiNo	The GUI number of one of the GUIs connected to the engine.
Return	An integer	1 if the GUI is connected, 0 otherwise.
Comment	A GUI with connection state “Disturbed” does not count as connected.	
Executed by:	Kernel	
Examples:	IsGUIConnected (2)	

DataOfGUI (GuiNo, Name)

Returns status information of of a GUI

Parameter:	GuiNo	The GUI number of one of the GUIs connected to the engine. If a star '*' is given, the data of the GUI with the lowest number is returned.
	Name	The requested data. Currently these names are supported: - "InfoTextEditMode" Is info text editor in edit mode ? (0/1) - "RtfTextEditMode" Is RTF (presentation) editor in edit mode ? (0/1) - "InfoTextHold" Is info text editor in hold mode ? (0/1) - "RtfTextHold" Is RTF (presentation) editor in hold mode ? (0/1) - "MiniDBMTables" The active table in format "DSN\table" in single-table mode, or a comma-separated list in multi-table mode. **Attention:** the name is case sensitive ! If a name is unknown or if the engine was not yet informed by the GUI about a data an empty string is returned. The following two data fields are always present and filled, even if no GUI with the requested number was ever connected: - “ConnState” The connection state of the GUI (as it is seen from engine side, because it is the kernel, which executes the function). Can be one of: * Disconnected * Disturbed * Connected - “PrevConnState” The last connection state before the current connection state was set. This can be used to distinguish e.g. a new connect from a reconnect after a disturbed connection.
Return	A string	The data.
Executed by:	Kernel	
Example:	DataOfGUI (*, InfoTextEditMode)	

DataOfEngine (Name)

Returns information of the engine.

Parameter:	Name	The requested data. Currently these names are supported: - "DigasUserID" The current DigaSystem user ID (this is the short name with max 8 characters), which was used by TurboPlayer for the DigaSQL login. - "DigasUserName" If the currently logged in DigaSytem user has a long name, this name is returned. Otherwise, the data is identical to DigasUserID. - "BCSConnectionState" The current state of the BCS connection (see chapter 1.4.4). One of: * Disconnected * Disturbed * Connected - "BCSConnectionID" The connection ID as it is used by TurboPlayer during the login to BCS. This ID appears also as last column in the client list of the BCS window. Typically, something like <ComputerName>\TurboPlayer is used. - "BCSName" The name of the current BCS, to which TurboPlayer is connected. In this case it is the registry name of a key below key "Digas\PlanServer". This name is always identical for master and slave BCS. - "BCSAddress" The IP address of the current BCS, to which TurboPlayer is connected. In this case it is the value of "Address" from the registry, which was used for connecting to BCS via TCP/IP – if the connection was made to the master. If the current connection is to the Buddy, it would be the "BuddyAddress". - "BCSPort" The port number of the currently connected BCS (the BCS side/listen port). It depends on: master or buddy connected, protocol: SeplProtocol, WebSockets, SecureWebSockets. 0 if there is no active connection. - "ProgramName" The name of the currently selected program (service). - "ProgramID" The full node ID of the currently selected program (service). - "ComputerName" The local computer name (NetBIOS). Same as function ComputerName(). - "StationName" The DigaSystem station name as it was read from registry value Digas\Settings\StationName. Same as function StationName(). - "StudioName" The studio name as it was read from registry value TurboPlayer\StudioName. - "IsStandaloneEngine" 1 if the macro is processed within a TurboPlayerEngine (without GUI), 0 if the macro is processed within a TurboPlayer_....exe with integrated GUI.
Return	A string or a number	The data. Only for the BCSPort it is a number, in all other cases a string.
Comments:	This function is available since version 6.0 of TurboPlayer. Pay attention when using some of the fields in disconnected / rehearse server connection mode. In this case the BCSCurrentName will be empty, BCSCurrentPort will be 0 and BCSConnectionState will be "Disconnected. Other fields will have the old content, they had before disconnecting from BCS. This is different if you really disconnect from BCS (with a macro command TP_ConnectToBCS("")). Then all BCS- and program-related fields will be empty (the port will be 0).	
Executed by:	Kernel	
Example:	DataOfEngine (BCSAddress)	

DataOfRundown (Type, Field)

Returns data of the rundown lists

Parameter:	Type	The rundown list type, one of: Show, Jingles, Stack1..Stack15 or the special term "TheRundown".
	Field	The requested field. Currently these fields are supported: - "NameOfActiveTree" returns the name of the current show or jingle group. - “NameOfActiveTrack” returns the track name (if set) of the current show. - “IdOfActiveTree” returns the full node ID of the current show or jingle group. - “InfoOfActiveTree” returns the field “Info” of the current track (for shows) or jingle group or pool. - “InfoOfActiveShow” returns the field “Info” of the current show - for loaded tracks. For other loaded node types this field is empty. - “StartDateOfActiveShow” and “StartTimeOfActiveShow” return the start date as a number YYYYMMDD or the start time in milliseconds after midnight of the current show (if a track is the loaded node, otherwise 0). - “StopDateOfActiveShow” and “StopTimeOfActiveShow” return the stop date as a number YYYYMMDD or the stop time in milliseconds after midnight of the current show (if a track is the loaded node, otherwise 0). - "Next" returns the index number of the <next element> pointer. This index is only the index within the elements known to the kernel (aka the preloaded elements) and starts with 1 for the topmost element in this list. If no next element is defined the function returns 0. - “IdOfNext” returns the full node ID of the <next element>. - “IdOfCursor<n>” returns the full node ID of the element onto which the GUI <n> has set its cursor. <n> must be the number of a known GUI, starting with 1. “IdOfActiveGroup” returns the full node ID of the active group. A group becomes active if the first element is started within the group. There is only one active group. If the last started element is outside of a group this ID will be empty. - "LastPlaying" returns the index number of the last playing element. If no really playing element is found, the function returns the index of the last element which is in its start process. I no playing or starting element is found the function returns 0. - "IdOfLastPlaying" like "LastPlaying" but returns the full node ID of the last playing element as a string or an empty string if no such element exists. - "KernelElements" returns the number of elements which are currently known to the kernel. This is a maximum of <preloaded elements> for this rundown and does not include groups or other elements within a rundown. - “LastPlaytimeMsgElapsed” returns the time in milliseconds since the last playtime or remaintime message from MultiPlayer was received for the specified rundown. This information can be used to monitor the playout and to trigger a start if nothing is playing for a too long time. If no element has been started for the rundown at all, the special number 999999999 (9 times a 9) is returned. - “TrackNumber” returns the current track number loaded for the specified rundown. If a rundown does not have a show track loaded (e.g. for jingles) -1 is returned. - “PlayingEndpointStory” returns 0 or 1. 1 is returned if the rundown (typically only “Show”) is just playing an EndpointStory. The value switches back to 0 if all elements in the EndpointStory have ended. See also the internal events “EndpointStoryBegin” and “EndpointStoryEnd”.
Return	A string	The data.
Comment:	If a rundown contains shows this includes typically multiple shows (depending on parameter ShowInterval). The identifiers “ActiveTree” or “ActiveShow” above mean: the track/show which is currently loaded to the position 0 of the show interval. This is also the show which is displayed in the GUI in the show selection box. TurboPlayer tries to keep the show which has playing elements at this position but if playout is near show borders switching to the next show will typically be delayed until the situation is clear and a background routine has performed the switch to the next show.	
Executed by:	Kernel	
Example:	DataOfRundown (Jingles, Name)	

DataOfShow (Type, ShowSpec, Value, Field)

Returns show metadata.

Parameter:	Type	The rundown list type, one of: "Show", "Jingles", "Stack1".."Stack15" or “TheRundown”.
	ShowSpec	This parameter together with Value specifies how the show node is identified. This is necessary, because a rundown can load multiple shows and it must be clear, from which of them the metadata should be retrieved. The following strings are possible: ActiveShow: Value = unused/empty, the active show is used. ID: Value = ID of show or any node within the show (e.g. of an element). Time: Value = any timestamp within the requested show, e.g. start of show. Name: Value = Name of show. Cursor: Value = Cursor (GUI) number. Next: Value = unused/empty, the <next> pointer of the rundown is used. Element: Value = element position within kernel (e.g. from %TheElement).
	Value	Depends on the parameter ShowSpec. Details: The specified ID can be the ID of any subnode with the show of interest, e.g. of a track, a group, a story, or an element. It can be the ID of the show itself, too. Pay attention to always put IDs you write directly into macros in quotation marks, because otherwise you write a numerical division. A timestamp must be given as a string as it is stored in BCS metadata. This is the format: “YYYY-MM-DD HH:MM:SS.mmm”. (Milliseconds and even seconds can be omitted.) For cursor the GUI number need to be specified. If this GUI has set a cursor onto an element, the show for this element is picked. If the GUI is not connected at the moment, an empty string is returned. The element position is the number of the element within the kernel rundown starting with 1. Attention: the kernel handles only the elements within the preload range. Therefore position 1 means: the first preloaded element. Using the position is mainly done if you use a state variable like %TheElement, which returns the element position for the current call.
	Field	The requested field. All metadata fields of a show node can be queried, including custom or private-data (PD_) fields.
Return	A string	The data.
Comment	ATTENTION: This function can only be called if the registry parameter “TurboPlayer\KernelHasFullData” is active. If the syntax of the function call is correct, but no element can be found, an empty string is returned (instead of an error result). See also the general hint above (top of this chapter). You can only retrieve show information, if a show is loaded to the rundown. For other nodes, like a jingle group, an empty string is returned. ActiveShow means: the track/show which is currently loaded to the position 0 of the show interval. This is also the show which is displayed in the GUI in the show selection box.	
Executed by:	Kernel	
Example:	DataOfShow (Show, Next, "", "Team_Speaker") DataOfShow (Show, ID, "00000000/00000067/00000004/00000069/00000067/0000006B", "Name") DataOfShow (Show, Time, "2019-12-24 18:00", "JingleGroupName")	

DataOfElement (Type, Value, Field)

Returns element data

Parameter:	Type	The rundown list type, one of: "Show", "Jingles", "Stack1".."Stack15", "TheRundown" or: "Line", "Channel". If you use Line or Channel there must be exactly one playing/paused element, otherwise an empty string will be returned.
	Value	If Type is "Line" or "Channel" it is the line / channel number. If Type is a rundown, it is a full node ID or the kernel rundown position as number (starting with 1) or one of the keywords: "Next", "Cursor<n>", "LastPlaying", "TheElement", "GroupOfNext", "StoryOfNext", "GroupOfCursor<n>", "StoryOfCursor<n>", "GroupOfLastPlaying", "StoryOfLastPlaying", "TheGroupOfTheElement", "TheStoryOfTheElement", “EndpointStoryOfNext”, “EndpointStoryOfCursor<n>”, “EndpointStoryOfLastPlaying”, “TheEndpointStoryOfTheElement”. If you use one of the cursor keywords, <n> denotes the GUI number (starting with 1).
	Field	The requested field. Notes: the field name is not always identical to the XML tagname because there are some different or additional fields which can be evaluated. Normally not all XML tagnames can be queried, because the RundownKernel holds only a small subset. These fields can be queried: ID, Position, Class, MediaType, Title, Source, FileState, SendState, Time_StartType, Time_StartMode, Time_OnAirStartType, Time_OnAirStartMode, StartDate, StartTime, StopDate, StopTime, Time_Duration, LoadState, PlayState, PlayTime, RemainTime, Fade_MarkIn, Fade_MarkOut, Fade_FadeIn, Fade_FadeOut, Fade_LinkIn, Fade_LinkOut, NodeType, Ressort, Reserve, MixerSource, MixerSourceKey, MixerSourceName, IsGroupMember, IsStoryMember, IsEndpointStoryMember, If the global flag <KernelHasFullData> is set, all possible XML tags can be queried, and in addition fields of not preloaded elements and of groups and stories and the special flags "IsGroup", "IsStory", “IsEndpointStory” and the "IDOfGroup", "IDOfStory", “IDOfEndpointStory”.
Return	A string	The data. Hints: The position is the number of the element within the kernel rundown starting with 1. Attention: the kernel handles only the elements within the preload range. Therefore position 1 means: the first preloaded element. The evaluation of the position of an element is mainly useful if you need it as parameter for one of the other macro functions or commands. The Date, time and duration fields are given as a number in the internal format, that is e.g. 20051224 for a date, the time is milliseconds after midnight, durations like PlayTime or RemainTime are in milliseconds. IsGroup, IsGroupMember, IsStory, IsStoryMember, IsEndpointStory and IsEndpointStoryMember are flags (0,1). For LoadState the possible values are: “Unloaded“, “Loaded”, “Load failed”.
Comment	If the syntax of the condition is correct, but no element can be found, an empty string is returned (instead of an error result). See also the general hint above (top of this chapter). The specifications "LastPlaying", "GroupOfLastPlaying" and "StoryOfLastPlaying" look for really playing elements first. If no such element is found, the last element which is just starting is returned. This might be important if the function is used within an internal event of type "ElementStarting". When evaluating the LoadState be aware that an element is normally moved to the rundown list of type “Deleted” before the element is really unloaded by a background task.	
Executed by:	Kernel	
Example:	DataOfElement (Show, Next, Class) DataOfElement (Jingles, „00000000/00000067/00000004/00000069/00000067“, Position)	

ControlOfElement (Type, Value, Query, Key, Reference, Offset)

Special version of <DataOfElement> for controls.

Parameter:	Type	The rundown list type, one of: "Show", "Jingles", "Stack1".."Stack15", "TheRundown" or: "Line", "Channel". If you use Line or Channel there must be exactly one playing/paused element, otherwise an empty string will be returned.
	Value	If Type is "Line" or "Channel" it is the line / channel number. If Type is a rundown, it is the full node ID or the kernel rundown position as number (starting with 1) or one of the keywords: "Next", "Cursor<n>", "LastPlaying", "TheElement", "GroupOfNext", "StoryOfNext", "GroupOfCursor<n>", "StoryOfCursor<n>", "GroupOfLastPlaying", "StoryOfLastPlaying", "TheGroupOfTheElement" or "TheStoryOfTheElement", “EndpointStoryOfNext”, “EndpointStoryOfCursor<n>”, “EndpointStoryOfLastPlaying”, “TheEndpointStoryOfTheElement”. If you use one of the cursor keywords, <n> denotes the GUI number (starting with 1).
	Query	The requested information. These values can be queried: – "Set" or "IsSet": The return value is 1 if the control is set, otherwise 0. – "Key" or "Operation": The registry key of the corresponding event – "Reference" or "ReferencePoint": The reference point, one of: MarkIn, MarkOut, Stop, LinkIn, LinkOut, Intro, Outro – "Offset": The offset from the reference point in milliseconds
	Key	The registry key (=operation) of the queried control. Can be a star "*" to query any control at the specified position (in this case the next two values must be specified).
	Reference	Optional the reference point of the queried control.
	Offset	Optional the offset of the queried control.
Return	A string	The data (see <Query> for the possible results).
Comment		If the syntax is correct but no control is found for the specified key / position, an empty string is returned. See also the general hint above (top of this chapter). If there are multiple controls which are found for the specified key / position, it is undefined, for which of them data is returned. The specifications "LastPlaying", "GroupOfLastPlaying" and "StoryOfLastPlaying" look for really playing elements first. If no such element is found, the last element which are just starting is returned. This might be important if the function is used within an internal event of type "ElementStarting". If the global parameter "KernelHasFullData" is not set, the kernel can handle only elements which are known to the kernel, which are only the preloaded elements ! If the parameter is set, other elements and groups can be handled, too. This also means: the group-keywords can only be used if the parameter is set.
Executed by:		Kernel
Examples:		ControlOfElement (Show, Next, IsSet, ARIOn) ControlOfElement (Jingles, “00000000/00000067/00000004/00000069/00000067”, IsSet, ARIOn) ControlOfElement (Show, Cursor1, IsSet, *, MarkOut, -5000)

DataOfLine (Line, Field)

Returns line data

Parameter:	Line	The line number starting with 1. For external lines you can specify the mixer source name instead. In internal events you can use the special term "TheLine".
	Field	The requested field. These fields can be queried: LineMode, CurrentLineMode, AutoLineMode, OpenState, MixerSourceName, MixerSourceKey, SignalProcessing, GainLevel, LastFaderOpByEngine, PrelistenState, PFLState
Return	A string	The data. Line mode can be: None, Passive, Active, Active/Variable, Active/OnOff. LineMode is the setting for activation Passive+LiveAssist, AutoLineMode is the setting for activation Automatic, CurrentLineMode returns the setting depending on the current activation. OpenState can be: Closed, Open pending, Open, Close pending. LastFaderOpByEngine, PrelistenState can be 0 or 1. SignalProcessing is one of the key names below the registry key "TurboPlayer\ChannelsOut". GainLevel is the dB value for the gain. PFLState lists all the active PFL numbers as a comma separated string, or an empty string if no PFL is active at all.
Executed by:	Kernel	
Comment:	In case you specify a mixer source as first parameter and this source is not selected, execution does not stop with an error, an empty string is returned instead. If you use the specifications "TheLine" for the first parameter and no active/current line is known, the function returns an empty string.	
Example:	DataOfLine (1, OpenState)	

DataOfChannel (Channel, Field)

Returns channel data

Parameter:	Channel	The channel number starting with 1. In internal events you can use the special term "TheChannel".
	Field	The requested field. These fields can be queried: PlayState, PlayTime, RemainTime, PrelistenState (only for prelisten channels). “LastPlaytimeMsgElapsed” returns the time in milliseconds since the last playtime or remaintime message from MultiPlayer was received for the specified channel. This information can be used to monitor the playout and to trigger a start if nothing is playing for a too long time. If nothing has been started for the channel at all, the special number 999999999 (9 times a 9) is returned. “LastPlaytimeMsgDate”, “LastPlaytimeMsgTime”: like before but the date / time as a number for the last received playtime message. You can use the date / time conversion functions to convert the numbers to a string. If nothing has been started for the channel at all, the special number -1 is returned.
Return	A string	The data. PlayState can be Free, Loaded, Playing, Paused. PrelistenState can be 1 (=active), 0 (=not active)

Alternative usage: (available since TurboPlayer 6.0)

Parameter:	Channel	Special keyword "CountInState". TurboPlayer will run in a loop over all known channels and will count all channels, which are in the specified states (=next 1-4 parameters). This count is then returned as number.
	Field 1 - 4	You can specify 1 - 4 parameters. Each can have one of the following keywords in order to count the corresponding state: • Free • Loaded (this is the "element is prepared in a channel" state) • Playing • Paused
Executed by:	Kernel	
Comment:	If you use the specifications "TheChannel" for the first parameter and no active/current channel is known, the function returns an empty string.	

Example: DataOfChannel (3, PlayTime)
DataOfChannel (CountInState, Loaded, Playing, Paused)

DataOfRecording (Recorder, Field)

Returns data of a live / show recording

Parameter:	Recorder	The name of the recorder as it is configured in the IO modules for recordings (TIOMultiCoder, TIOROAD) and as it is used in calls to the command TP_ControlRecording.
	Field	The requested field. These fields can be queried: • IsRecorderKnown: Returns 1 if the recorder name is known, otherwise 0 is returned. A recorder name is known if at least one control request (e.g. a start recording) was sent to the recorders or if at least one status or progress report was received from the recorder. • RecordState: the last state as it was reported from the recorder. So far only ROAD can report the state but not MultiCoder. The possible values are: Unknown, Recording, Pause, Stopped, Error. • RecordedTime: The last recorded time as it was reported from the recorder. So far only ROAD can report the progress but not MultiCoder. The value is a number in milliseconds. • RequestCounter: When TurboPlayer controls a recorder, a reference counter is used in order to really start a recording only with the first request and only stop a recording with the last request. Additional start or stop requests only produce a break / set marker. The reference counter is returned as number.
Return	A string or a number	See above in the "Field" description.
Executed by:	Kernel	
Comment:	This command returns data only for live/show recordings via a recorder (MultiCoder, ROAD). It cannot be used to retrieve data of a GUI recording. In case you specify a recorder name as first parameter and this name is not known, execution does not stop with an error, an empty string or "Unknown" or number 0 is returned instead.	
Example:	DataOfRecording ("Recorder1", RecordState)	

SubMixerSource (Source)

Returns the currently selected sub mixer source name

Parameter:	Source	A mixer source name (key of the registry)
Return	A string	The currently selected sub mixer source name. The registry key is returned, not the content of the parameter "MixerSource".
Executed by:	Kernel	
Examples:	SubMixerSource (MainMic)	

GetVariable (Name)

Returns the value of the specified global variable. If the variable is unknown, the function fails.

Parameter:	Name	The variable name (name comparison is case insensitive)
Return	The variable value. Depending on the variable type it is either an integer number or a string. See also the function TP_SetVariable.	
Comment:	This function exists for backward compatibility with old macros only. Now you can query the content of variables directly by using their names, like: $SpecialFlag. Local variables cannot be queried with this old function.	
Executed by:	Kernel	
Examples:	GetVariable (SpecialFlag)	

4.5.4 Special values which can be queried

Some of the functions above return values which are internal to the TurboPlayer and are not documented in the BCS tech manual:

PlayState

The PlayState for an element is like the SendState, but it represents the internal state of an element/channel. The SendState in contrast is the value stored in the XML data in the BCS and can be delayed against the PlayState. Besides there are more possible values for the PlayState. It can be:

"Stopped", "Stop pending", "Break pending", "Play timed", "Play pending", "Playing",
"Pause pending", "Paused"

In addition you can query the PlayState of a channel. Possible values are:
"Free", "Loaded", "Playing", "Paused"

4.5.5 TurboPlayer procedures

Executed by either GUI or Engine

General remark

All the commands, which can executed by either a GUI or the engine have a first parameter, which is the GUI number. If a GUI should execute the command, then use a valid GUI number starting with 1. In some cases it can make sense that all connected GUIs should execute a command. In this case you can use a star "*" for the first parameter. If the engine should execute a command, then use the word "Engine" as first parameter. Hint: any word, starting with an 'E' or 'e' works, or only this letter alone. You can use the number 0, too, but as this is not as clear as "Engine", it is recommended to not use this special number 0.

TP_Close (GuiNo, Confirmation)

Closes Turbo Player interface (equivalent behaviour of using Alt+F4 windows shortcut)

Parameters:

GuiNo	GuiNo of the Gui which will execute the command. "Engine" to make the engine execute the command.
Confirmation	This parameter is only available, if a GUI number is specified as first parameter, not for the engine. Defines whether the user has to confirm termination with a dialog box. If set to "Default" or if empty the corresponding parameter from the main settings dialog (tab Dialogs & Menus) is applied. If set to "Silent" the confirmation dialog box is supressed.

Comment: This command need not be defined to close TurboPlayer via ALT-F4 key shortcut. Normally this is handled by Windows correctly. But in case you encounter problems, you can assign the Alt-F4 shortcut to the TP_Close command in order to circumvent the Windows function.
The engine can be closed by using the command TP_Close (Engine). This works when the engine is running as a desktop program (in sys-tray) or as a service. But you cannot use the engine command, if the process is a full TurboPlayer with integrated GUI.
You can use the command TP_Close() without any parameters if it is possible that the process is a either a standalone engine or a process with integrated GUI. In case you have these both possibilities and you want to specify the "Silent" parameter, then you must first determine, whether the process is a standalone engine with the function DataOfEngine (IsStandaloneEngine) and then either execute the engine command or the GUI command with the Silent parameter.

Examples: TP_Close (1)

TP_Dump (GuiNo, Parts)

If executed by a GUI, it opens the dump internal state dialog. If executed by the engine, the dump of the internal states is directly being executed without displaying a dialog first.

Parameters:

GuiNo	GuiNo of the Gui which will execute the command. "Engine" to make the engine execute the command.
Parts	This second parameter is only available if the command is executed by the engine. Then you can use the parameter to specify the parts, which should be dumped (what you otherwise select in the dialog shown by the GUI). These parts are defined: - TreeManagerState - TreeManagerTree - TreeManagerList - RundownKernelState - RundownKernelList Specify these parts in one string and separate the parts with a pipe '|'. If the parameter is missing, all parts are dumped.

Examples: TP_Dump (1)
TP_Dump (Engine, "TreeManagerState|RundownKernelState")

Comment: The old documentation mentioned the command "TP_DumpState()" without parameters for executing the dump by the engine directly. This still works.

TP_UpdateStarttimes (GuiNo, RundownList)

Updates all start-/stoptimes of the elements in the rundown

Parameter:

GuiNo	GuiNo of the Gui which will execute the command. This can be an arbitrary GUI number (an existing one, of course), because in reality the command is being executed by the engine. Alternatively, you can directly specify “Engine” for the first parameter, then the command will be executed completely by the engine.
RundownList	Show, Stack<1..15>

Comment: This function is intended as solution for the request that the user wants to perform a time calculation for a rundown, though no element is running. TurboPlayer enters the current time as faked real-start-time for the <next> element, as if the element was just started "now". For the elements below the BCS will do a normal time calculation.

Attention: the function is only a workaround and has 2 disadvantages:

- Each time the function is called a BCS time calculation and update notifications to all clients are generated. This can cause a very high computer or network load, especially if this function is triggered regularly by a timed event.
- The <next> element has faked real-times. This data might be a problem when the element is being used somewhere else – meaning: copied or moved to a different place.

Since version 6.0 of TurboPlayer this command can be used for pausing elements, too. If there are only pausing elements (and no running ones!) TurboPlayer will adjust the expected real stop times/durations of these elements so that BCS can compute new times for all following elements in the rundown.

Examples: TP_UpdateStarttimes (Engine, Show)

Executed by the GUI

TP_ChangeWndState (GuiNo, Action, Sub wnd type)

Docks / undocks a sub window of specified type.

Parameter:	GuiNo	GuiNo of the Gui which will execute the command or * for all Guis
	Action	Dock, Undock, ToggleDock, Hide, Show, ToggleShow, ToggleMinimizeOnCaptionbar
	Sub wnd type	CFM, OTM, Infotxt, Modtxt, DigaMessage, Story, MiniDBM
Comment:	'ToggleMinimizeOnCaptionbar' only works in undocked state.	
Example:	TP_ChangeWndState (1, Undock, OTM)	

TP_ChangeUserButtonProp (GuiNo, UserButtonID, Property, Value)

Changes attributes of a UserButton, StackButton or ImportButton (e.g. text color, background color, etc)

Parameter:	GuiNo	GuiNo of the Gui which will execute the command or * for all Guis
	UserButtonID	String identifier of the button; adjustable in the UserButton settings (dialog)
	Property	One of these: "ColorBkgnd", "ColorText", "Text", "Bitmap"
	Value	R,G,B for colors; otherwise the value as string; the filename if bitmap
Examples:	TP_ChangeUserButtonProp (1, UB#1, ColorBkgnd, 255,255,255) TP_ChangeUserButtonProp (*, UB#2, Text, UserButtonTitle)	

TP_DeleteAll (GuiNo, Rundown)

Deletes all items in the rundown if rundown is in clipboard mode!

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	Rundown	Show, Stack<1..15> or * for the selected rundown list; rundown must be in clipboard mode! Otherwise DeleteAll is denied!
Comments	You can use the engine command TP_Delete, too.	
Examples:	TP_DeleteAll (1, Show); TP_DeleteAll (1, *)	

TP_DeleteSelected (GuiNo, Rundown, WindowNo, Confirmation)

Deletes the selected item in the rundown

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	Rundown	Show, Jingles, Stack<1..15> or * for the selected rundown list
	WindowNo	The window number <1..n> (deletes the cursor item in this window of specified rundown list type)
	Confirmation	Optional parameter. Can be TRUE, Yes, or 1 to request a confirmation with a message box from the user, FALSE, No or 0 to suppress such a confirmation dialog, or Default or the parameter can be left out for the standard behaviour as it is configured in the main settings.
Comments	You can use the engine command TP_Delete, too.	
Examples:	TP_DeleteSelected (1, Show, 1); TP_DeleteSelected (1, *, 1, FALSE)	

TP_ExecuteStackButton (GuiNo, <StackButton-ID>)

Performs the stack button action as if the button would mouse-clicked.

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	StackButton-ID	ID of the stack button, see button settings
Examples:	TP_ExecuteStackButton (1, #StackButton1)	

TP_Filter (GuiNo, FilterButtonTitle, Action)

"Presses" a filter button on the surface of a GUI

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	FilterButtonTitle	The Title as configured in the filter button settings (button search is case-insensitive)
	Action	Can be Toggle, On, Off, TRUE, FALSE, Yes, No, 1, 0
Examples:	TP_Filter (1, "TextElements", Toggle)	

TP_Flip (GuiNo, Type, WindowNo)

Swaps the NEXT item and the NEXT + 1 item in the rundown

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	Type	Show, Stack<1..15> or * for the selected rundown list
	WindowNo	The window number <1..n> of the specified rundown list
Examples:	TP_Flip (1, Show)	

TP_Import (GuiNo, ImportButtonID)

"Presses" an import button on the surface of a GUI

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	ImportButtonID	The ID as configured in the import button settings
Examples:	TP_Import (1, ImportMTEProduction)	

TP_Infolines(GuiNo, Rundown, Parameter)

Macro that operate opens / closes the info lines in the rundowns

GuiNo	GuiNo of the Gui which will execute the command
<Rundown> *Jingles, Show, Stack<n>* or *****	* = the rundown which is selected (highlighted by a frame around)

Param1	Param2
Toggle *Toggle the infolines on/off*	
Scroll *Scrolls the infotext lines of the selected element in the rundown*	**Up / Down / PageUp / PageDown / FirstPage / LastPage**

TP_LoadCurrentShow (GuiNo, Type)

Command loads the current show according to time of day

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	Type	Today, SelectedDate
Examples:	TP_LoadCurrentShow (1, Today) loads the current show of today; TP_LoadCurrentShow (1, SelectedDate) loads the current show of the just selected date;	
Hint:	You can also use the command TP_LoadNode of the engine to load the current show.	

TP_LoadJinglegroup (GuiNo, Param)

Command loads a dedicated jinglegroup

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	Param	Name=<Name of the jinglegroup to be loaded> Id=<Id of the jinglegroup to be loaded>
Examples:	TP_LoadJinglegroup(1, Name=Der schöne Morgen/radioEINS); TP_LoadJinglegroup(1, Id=00000000/00000066/00000004/00000064)	

TP_MiniDBM(GuiNo, Param1, Param2, Param3)

Commands refer to the miniDBM within TurboPlayer

GuiNo	GuiNo of the Gui which will execute the command		
Param1:	Param2:	Param3:	Param4:
Actualize *Actualizes the content view.*	-	-	-
AllFilters *Activates/deactivates all selection boxes (filters).*	**On, Off, Toggle**	-	-
CopyItems *Copies the first <n> items of the current selection into the specified rundown, at the end or NEXT position. The count <n> is limited to 20 items.*	**Rundown** *Show, Stack1..15*	**NEXT, End** *Copies the item at NEXT or end position in the rundown list*	**<n>** *The number of items to copy; limited to 20 items.*
CopySelectedItem *Copies the selected item into the specified rundown at the given position.*	**Rundown** *Show, Stack1..15*	**Next, End** *Copies the item at NEXT or end position in the rundown list*	-
CursorInSearchTextbox *Sets the cursor into the search text field. Press Escape to leave this field and set focus back to main application.*	-	-	-
DefinitionToolbar *Shows/hides the selection toolbar.*	**On, Off, Toggle**	-	-
LoadInPlayer *Loads the selected item into the specified player.*	**Player, Channel** *If Channel is specified the player is used for which the channel is assigned.*	**<1..n>** *the player resp. channel number*	-
Prelisten *Prelisten command for the selected item.*	**Begin, End, Exit**	-	-
Scroll *Scrolls the list and does not change the selection.*	**Up, Down, PageUp, PageDown, FirstPage, LastPage**	-	-
Selection *Changes the selection*	**Up, Down, PageUp, PageDown, Left, Right, First, Last** *Selection left/right is for cart view (has no action in list view)*	-	-
SetCriterion *Sets the specified criterion for selection <n>, activates the selection and opens the corresponding content box to select a value.* *If filter value is -1 the first selection box which matches the given criterion is activated and its content box is opened for further selection.* *If the definition toolbar is not visible this command does nothing.*	**<Criterion>** One of: *Author, BroadcastShow, Category, Class, Composer, Correspondent, Customer, Department, Editor, Flags, Group, Intensity, Label, Language, Performer, Presenter, Product, ProductGroup, Program, Project, Publisher, Seasonal, Speaker, SubDepartment, Type*	**<1..n>** *Number of the selection (filter)*	-
ToggleFilter *Activates/deactivates the specified selection criterion (filter)* *If the definition toolbar is not visible this command shows the toolbar if necessary.*	**<1..n>** *Number of the selection (filter)*	**On, Off, Toggle**	-
ToggleOpenSearchMask *Opens/closes the dialog to set the search fields.*	-	-	-

TP_Move (GuiNo, Type, WindowNo, Option)

The first call of the command selects the item to move and the second call of the command moves the item to the new cursor position

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	Type	Show, Stack<1..15> or * for the selected rundown list
	WindowNo	The window number <1..n> of the specified rundown list
	Option	This optional parameter can be used to exit an active move state without any further operation by setting the parameter to "Exit".
Examples:	TP_Move (1, Show, 1)	

TP_NewElementColor (GuiNo, Rundown, Command)

Changes the highlighting of new elements in a rundown and/or the corresponding time info window.

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	Rundown	Show, Stack<1..15> or * for the selected rundown list (marked with a light colored frame)
	Command	At the moment only the command "Reset" is supported. An active highlighting of new elements is reset.
Examples:	TP_NewElementColor (1, Show, Reset)	

TP_OpenSelection (GuiNo, Type)

Opens one of the selection windows

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	Type	One of: Program, Date, Show, Jingles
Comment:	This command opens the specified selection window, which typically means that a combo box is dropped down and the user can use the keypad to make his selection. In case of type "Show" the free show selection dialog will appear when the FreeShowlist mode is active.	
Examples:	TP_OpenSelection (1, Jingles)	

TP_Player (GuiNo, PlayerNo, Type, Param)

Changes properties of a player.

Parameter:	GuiNo	GuiNo of the Gui which will execute the command or * for all Guis
	PlayerNo	The number of the player 1..n
	Type:	Param:
	Loop	On, Off, Toggle – changes the loop mode, only available if TurboPlayer is running with a MultiPlayer V3.
Examples:	TP_Player (1, 1, Loop, Toggle)	

TP_Prelisten (GuiNo, Type, Param1, Param2)

Prelisten commands

Parameters: GuiNo — GuiNo of the Gui which will execute the command or * for all Guis

Depending on the type different parameters are possible. The second column below lists the possible types whereas the 3rd and 4th columns list the corresponding values:

Type:	Param1:	Param2:
FastPrelisten (since V 2.2)	**0** (take begin), **MarkIn**, **MarkOut** (plays before MarkOut till MarkOut), **End** (plays before take end), **LinkOut** (plays before LinkOut till take end) - starts always the fast prelisten by MultiPlayer engine	
PlayAtPos	*0, MarkIn, MarkOut, End, LinkOut* - refers either to CFM if active or FastPrelisten by MultiPlayer or EasyPlayer; CFM: *0, MarkIn* -> plays FadeIn part *End, MarkOut, LinkOut* -> plays FadeOut part	
Step	Left, Right (refers to the active prelisten either FastPrelisten or CFM)	
ToggleLoop	MarkIn, MarkOut (refers to the active prelisten either FastPrelisten or CFM)	
Exit	Save, NoSave (refers to the active prelisten either FastPrelisten or CFM)	
LoadCFM	Play, NoPlay, PlayEnd	Specifies the **rundown = Show, Jingles, Stack1..15, PrelistenSelected or ***; If not specified always the showlist selected item will be loaded into CFM; PrelistenSelected = last selected item (mouse click or last selection change in any rundown) * = the selected rundown (is highlighted by a coloured frame);
CFM_ToggleBeginEnd	(refers only to CFM)	
CFM_ToggleGainAmplification	(refers only to CFM)	
CFM_TogglePlayStop	(refers only to CFM)	
CFM_ResetMarkInOut	(refers only to CFM)	
CFM_Zoom	In, Out (refers only to CFM)	
CFM_NumberOfTracks	1, 2 or Toggle	
MoveSoundhead	**<time in msec>**, neg. value indicates move towards MarkIn/ file begin	

Examples: TP_Prelisten (1, PlayAtPos, MarkIn); TP_Prelisten (*, LoadCFM, NoPlay)

TP_Prelisten2 (GuiNo, Type, Parameters...) – Since version 3.2 (Build >= 800)

GuiNo GuiNo of the Gui which will execute the command or * for all Guis

Depending on the type different parameters are possible. The second column below lists the possible types whereas the 3rd and 4th columns list the corresponding values:

Type: MP (MultiPlayer prelistening)

Parameters

Start

Starts prelistening on the specified channel

[Note]
If the combination Prepared/NEXT or Prepared/Selected starts a prepared item on the channel a further call of the same Start macro refers to this previously prepared item than the NEXT or Selected.

Ch=n
Channel n=-1, 1..
Value -1 indicates the lowest channel specified for prelistening for that GUI, see \Lines\ Line-x\ Channel-y\ RundownListType = Prelisten

NEXT
the element which is NEXT in the rundown (only valid if a rundown or '*' is specified)

Selected
the selected element in the rundown or MiniDBM

Prepared
Item prepared on channel n

Prepared/NEXT
Prelistens the item prepared on channel, if no prepared item prelistens the NEXT of the specified rundown

Prepared/Selected
Prelistens the item prepared on channel, if no prepared item prelistens the selected of the specified rundown

ID
The full node ID of an element in the specified rundown. Put this parameter into quotation marks so that a node ID is not interpreted as a divison.

GUID
The GUID of an element in the specified rundown. A GUID must have enclosing curly braces: { .... } Put this parameter into quotation marks so that a GUID is not interpreted as a subtraction.

Attention: This parameter must not be specified for "Prepared "! Instead specify the next option (like 0, MarkIn, ...) as 5. parameter.

<Rundown>
Jingles, Show, Stack<n>

***

the rundown which is selected (highlighted by a frame around)

PrelistenSelected
the element which is selected for prelistening (last clicked/selected element). If you use this value, the previous parameters does not matter.

MiniDBM
The selected item within MiniDBM

0
plays from take begin

MarkIn
plays from MarkIn

MarkOut
plays before MarkOut

End
plays before take end

LinkOut
plays before LinkOut

Fast and successive press of end keys (MarkOut, End, LinkOut) positions further before end.

Exit Stopps prelistening	**Ch=n** ... stopps channel n=-1, 1.. **All** ... stopps all channels Channel value -1 indicates the lowest channel specified for prelistening for that GUI, see \Lines\ Line-x\ Channel-y\ RundownListType = Prelisten	**NoSave** **Save**... saves any changed Mark value
MoveSoundhead Positions the soundhead in one step relative to the current position	**Ch=n** ... channel number n=-1, 1.. Channel value -1 indicates the lowest channel specified for prelistening for that GUI, see \Lines\ Line-x\ Channel-y\ RundownListType = Prelisten	**Left** ... moves soundhead towards MarkIn/ begin in little steps; keep shortcut pressed for faster movement **Right** ... moves soundhead towards MarkOut/ end in little steps; keep shortcut pressed for faster movement **<Time> [msec]** ... neg. time value indicates move towards MarkIn/ begin; towards MarkOut/ end otherwise
PlayAtPos Sets the position	**Ch=n** ... channel number n=-1, 1.. Channel value -1 indicates the lowest channel specified for prelistening for that GUI, see \Lines\ Line-x\ Channel-y\ RundownListType = Prelisten	**0** ... plays from take begin **MarkIn** ... plays from MarkIn **MarkOut** ... plays before MarkOut **End** ... plays before take end **LinkOut** ... plays before LinkOut Fast and successive press of end keys (MarkOut, End, LinkOut) positions further before end.
SetMark Sets the mark point to the current play position	**Ch=n** ... Channel n = -1, 1.. Value -1 indicates the lowest channel specified for prelistening for that GUI, see \Lines\ Line-x\ Channel-y\ RundownListType = Prelisten	**MarkIn** **MarkOut** Enters/ leaves loop play around the MarkIn/ Out point, which can be moved by MoveSoundhead command Left/ Right
ResetMark Resets the MarkIn/MarkOut points: MarkIn to 0, MarkOut to take end.	**Ch=n** ... Channel n = -1, 1.. Value -1 indicates the lowest channel specified for prelistening for that GUI, see \Lines\ Line-x\ Channel-y\ RundownListType = Prelisten	**MarkIn, MarkOut, MarkInOut** References MarkIn only, MarkOut only or both.

Type: EP (EasyPlayer / MultiRec)

Parameters

Start

Starts prelistening

<Rundown>
Show, Jingles, or Stack<n>

*****

the rundown which is selected (highlighted by a frame around)

PrelistenSelected
the element which is selected for prelistening (last clicked/selected element)

MiniDBM
The selected item within MiniDBM

Ch=n
Prelisten what is playing/prepared on channel n = 1..

Attention: This parameter must not be specified for "PrelistenSelected", "MiniDBM" or "Ch=n"! Instead specify the next option (like 0, MarkIn, ...) as 5. parameter.

NEXT
the element which is NEXT in the rundown (only valid if a rundown or '*' is specified)

Selected
the selected element in the rundown or MiniDBM (for MiniDBM, if you want to use any parameter of the next column, like "MarkInMarkOut", **don't** use the "Selected" option, but directly the parameter. Ex: *TP_Prelisten2 (1,EP,Start,MiniDBM, MarkInMarkOut)*).

ID
The full node ID of an element in the specified rundown. Put this parameter into quotation marks so that a node ID is not interpreted as a divison.

GUID
The GUID of an element in the specified rundown. A GUID must have enclosing curly braces: { .... } Put this parameter into quotation marks so that a GUID is not interpreted as a subtraction.

0
plays from take begin

MarkIn
plays from MarkIn

MarkOut
plays before MarkOut

MarkInMarkOut
plays between

MarkIn and MarkOut

BeforeMarkOutAndStop
plays ~5 seconds

before MarkOut and

stops at it

End
plays before take end

LinkOut
plays before LinkOut

Fast and successive press of end keys (MarkOut, End, LinkOut) positions further before end.

Exit	Stops prelistening
MoveSoundhead Either FFwd/ FRwd mode or sets the soundhead relative to the current position	**FastForward** ... Enters/ leaves fast forward playback mode **FastRewind** ... Enters/ leaves fast rewind playback mode **<Time> [msec]** neg. time value indicates move towards MarkIn/ begin; towards MarkOut/ end otherwise
PlayAtPos Sets the position of the active prelistening. If no prelistening is active the prelisten selected (last clicked/selected) item will be started.	**0** ... plays from take begin **MarkIn** ... plays from MarkIn **MarkOut** ... plays before MarkOut **End** ... plays before take end **LinkOut** ... plays before LinkOut **MarkInMarkOut** … plays between MarkIn and MarkOut **BeforeMarkOutAndStop** … starts to play ~5 seconds before MarkOut and stops at MarkOut Fast and successive press of end keys (MarkOut, End, LinkOut) positions further before end.

Type: CFM (CrossfadeMixer) or Type: OTM (OnAIR TrackMixer)

Parameters

Load Loads the element into CFM / OTM	**Play** load and play **NoPlay** only load the element **PlayEnd** load and play fade-out part	**<Rundown>** Jingles, Show, Stack<n> ***** the rundown which is selected (highlighted by a frame around) **PrelistenSelected** the element which is selected for prelistening (last clicked/ selected element) **MiniDBM** The selected item within MiniDBM **Ch=n** Load what is playing/prepared on channel n=1..	**Attention:** For "PrelistenSelected", "MiniDBM" and "Ch=n" the only possible keyword is "Selected". The other 3 are only available if a rundown is specified in the previous parameter. **Selected** the selected element in the rundown. **NEXT** the element which is NEXT in the rundown (only valid if a rundown or '*' is specified) **ID** The full node ID of an element in the specified rundown. Put this parameter into quotation marks so that a node ID is not interpreted as a divison. **GUID** The GUID of an element in the specified rundown. A GUID must have enclosing curly braces: { .... } Put this parameter into quotation marks so that a GUID is not interpreted as a subtraction.
Save	Saves changed metadata for the currently loaded transition.		
SaveAndClose	Like Save but empties CFM/OTM and hides the CFM/OTM window afterwards.		
Exit Stopps prelistening	**NoSave** **Save**.... Saves any changed values (Mark, Fade, Link, Amplification, Gain, etc.)		
Transition	**Prev** load the previous transition **Next** load the next transition		

MoveSoundhead Either moves the MarkIn/ Out point or sets the soundhead relative to the current position	**Left** moves MarkIn/ Out (depending whether view is FadeIn/ Out part) in little steps **Right** moves MarkIn/ Out (depending whether view is FadeIn/ Out part) in little steps **<Time> [msec]** moves the current soundhead position neg. time value indicates move towards MarkIn/ begin; towards MarkOut/ end otherwise
NumberOfTracks Sets the number of tracks	**1** ... one track **2** ... two tracks (crossfade) **Toggle** ... toggles between one-/ two track mode
PlayAtPos Plays either from begin or the end	**MarkIn** ... plays the fade-in part **MarkOut** ... plays the fade-out part Press of MarkOut key successively positions further before end.
ResetMarkInOut	Resets the mark values to its defaults
ToggleBeginEnd	Toggles between fade-in and fade-out part
SetMarkToSoundhead	Sets the MarkIn/ Out (depending whether view is fade-in/ -out part) to current play position
ToggleGainAmplification	Toggles activation of gain and amplification: either the gain or amplification value can be entered in the activated edit field and the corresponding volume line can be dragged in waveform;
TogglePlayStop	Toggles between play and stop
Zoom Sets the visible waveform zoom range	**In** ... increases the zoom range **Out** ... decreases the zoom range
Control	Sends commands to CFM or OTM for execution. The following command IDs are defined: (for CFM only up to number 28, higher numbers are only available with OTM. You can either specify the number or the string.) (1) Record (2) Play (3) PlayFRwd (4) PlayFFwd (5) Stop (6) Undo (7) Redo (8) ZoomIn (9) ZoomOut (10) Default (11) MoveStart (12) MovePos (13) MoveEnd (14) ToggleInOut (15) MoveMarkLeft (16) MoveMarkRight (17) MoveAudioLeft (18) MoveAudioRight (19) MoveMarkToSoundhead (20) ClearRecord (21) TogglePlay

(22) SetIntro1
(23) SetIntro2
(24) SetOutro1
(25) SetOutro2
(26) PlayFromMarkIn
(27) GotoLink
(28) ToggleGainAmplification
(29) AutoDuck
(30) NextTransition
(31) PrevTransition
(32) Save
(33) Cancel
(34) AddTrack
(35) AddTrackAndLoadNext
(36) EmptyTrack
(37) RemoveTrack
(38) TrackSelectionUp
(39) TrackSelectionDown
(40) AutoCrossfade
(41) SetMarkIn
(42) SetMarkOut
(43) SetLinkIn
(44) SetLinkOut
(45) SetFadeIn
(46) SetFadeOut
(47) ToggleVoiceTracking
(48) VoiceTrackingStep
(49) ToggleWaveformOffset
(50) StartNext
(51) ToggleZoomAllOut
(52) CollapseExpandTracks
(53) RecordStartNext
(54) CollapseExpandTracksZoomAllOutIn
(55) ResetFadecurve
(56) AvtEnterMode
(57) AvtMusic1StartDucked
(58) AvtMusic1StartUnducked
(59) AvtMusic2StartDucked
(60) AvtMusic2StartUnducked
(61) AvtDrop1StartDucked
(62) AvtDrop1StartUnducked
(63) AvtDrop2StartDucked
(64) AvtDrop2StartUnducked
(65) AvtMusic1Down
(66) AvtMusic1Up
(67) AvtMusic2Down
(68) AvtMusic2Up
(69) AvtDrop1Down
(70) AvtDrop1Up
(71) AvtDrop2Down
(72) AvtDrop1Up
(73) RecordStartNextWithAutoDucking
(74) PlayToMarkOut
(75) ResetAndRestartRecording
(76) MoveMarkInLeft
(77) MoveMarkInRight
(78) MoveMarkOutLeft
(79) MoveMarkOutRight
(80) ResetTransition

TP_Selection (GuiNo, Rundown, Value, WindowNo)

Sets the cursor (e.g. in the show) or the NEXT element (e.g. for jingles)
Selecting an item automatically sets the prelisten focus on this item (next prelisten start command refers to this item).

Parameter:	GuiNo	GuiNo of the Gui which will execute the command or * for all Guis
	Rundown	Show, Jingles, Stack<1..15> or * for the selected rundown list (marked with a light colored frame)
	Value	Up / Down / PageUp / PageDown / Full node ID / GUI Position (1..n) / NEXT; Position means: the element with that position in list will be cursor selected or becomes NEXT (only if rundown mode is not sequenced); NEXT means: the cursor will be set on the NEXT element;
	WindowNo	The window number <1..n> of the specified rundown list type.
Examples:	TP_Selection (1, Show, Up, 1) TP_Selection (1, Jingles, 16) TP_Selection (1, Jingles, Down) TP_Selection (1, Show, NEXT) TP_Selection (1, Jingles, “00000000/00000067/00000004/00000071/00000070”)	

TP_SelectJingletab (GuiNo, Up/Down, Load/NoLoad, [WindowNo])

Selects one of the tabs of the Jingles window

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	Up/Down	Selects the next / previous jingletab
	Load/NoLoad	Loads or does not load the new jinglebank selected by the jingletab
	WindowNo	Optional, the number of the jingle window (1..n), if not specified window no. 1 is addressed;
Examples:	TP_SelectJingletab (1, Up, Load)	

TP_SetNext (GuiNo, Rundown, [Parameter])

Copies or moves (depends on send state) the selected item to NEXT position in rundown

GuiNo	Gui number of the GUI which will execute the command
Rundown	Show, Stack<1..15> or * for the in the GUI selected rundown list

Parameter:

LastPlaying	Makes a copy of the last playing item to become the new NEXT; Example of use: advertising items are running, sequence must be faded out, the last (out-fading = still playing) commercial item can so easily be copied to become the NEXT again;
LastPlayed	Makes a copy of the last played item to become the new NEXT; Example of use: advertising items are running, sequence must be interrupted, the last (interrupted) commercial item can so easily be copied to become the NEXT again;
LastPlayingOrPlayed	Ditto: To copy of the last playing item has priority over the last played item; means if there is no current playing the last played item will be copied to the NEXT position (becomes the new NEXT).
LastPlayed, Delete	Same as LastPlayed, but it deletes the origin last played element
BehindGroupIfClassNotAllowed	If the NEXT is within a group and the class of the item which should be set to become NEXT is not allowed to the group then this parameter copies/moves the item just behind the group, otherwise the macro would do nothing.
InsertIntoStoriesNotAllowed	If the NEXT is within a story, this parameter will prevent the item which should be set as NEXT to be copied inside the given story.If parts of the story are already broadcasted/broadcasting, the copied item will be placed below the story. If the story is not played the copied item will be placed above the story.
WindowNo	The window number <1..n> (sets the selected cursor item in this window of the specified rundown list type as NEXT)

Comment: For a sequenced rundown "last" means the lowermost playing/played element in the rundown. For all other rundown modes the list is not ordered by start time, therefore the element with the latest start time is taken.
Please note: there is also the engine command TP_Recall below which can reactivate sent elements in a rundown.

Examples: TP_SetNext (1, Show, 1); TP_SetNext (1, *, BehindGroupIfClassNotAllowed)

TP_Skip (GuiNo, Type, WindowNo)

Sets the current NEXT element in the rundown to sendstate Skipped

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	Type	Show, Stack<1..15> or * for the selected rundown list
	WindowNo	The window number <1..n> of the specified rundown list

Examples: TP_Skip (1, Show)

TP_StandardScreen (GuiNo, Type, WindowNo)

Sets the standard screen and cursor position in the rundown

Parameter:	GuiNo	GuiNo of the Gui which will execute the command or * for all Guis
	Type	Show, Stack<1..15> or * for the selected rundown list
	WindowNo	The window number <1..n> or * for all windows of the specified rundown list
Examples:	TP_StandardScreen (1, Show, 1)	

TP_Story (GuiNo, WindowNo, Command, Param)

Sends commands to a DigaStory window

Parameter:	GuiNo	The number of the Gui which executes the command or * for all Guis
	WindowNo	The window number <1..n> (= instance of DigaStory)
	Command	See the table below for the possible commands DigaStory can handle.
	Param	Depending on the command an additional parameter can be specified, see the command description below. If not explicitly mentioned, the parameter can be omitted.
Examples:	TP_Story (1, 1, ScrollLineDown, 3); TP_Story (1, 1, PressPlayOfNext)	

Commands and shortcuts for DigaStory

The commands must be sent to DigaStory via the OCX function “ExecuteCommand”. This is done by TurboPlayer with the “TP_Story” macro command. It is possible to assign a key shortcut to each of these commands in the general settings dialog – tab “Shortcuts” - of TurboPlayer. Hint: by default this subdialog lists only a few standard commands for DigaStory and GUI 1 / window 1. If you want to use one of the many other commands or if you have multiple GUIs / multiple DigaStory windows, you have to use the extended function “Create a new command”.

The configured key shortcuts work whenever TurboPlayer has the input focus and each command can even be hooked, to make it work if any other application is active. But pay attention: you must not use any key combinations which are valid text input (e.g. simple keys like ‘A’) or which are used for text navigation/selection (like PageUp or Ctrl-Right). If you try to define such shortcuts, either text input will not be possible or the shortcuts will not work correctly.

In contrast, the standard shortcuts of DigaStory cannot be configured and are only available if the input focus/caret is within the DigaStory window.

Command	Std shortcut	Description
Loading, unloading, initalizing		
Hint: All the commands in this section are recognized by TurboPlayer and are not available in other programs.		
LoadCursorNode	-	Loads the node of the assigned rundown which has the cursor. This is like the fill mode “follow cursor” but triggered by macro command
LoadPrev	-	Load the previous story or element.
LoadNext	-	Load the next story or element.
EditMode	-	Changes the edit mode (read-only or editing is active).Param can be On, Off or Toggle.
EditModeAndSetFocus	-	Same as EditMode but in addition the input selection/focus/caret is set to the main node or to the first text element if a story is the main node. If EditMode is switched off, the focus is set back to the main TurboPlayer window.
Exit	-	Performs a cancel operation. If changes were made the user is asked whether he wants to commit or undo his changes. Afterwards DigaStory is empty.
Save	-	Performs a save operation. Hint: DigaStory permanently saves all changes in the background, so that they are visible to other users / on other computers. The “save” only commits the changes in the undo handler of DigaStory. After saving the edit mode is switched off.
SaveAndExit	-	Performs a save and afterwards an exit operation, meaning: DigaStory is emptied after saving.
Scrolling, cursor / selection movement		
AutoScrollStart	-	Begin auto-scrolling (Param = pixels/sec).

AutoScrollStop	-	Stop auto-scrolling.
AutoScrollFasterFwd	-	Accelerate auto-scrolling.
AutoScrollFasterBwd	-	Deccelerate auto-scrolling.
-	(Ctrl-)Up	These keys work as expected if the caret has been set into a text (like in every other text editor). If nodes are selected these keys move the node selection and if the Ctrl key is being pressed in addition, they move the vertical scrollbar.
-	(Ctrl-)Down	
-	(Ctrl-)PageUp	
-	(Ctrl-)PageDown	
-	(Ctrl-)Top	
-	(Ctrl-)Bottom	
ScrollPixelUp	-	Scroll n pixels up (n = Param).
ScrollPixelDown	-	Scroll n pixels down (n = Param).
ScrollLineUp	-	Scroll n lines up (n = Param).
ScrollLineDown	-	Scroll n lines down (n = Param).
ScrollPageUp	-	Scroll n pages up (n = Param).
ScrollPageDown	-	Scroll n pages down (n = Param).
ScrollTop	-	Scroll to top of loaded nodes.
ScrollBottom	-	Scroll to bottom of loaded nodes.
SelectionUp	-	Move the selection one node up.
SelectionDown	-	Move the selection one node down.
SelectionTop	-	Move the selection to the top / first subwindow.
SelectionBottom	-	Move the selection to the bottom / last subwindow.
SelectionOntoNext	-	Set the selection onto <next> element.
SelectFirstMediaElement	-	Set the selection onto the first media element (=audio, video or graphic).
SelectMainNode	-	Set the selection onto the main node (=the one initially loaded by the user).
ToggleSelectionTextfocus	Ctrl-Enter / Enter Ctrl-Insert / Insert	Changes between node selection and the focus/caret inside of a text. The standard commands without “Ctrl” work only to set the focus/caret into a text. Afterwards they make a line break and change the insert mode, of course.
Editing		
Cut	Ctrl-X	Move the selected text or the selected nodes into the Windows clipboard.
Copy	CtrlC	Copy the selected text or the selected nodes into the Windows clipboard.
Paste	Ctrl-V	Paste the clipboard content into the text (if there is text in the clipboard) or into the rundown (if there are BCS nodes or DBM entries in the clipboard).
DeleteSelected	-	Deletes all elements with a node selection. Stories cannot be deleted.
Bold	Ctrl-Shift-B Ctrl-Shift-F	Toggle the character attribute “bold” for the selected text. If language is set to “German” the ‘F’ is being used.
Italic	Ctrl-Shift-K	Toggle the character attribute “italic” for the selected text.
Underline	Ctrl-Shift-U	Toggle the character attribute “underline” for the selected text.
ParaLeft	Ctrl-L	Set the paragaraph alignment to “left”.
ParaCenter	Ctrl-E	Set the paragaraph alignment to “centered”.
ParaRight	Ctrl-R	Set the paragaraph alignment to “right”.
ParaJustify	Ctrl-J	Set the paragaraph alignment to “justify”.
SelectFont	-	Open the font selection combo box.
SelectFontSize	-	Open the font size combo box.
SelectColor	-	Open the color combo box.
SelectSpeaker	-	Open the speaker dropdown dialog.
ComputeDuration	-	Compute new text durations for all text elements.
Undo	Ctrl-Z	Perform an undo step.

Redo	Ctrl-Y	Perform a redo step.
Misc		
OpenContextMenu	-	Open the context menu, either for text (if the caret has been set into a text) of for the full view (if nodes are selected or the caret has not been set).
LockSelected	-	Locks/unlocks the selected nodes. If param=1 the nodes are locked, if param=0 the nodes are unlocked, if param is “Toggle” or empty the lock state is toggled.
PressStartOfNext	-	Press play/stop button of <next> element.
PressStartOfSelected	-	Press play/stop button of selected element.
PressToggleOfNext	-	Press toggle (send state) button of <next> element.
PressToggleOfSelected	-	Press toggle (send state) button of selected element.
TogglePrelistenOfSelected	-	Start/stop prelisten of selected element.
ToggleStartModeOfNext	-	Change start mode between manual <-> sequenced of <next> element.
ToggleStartModeOfSelected	-	Change start mode between manual <-> sequenced of selected element.
Record		
StartRecording	-	Starts recording over the selected element, if it is an element and a valid one. Just one element can be recorded at a time, which means that a second call of this option will fail if there is already something being recorded.
StopRecording	-	Stops the recording of the element being recorded (if any is at all) and saves the recorded audio. If the element was previously a text element, it turns to be an audio element with the original text saved in presentation. If the element was already an audio element, then the old audio part of the element is replaced by the recently captured audio.
EscapeRecording	-	Like the StopRecording option, this will also stop the recording, but nothing will be saved and the recorded content will be discarded.

TP_Text (GuiNo, Ident, Param)

Enters or leaves edit mode, scrolls or holds the text window

Parameter:	GuiNo	The number of the Gui which executes the command or * for all Guis	
	Ident	Text	Command refers to the presentation text
		Infotext	Command refers to the infotext
		Message	Command refers to the DigaMessage window
	Param	Edit	Enters in edit mode with cursor inside the edit control or leaves the edit mode if already in edit mode (= toggle);
		ScrollUp	Scrolls text up
		ScrollDown	Scrolls text down
		Hold	Toggles the hold state of the text view
		HoldOn	Changes the hold state of the text view to On/Active
		HoldOff	Changes the hold state of the text view to Off/Inactive
Examples:	TP_Text (1, Text, ScrollUp);	TP_Text (1, Text, Edit); TP_Text (1, Text, HoldOn)	

TP_Timeline (GuiNo, Rundown, Value, WindowNo)

Calls a given function of the timeline, which is usually corresponds to one of its buttons.

Parameter:	GuiNo	GuiNo of the Gui which will execute the command or * for all Guis
	Rundown	Show, Stack<1..15> or * for the selected rundown list (marked with a light colored frame).
	Value	Home / ToggleLink / Zoom in / Zoom out / Left / Right; The macro will execute the exact same function executed by pressing the respective timeline buttons;
	WindowNo	The window number <1..n> of the specified rundown list type.
Examples:	TP_Timeline (1, Show, Zoom In, 1) TP_Timeline (1, Show, ToggleLink) TP_Timeline (1, Stack1, Home) TP_Timeline (1, Stack2, Left)	

TP_ToggleDSSColumn (GuiNo, Rundown, Change)

Changes the content which is being displayed in the DSS (duration / start time / stop time) column of the specified rundown list.

GuiNo	The number of the Gui which will execute the command
Rundown	One of: Show, Stack<n>
Change	- Duration: audio and/or text duration is displayed. - StartTime: the start time is displayed. - StopTime: the stop time is displayed. - ToggleContent: toggles the content as: duration – start time – stop time. - ModeAudio: duration mode for audio elements is: only audio duration. - ModeAudioText: duration mode for audio elements is: audio plus text duration. - ToggleMode: toggles duration mode for audio elements.
Comments:	This command is intended to achieve a behaviour like for the same column in NewsPlayerX. Therefore it is also possible to display a "c" at the end of the column in case a mark-in (=cue) is being set. This can only be activated permanently in the list settings. There you can also choose whether the times / duration are displayed for the mark or the link points. A rect in a brighter or darker background color signals the user whether start or stop times are being displayed. This is also visible from the legend above the columns. Attention: Changing the content is handled by TurboPlayer internally. But in BCS system the time calculation is done only by BCS. Therefore changing the duration mode requires to modifiy this field for all elements, triggering a BCS time recalculation.
Examples:	TP_ToggleDSSColumn (1, Show, ToggleContent)

TP_ToggleListSelection (GuiNo, [<Rundown>])

Toggles rundown selection (important to receive key input, e. g. selection up/down) – highlighted by a light frame around – through all rundown windows of list types which can have a cursor (Show, Jingle, Stacks)

GuiNo	The number of the Gui which will execute the command
Rundown	Parameter is optional; if not specified the macro toggles through the rundown windows; otherwise (Show, Jingles, Stack<n>) the specified rundown window is selected
Examples:	TP_ToggleListSelection(1), TP_ToggleListSelection(1, Stack1)

TP_ToggleOpenGroup (GuiNo, Rundown, [WindowNo], [Open/Close])

Opens/closes the cursor selected group

Parameter:	GuiNo	GuiNo of the Gui which will execute the command or * for all Guis
	Rundown	Show, Stack<1..15> or * for the selected rundown list
	WindowNo	Optional: The window number <1..n> of the specified rundown
	Open/Close	Optional: Opens or closes the group; If parameter is missed it toggles;
Examples:	TP_ToggleOpenGroup (1, Show)	

TP_ToggleSendState (GuiNo, Type, WindowNo)

Toggles the sendstate (planned <-> skipped) of the selected item in the rundown and specified list

Parameter:	GuiNo	GuiNo of the Gui which will execute the command
	Type	Show, Stack<1..15> or * for the selected rundown list
	WindowNo	The window number <1..n> of the specified rundown list
Examples:	TP_ToggleSendState (1, Show, 1)	

TP_ToggleStartMode (GuiNo, <Rundown>, [WindowNo] or All or NEXT)

Toggles the startmode (Manual <-> Sequenced) of the selected item in the rundown and specified list

GuiNo	GuiNo of the Gui which will execute the command
Param1	Param2
Rundown or * *Show, Stack<1..15> or* * *for the selected rundown (highlighted by a frame around)*	**[WindowNo]** *Optional: The window number 1..n of the specified rundown list. Only the selected item of this window of the rundown is toggled; usually one window is configured to be shown for a rundown, e. g. one showlist window.* **All** *Toggles all items in the rundown (of the active show)* **NEXT** *Toggles only the NEXT item in the rundown*
Examples:	TP_ToggleStartMode (1, Show, 1), TP_ToggleStartMode (1, Show, All)

TP_Trashcan (GuiNo, Param1, Param2, …)

Macro that operate on the trashcan

GuiNo	GuiNo of the Gui which will execute the command	
Param1	Param2	Param3
Open *Opens the trashcan*	**--, Toggle** Toggles open and close	
Close *Closes the trashcan*	--	
Selection *Changes the selection item in trashcan*	**Up, Down**	
Insert *Inserts the selected trashcan item into a rundown at the specified position*	**<Rundown>** *Jingles, Show, Stack<n>* ***** *the rundown which is selected (highlighted by a frame around)*	**AboveCursor, BelowCursor**

ToggleFocus

Toggles the focus between turboplayer main application and the trashcan dialog (this is helpful to use e. g. the Up/Down or PageUp/~Down keys in the rundown as well as in the trashcan window (trash has focus))

Executed by the Kernel

Sleep (Duration)

Performs a pause / sleep in the command execution.

Parameter: Duration The pause time in milliseconds (1-10000)

Executed by: Kernel

Comment: Pay attention to a fact: the sleep command interrupts the execution of the current macro. If the macro is a sub-macro – meaning: it was called as a result of the execution of an outer macro – the outer macro is not interrupted !
The execution of the normal TurboPlayer functionality is not influenced by a sleep.
Don't use a Sleep() within a defined macro function! This does not work.

Example: Sleep (500)

TransferExecution (Computername, Flags, ReplaceVars)

The remaining command string is executed on a different computer

Parameter:

- Computername — The Windows computer name of the receiving computer
- Flags — A string with some flags, combine the keywords with underscore.
  If Flags contains "DIRECT", the direct IP communication to the remote TurboPlayers is used. If Flags contains "BCS" broadcast messages via the BCS are used. Specify something like "DIRECT_BCS" to send a macro via both ways. Default is "BCS".
  Hint: It does not matter if a macro is received multiple times, because TurboPlayer uses an unique GUID to identify each macro and will execute every macro only once.
  If Flags contains "SELF" the macro will be executed even if Computername is the name of the local computer. By default or without this flag the rest of the macro is not executed on the local computer.
- ReplaceVars — A list with variables (global, local or state) to be replaced in the macro text before it is being sent to the remote TurboPlayer. To specify multiple variables separate them by a comma.
  Attention: this parameter must be enclosed in quotation marks like: "%1,$var2" Otherwise the function TransferExecution would be called with the content of the variables, not with their name.
  All specified variables must be known on the local (=sending) TurboPlayer. Important: The rest of the macro is not interpreted or executed to find the variables to be replaced. Only a pure text comparison is being done and every occurrence of the specified variables is replaced by the current value. Afterwards the new text is sent to the remote TurboPlayer.

Executed by: Kernel, TreeManager

Comment: Must be the first command within a macro. The remaining part of the macro will be executed on a different computer. On the requested computer there must be a TurboPlayer running, of course.
Attention: Computername is a string, so you should enclose the parameter in quotation marks. This is inevitable if your computername contains a '-' because otherwise the macro processor interprets the expression as a subtraction.

Example: TransferExecution (TurbStu2)
TransferExecution ("Turbo2", "DIRECT_BCS", "%1, $ABCState")

TP_ActivateEvent (Name, IsActive)

Activates or deactivates an event

Parameter:	Name	The name of an event (registry key name below "EventsOut"). It cannot be the name of an internal command. The string comparison is case-insensitive.
	IsActive	TRUE or Yes or 1, or FALSE or No or 0, or Toggle. If nothing is specified, the event is activated.
Executed by:	Kernel	
Examples:	TP_ActivateEvent (FollowEvent, FALSE)	

TP_ActivateFaderStart (Line, IsActive)

Activates or deactivates fader starts for a line or a mixer source

Parameter:	Line	Either the line number or a known mixer source name or a mixer source group (which must be preceded by a # character, like in the BCS data).
	IsActive	TRUE or Yes or 1, or FALSE or No or 0, or Toggle. If nothing is specified, fader starts are activated.
Executed by:	Kernel	
Examples:	TP_ActivateFaderStart (Mic1, TRUE)	

TP_ActivateTimer (Name, IsActive)

Activates or deactivates the timer for a timed event

Parameter:	Name	The name of a timed event (registry key name below "EventsTimed"). It cannot be the name of an internal command. The string comparison is case-insensitive.
	IsActive	TRUE or Yes or 1, or FALSE or No or 0, or Toggle. If nothing is specified, the event is activated.
Executed by:	Kernel	
Examples:	TP_ActivateTimer (BlinkEvent, 1)	

TP_AddTimedRecording (Recorder, StartTime, StopTime, ElementXML, Exclusive)

Adds a new timed recording (=booking) via a (Multi-)coder.

Parameter:	Recorder	See the description of TP_ControlRecording.
	ElementXML	For ElementXML you should specify at least a title, because otherwise the created element will not have a title. If you omit this parameter completely TurboPlayer will create an English default title for the recording.
	StartTime	A timestamp in format YYYY-MM-DD hh:mm:ss.fff when the recording should start. The milliseconds are optional (MultiCoder cannot handle them).
	StopTime	A timestamp in format YYYY-MM-DD hh:mm:ss.fff when the recording should stop. The milliseconds are optional (MultiCoder cannot handle them).
	Exclusive	TRUE or Yes or 1, or FALSE or No or 0. If true all other existing (future) bookings are deleted. If false all other existing bookings are kept. This parameter is optional. If it is not present false is assumed.
Executed by:	Kernel	
Examples:	TP_AddTimedRecording ("Coder2", 2012-12-24 18:00:00, 2012-12-24 19:00:00, "<Element><Title>Christmas evening recording of %D %T</Title></Element>", FALSE)	

TP_AutoSetMarkIn (Type, SendStates, MaxElements)

Sets the mark-in of an element to "now" (see comment)

Parameter:	Type	One of the rundowns: Show, Jingles, Stack1…Stack15
	SendStates	The send states of the elements to be treated. Can be: “Sending”, “Sent” or “Sending+Sent”. The last one is the default.
	MaxElements	Maximum number of elements which are changed. Default is 1.
Executed by:	Kernel	

Comment: This function can be used when TurboPlayer is restarted after a crash. There might be an element which has been played partially and which is still in state "sending". Or a “sent” elements has not been played to the end. This function toggles the state to "planned" and sets the mark-in to a position which would have been reached if the element had played continuosly. Attention: this operation changes the mark values of the affected elements! Therefore TurboPlayer will append a hint to the title. This must be considered when an element is re-scheduled. Especially for rundowns like jingles this command must not be used because the elements will start later and later within the take with each execution of this command.

The function starts looking for elements from the bottom of a rundown to the top. In order to work correctly it is absolutely important to have a correct time synchronization between the computers.

If you write a macro which starts an element directly after calling this function you have to wait (Sleep) for some time (~100ms) because the changed metadata must be transferred from TreeManager to RundownKernel. A MultiPlayer reload of the element at the new start position is also required, but TurboPlayer is able to automatically delay a start until a load succeeds.

Do not use this function to start an element, which is still running on a second TurboPlayer. If the second TurboPlayer has some problems but is still running (and reacting to user input or macros) use for example a macro to stop the element on the second TurboPlayer (see the macro examples below). If the network connection is still good, it might be possible to send such a stop command from the starting TurboPlayer.

In TurboPlayer versions before 5.0 this command was executed by the Kernel and behaved slightly different. The parameters 2 and 3 did not exist in old TP versions.

Examples: TP_AutoSetMarkIn (Show, “Sending”, 1)

Preliminary information !

TP_CallREST (URL, Data, ContentType, User, Password)

Asynchronously establishes a connection to a REST service and sends a HTTP message.

Parameter:	URL	The URL of the REST service.
	Data	A string, which is sent as body of the HTTP message.
	ContentType	Within the message, HTTP headers are specified. “Accept: */*” is hard-coded. The “Content-Type: …” can be freely specified.
	User	A user name.
	Password	The corresponding password. TurboPlayer uses the Windows library WinInet for sending the messages. It is up to this library how the user name and the password are being used.
Executed by:	Kernel	

Comments: **Attention:** this description of the command is only preliminary! This command might be changed in future versions of TurboPlayer!

The 3 parameters URL, Data and ContentType are mandatory. User and Password are optional.

Examples: TP_CallREST (“http://test/david-gmbh.de:1234”, “… some XML … ”, “application/xml”)

TP_ClearTimedRecordings (Recorder)

Removes all future timed recordings (=bookings) of a (Multi-)coder

Parameter: Recorder See the description of TP_ControlRecording.

Executed by: Kernel

Examples: TP_ClearTimedRecordings ("Coder3")

TP_ConnectToBCS (BCSName, User, Password)

Connects TurboPlayer to a new BCS or closes the current connection.

Parameter:

BCSName
The BCS name for the new connection. It must be the name of a DigaSystem registry key below the top-level key "Digas\PlanServer".
In order to perform only a disconnect specify the BCSName="".

User
If you want to connect to a BCS and if you want to change the current DigaSystem user you can optionally specify a DigaSystem user ID here. This is not necessary if a user is already logged in (which is simply kept) or for a pure disconnect.
If the DigaSystem login fails, TurboPlayer will stop the operation and will not try to establish a BCS connection.

Password
If you specify a user you must also give the password of the user.
This password should be given with the same encryption as it is used for the registry parameter TurboPlayer\Password. Therefore, you should use Admin.exe in order to create a (dummy) parameter with the crypting method "TurboPlayer (BCS connection)". Then copy the value out of the registry and use it within your macro.
For testing purposes you can also specify a password in plain text. Therefore, you should precede the password with the text "Plain:", e.g: "Plain:my_Password". If you forget that and specify only the plain text password without a preceding “Plain:” it might work, nevertheless. TurboPlayer tries to detect, whether the specified string is correctly encrypted or not. But in order to be on the safe side: use the prefix “Plain:”.

Comments: This procedure is available since TurboPlayer version 6.0.
As first step this command always performs a reset and a disconnect. Only afterwards a BCS connection is established. Normally, you should execute a command to set the program and load a content (e.g. the current show) afterwards, because otherwise TurboPlayer will still be empty.
This command cannot be executed if the kernel has loaded at least one single element, which is not in state "stopped". Typically, it can happen that some elements are playing and switching the BCS is not possible. If this happens, macro execution is stopped with an error report in the log. Therefore it is recommended to first check, whether there is no playing or paused element, e.g. with the check:
DataOfChannel (CountInState, Playing, Paused) == 0
Hint: If an element is only prepared to a channel, it does not prevent this command.

Executed by: Kernel

Examples: TP_ConnectToBCS (“BCS5”, “ANDREA”, “Plain:P@ssw0rd”)

TP_ControlRecording (Recorder, Command, ElementXML, Immediately)

Starts or stops the live recording via MultiCoder or ROAD

Parameter:

Recorder	The recorder name if there are multiple recorders to be controlled. The settings of the TIOMultiCoder.dll and of TIOROAD.DLL hold a list of available recorders and the first field "Recorder" (or "Name" in old versions) of the list is the recorder name to be used here. If the parameter is empty the special recorder name "<Default>" is used.
Command	One of: Start, Stop, Pause, Continue, Break It depends on the settings of the IO module whether the stop command is executed as a stop or a pause. "Break" typically sets a new marker in the recording.
ElementXML	This parameter is optional for a start command and can contain metadata in the XML format for a BCS element (the root node must be <Element>). These metadata fields are converted to DBX and are transferred to the recorder to be stored with the database element, which is being created, by the recorder. See the documentation of the recording program for a list of possible fields. Within this XML string you can use some special placeholders (case sensitive !) which will be replaced before the XML string is sent to the recorder: %Program% = The current program / service name %Show% = The name of the current show %Studio% = The name of the studio (=TurboPlayer\StudioName) %Station% = The DigaSystem station name (Digas\Settings\StationName) %Computer% = The Windows computer name %Date% = The current date in international format %Time% = The current time In addition you can also use placeholders which are provided by the recording program. E.g. MultiCoder has placeholders in the title: %D for the date at which a recording starts, %T for the time at which a recording starts, %M for the computer name on which MultiCoder runs, etc.
Immediately	TRUE or Yes or 1, or FALSE or No or 0. Only used if command is: Start, Stop or Pause. See the comment for an explanation.

Comment: Be aware of the fact that there might be multiple sources which request a recording. For example one or multiple running elements can have a set record event. The same is true for running groups. In addition this macro can trigger a recording. The following logic is applied: the first start command really starts the recording, additional start commands only increment a counter and send a break message to the recorder, which creates a standard marker in the file. Each stop or pause command decrements the counter. If the counter reaches 0 the recording is really stopped.

If <Immediately> is being set, the counter is ignored and stop/pause is executed immediately. The <Immediately> parameter can be used for a start, too. This allows to send a start command even if the internal counter is not 0. This can be useful if a TurboPlayer starts recordings, but the recordings are stopped by something else. In this case the internal counter of the starting TurboPlayer will never reach 0.

Executed by: Kernel

Examples: TP_ControlRecording ("Recorder2", "Start", "<Element><Title>Manual recording </Title></Element>")

TP_ControlRecording (“<Default>”, “Stop”, “”, TRUE)

TP_ControlGuiRecording (GuiNo, Command, Rundown, Position, ElementXML, ShowRecorder)

Starts or stops the recording via a GUI.

Parameter:	GuiNo	Number of GUI which should execute the recording command.
	Command	One of: Start, StopSave, StopCancel, SetMarker
	Rundown	With Rundown and Position you must specify exactly one element for which the recording is done. There are different ways to define one element: you can specify one of: "Show", "Jingles", "Stack1".."Stack15", "TheRundown" or: "Line", "Channel". If you use Line or Channel there must be exactly one playing/paused element.
	Position	If Rundown is "Line" or "Channel" it is the line / channel number. If Rundown is a rundown, it is the full node ID or the kernel rundown position as number (starting with 1) or one of the keywords: "Next", "Cursor<n>", "LastPlaying", “LastPlayingExternal” or "TheElement". If you use one of the cursor keywords <n> denotes the GUI number (starting with 1). The special keyword “LastPlayingExternal” will also find a just created spontaneous element, even if this element has not yet been inserted into the rundown.
	ElementXML	This parameter is optional for a StopSave command. For the format see the macro command “TP_ControlRecording” above.
	ShowRecorder	TRUE or Yes or 1, or FALSE or No or 0. Only used if command is: Start. This parameter controls the visibility of the user MultiRec.ocx.
Comment:	The kernel only makes a pre-evaluation of the parameters and locates the element. The command itself is executed by the specified GUI with the help of the MultiRec_x.ocx which is in use. Only a single recording is possible for each GUI and no simultaneous MultiRec-prelistening or a usage of CrossfadeMixer or OTM within the same GUI is possible. The Rundown, Position and ShowRecorder parameters are only evaluated for the start of a recording. In order to set a marker or for stopping a recording none of them need to be specified. In contrast, for a StopSave it is possible to specify the ElementXML parameter because with this command the new metadata of the recording is written to the destination element and therefore metadata changes can be applied. Also see the macro examples below. There is an extra example for this command.	
Executed by:	Kernel / GUI	
Examples:	TP_ControlGuiRecording (1, "Start", Show, Cursor1, “”, TRUE) TP_ControlGuiRecording (1, Stop, "", "", "<Create_Creator>It was me</Create_Creator>")	

TP_Delete (Rundown, Type, Details, Flags)

Deletes a node in the rundown

Parameter:	Rundown	One of: Show, Jingles, Stack<1..15>
	Type	One of: - ID: Parameter <Details> must specify the full node ID of the node to be deleted. - Next: The <next> element of the specified rundown is deleted. - Cursor: Parameter <Details> must specify a valid GUI number. If the current cursor position of this GUI in the specified rundown is known, this element is deleted. - All: everything in the rundown is deleted. This operation is limited to rundown lists in stack mode “Clipboard”.
	Details	Either the full ID in BCS format, e.g: 00000000/00000067/00000004/00000069 (The leading zeros can be omitted.) or a GUI number. For <Next> this parameter is not needed.
	Flags	Do not specify for “All”! For the types “ID”, “Next” and “Cursor” the optional flag “NoMoveToTrash” can be given as a string. If this flag is not set, the deleted node will be moved to the day trash (if applicable).
Comments	This command is executed by the engine. Therefore, GUI properties will not be modified. This means especially, that setting the cursor from the deleted node to another sensible node cannot be done by the engine. You can use the GUI commands TP_DeleteSelected and TP_DeleteAll, too.	
Examples:	TP_Delete (Show, Next) TP_Delete (Stack2, Cursor, 1, “NoMoveToTrash”) TP_Delete (Show, ID, “00000000/00000067/00000004/00000069”)	

TP_Demute (Rundown, Time, Operations)

If there are running mute elements in the specified rundown, they are opened (de-muted) and all other running elements in the rundown are faded out.

Parameter:	Rundown	One of: Show, Jingles, Stack1..Stack15, Line, Channel
	Time	The crossfade time in milliseconds. This parameter is optional, default is the registry value "TurboPlayer\Stinger\DefaultFadeOut".
	Operations	A string with the operations to perform. It can contain the words "FadeIn" for opening the fader of the mute element, "FadeOut" for closing the fader for previous elements, or it can be the string "NoFade" for no fade operation at all. Hint: if a mute start element runs on a fully closed fader/line this line is opened in any case.
Executed by:	Kernel	
Examples:	TP_Demute (Show, 2500)	

TP_Fade (Type, Value, Time, Level)

Performs a fade, either with the Fader or the audio engine (depending on settings)

Parameter:	Type	One of: Show, Jingles, Stack1..Stack15, Line, Channel
	Value	For rundown list types it is the rundown kernel position (starting with 1), or otherwise the line / channel number (starting with 1).
	Time	The fade time in milliseconds.
	Level	The end level in dB (the fade starts at the current level). Levels <= -99 dB are interpreted as "close fader".
Executed by:	Kernel	
Examples:	TP_Fade (Channel, 1, 3000, -20)	

TP_Fadeout (Type, Value, Time, Level)

Stops an element with fadeout

Parameter:	Type	One of: Show, Jingles, Stack1..Stack15, Line, Channel
	Value	For rundown list types it is the rundown kernel position (starting with 1), or otherwise the line / channel number (starting with 1). Besides the keyword "All" can be used to fade-out all running elements.
	Time	The fade out time in milliseconds. This parameter is optional, default is the registry value "TurboPlayer\Stinger\DefaultFadeOut".
	Level	The end level in dB, optional, standard is -99 dB
Executed by:	Kernel	
Examples:	TP_Fadeout (Channel, 1, 2500); TP_Fadeout (Jingles, 1, 5000, -99)	

TP_FadeSubChannels (Type, Value, Time, Level1, Level2, Level3, Level4, …)

Performs a fade of the sub-channels (of a multi-channel audio)

Parameter:	Type	Currently only “Channel” is supported.
	Value	Channel number
	Time	The fade time in milliseconds.
	Level1 - …	The end levels in dB (the fade starts at the current level). You have to specify a value for each sub-channel, e.g. for a 4.0 channel 4 values. It is your responsibility to use values which add up to the desired total level.
Executed by:	Kernel	
Comment:	For a possible usage see the chapter „CartBeat, CartMotion“ of the „concepts“ above. This feature requires a version of MultiPlayer which can handle this command (the command is directly relayed to MultiPlayer).	
Examples:	TP_FadeSubChannels (Channel, 3, 1000, -99, -99, 0, 0)	

TP_ForNodesDo (Type, Begin, End, Direction, FullTree, Callback, UserData1, …)

Iterates over the specified range and calls a callback function for each node in the range.

Parameter:

Type	The rundown list type, one of: "Show", "Jingles", "Stack1".."Stack15".
Begin End	Either a kernel rundown position (if you want to iterate among the preloaded kernel elements) or a full node ID (if you want to iterate over the full tree). Alternatively, one of the following keywords can be used: "Next", "Cursor<n>", "LastPlaying", "TheElement", "GroupOfNext", "StoryOfNext", "GroupOfCursor<n>", "StoryOfCursor<n>", "GroupOfLastPlaying", "StoryOfLastPlaying", "TheGroupOfTheElement", "TheStoryOfTheElement", “EndpointStoryOfNext”, “EndpointStoryOfCursor<n>”, “EndpointStoryOfLastPlaying”, “TheEndpointStoryOfTheElement”."First", "Last", "FirstInActiveTree", "LastInActiveTree". If you use one of the cursor keywords <n> denotes the GUI number (starting with 1). The meaning of "First" and "Last" depends on whether you want to iterate over the rundown list or the full tree. In the first case the keyword denotes the first/last element in the list of preloaded elements. In the second case these keywords denote the first/last node in the full tree of the rundown. The keywords with "ActiveTree" can only be used when iterating over the full tree and mean: within the active show. (This is the show in which the last playing element was started, or – if nothing was started – the show with the <next> element.) The range you specify includes both border elements/nodes. This means: the callback function will be called for the last element/node, too.
Direction	Can be "Up" or "Down". Specifies the direction of iteration.
FullTree	Flag, whether you want to iterate over the kernel rundown elements (the preloaded range) or the full tree. Iterating over the full tree requires that the kernel has knowledge about this tree. Therefore, the parameter "TurboPlayer\KernelHasFullData" must be set. You can specify "true", "yes", 1, "FullTree" or "false", "no", 0.
Callback	This must be the name of a defined macro function. This function must have at least 3 parameters: the name of the rundown list, the rundown position of the element and the full node ID. The rundown and the full ID are filled in every case, but the second parameter with the index of the rundown element is only filled when you iterate over the rundown elements, not for the full tree. The callback function is called once for every iterated node. As long as true or 1 is returned, iterating continues. When the callback function returns false or 0, the iteration is stopped.
UserData1	You can specify additional parameters in the call to TP_ForNodesDo. These additional parameters will be transferred as additional parameters to the callback function. In this case the callback function must define more than 3 parameters.

Comment:

This function requires the usage of defined macro functions. They are only available in TurboPlayer version 6 or higher.

Pay attention that this command can take a very long time! This depends heavily on the number of elements/nodes over which is iterated and on the actions being taken in the callback function. While this command is running the kernel is blocked and cannot perform any other action or react on any message. Therefore, keep the amount of iterate elements/nodes as small as possible and the callback functions as quick as possible. If you are in doubt, measure the duration of execution, e.g. by putting calls to TP_Log with the CurrentTime() before and after the call to TP_ForNodesDo.

Nevertheless, do not call Sleep() within the callback function! This does not work.

You can use the rundown type and either the rundown position or the full node ID to call other functions – e.g. GetDataOfElement(). Unfortunately, these functions need to find the specified element, which can take some time. Therefore, an optimization is available: you can call some special functions, which are only available within the callback function to have a fast access to metadata for the current node or one of its parents. The following special functions are available:

- Data (FieldName)
- DataOfParent (FieldName)

- DataOfGrandParent (FieldName)
- Tagname ( )
- TagnameOfParent ( )
- TagnameOfGrandParent ( )

Executed by: Kernel

Examples:
```
function LogTitle ( rundown, position, id )
{
   TP_Log ( "*** Title of preloaded element ", %position, "=", Data ( "Title" ) )
   return true
}
TP_ForNodesDo ( Show, First, Last, Down, false, LogTitle )
```

See chapter 4.5.8 for more examples of using the TP_ForNodesDo command.

TP_LoadNode (Rundown, Type, Details)

Loads a BCS node into a rundown

Parameter: Rundown — The rundown list type, one of: "Show", "Jingles", "Stack1".."Stack15", "TheRundown"

The following combinations of Type and Details are possible:

Type:	Details:
ID	The full ID in BCS format, e.g: 00000000/00000067/00000004/00000069 (The leading zeros can be omitted.) Pay attention not to specify an unloadable node, e.g. a whole month. There is no full protection against such a wrong specification.
IDList	A comma separated list of full IDs in BCS format. Specify the IDs as additional parameters (ID2 is parameter 4, ID3 is parameter 5, …). This load type can only be used to make a full loading (including a reset) of shows into TurboPlayer when the FreeShowlist mode is active. The specified IDs can only be real shows within the calendar and they must not contain track IDs at the end.
GlobalJingles	The name of the jingle group (case insensitive).
ServiceJingles ProgramJingles	The name of the jingle group (case insensitive, use either "ServiceJingles" or "ProgramJingles, as you like).
Jingles	The name of the jingle group (case insensitive). This loads either a global or a service-specific jingle group according to the current service settings.
PreProductions	The name of the preproduction node (case insensitive).
GeneralPool	The name of the general pool (case insensitive).
PersonalPool	<Username>/<Name of personal pool> (case insensitive). You can use a star "*" for the username which means: the current DigaSystem user.
DayPool	Either a real date like "2001-12-24" or "24.12.2001" or an offset: 0=today, 1=tomorrow, -1=yesterday…
Show	<Date>/<Show>/<Track> <Date> is either a real date like "2001-12-24" or "24.12.2001" or an offset: 0=today, 1=tomorrow, -1=yesterday… <Show> is either a show name or the start time of the show like "12:00:00" or an offset: 0=current show (meaning: now, not the main loaded show), 1=next show, 01=current show, if not existing use next show, -1=previous show… Pay attention if you specify 2 offsets, one for the date and one for the show. If you specify a show offset, the date does no longer matter! TurboPlayer always uses the show offset from “now”. <Track> is either the track number like "1" or the special word "Pool" to load the show pool or you leave out the parameter to load the track which is currently assigned to the rundown (=list mode: track). Since version 5.2 you can also specify one of the keywords “Active” or “Main” for <Date>. In this case a specified show offset refers to the show in the show rundown which is currently active. (The one which is displayed by the GUI in the show selection combo box.) This might be a different one than the show defined for “now”. If you specify “Active” you can also use a show start time or a show name as second value. In this case the lookup for the show is done in the same day as the current active show is located.

Comments: Pay attention to enclose the details parameter in quotation marks. Otherwise the macro processor will try to resolve something like "0/0/Pool" or an ID as a numerical division.
Loading a node into a stack is only possible if the stack is in stack mode "free". TurboPlayer will automatically change the stack mode before it continues with the loading of the node.
The show and jingle rundown do not automatically change their "stack mode". For the show rundown you have to set the stack mode and the free-showlist mode before calling TP_LoadNode, otherwise the desired node might not be loadable. Besides, be careful if you allow loading normal show sequences and other node types into the show rundown. It might confuse the user, e.g. when shows are no longer automatically advanced. For the Jingles rundown it is only allowed to load a jingles node.
You can use “ID” together with a real show node (=a show within a day / the calendar) to perform a full show loading together with a TurboPlayer reset. The ID must be the ID of a show without any track ID at the end. In this utilization, the stack mode of the show rundown must be “track” (as it is by default).
If you specify a track number when using the type "Show" TurboPlayer will automatically change the track number which is currently assigned to the rundown (either by registry parameter or by a previous call of TP_ToggleListMode.
Hint: for show pools TurboPlayer uses the virtual track number 10000 internally. This number might appear on the GUI and you could use it as track number in the TP_ToggleListMode or TP_LoadNode command.

Executed by: Kernel

Examples:
TP_LoadNode (Stack1, Jingles, "Morning Jingles")
TP_LoadNode (Stack1, DayPool, 1)
TP_LoadNode (Stack1, Show, "0/0/Pool")
TP_LoadNode (Stack1, Show, “Active/0/Pool”)
TP_LoadNode (Stack1, Show, "2001-12-24/Morning show/1")
TP_LoadNode (Show, ID, “00000000/00000067/000007DF/0000000A/00000005/00000067”)
TP_LoadNode (Show, Show, “Active/1”)

TP_Log (…)

Prints information to the TechEx log

Parameter: … A free number of string parameters or expressions which evaluate to a number or a string.

Executed by: Kernel

Examples: TP_Log ("Channel ", %TheChannel, " is not free.")

TP_MoveFader (Line, Time, Level, TreatAsUserAction)

Moves a fader

Parameter:

Line	Either the line number or a known mixer source name
Time	The fade time in milliseconds.
Level	The end level in dB (the fade starts at the current level). Levels <= -99 dB are interpreted as "close fader".
TreatAsUserAction	Optional parameter. As standard behaviour the command is seen as initiated by the engine. That means no start/stop commands are executed if the fader is opened/closed. If you set the parameter to Yes/TRUE/1 the command will be treated as if the user had opened/closed the fader and start/stops will be performed.

Executed by: Kernel

Comment: This command should only be used for lines/faders/mixer sources which cannot be controlled with a <TP_Fade> command.

Examples: TP_MoveFader (Mic1, 100, 0)

TP_Recall (Rundown, OptionString)

Reactivates sent/skipped elements in a rundown.

Parameter: Rundown — One of the rundowns: Show, Jingles, Stack1…Stack15

OptionString — Optional string parameter with a list of keywords. If a keyword is present the corresponding behaviour changes. Possible keywords are:

- LeaveUserSkipped: the command leaves elements which have been skipped by the user untouched and does not recall them.
- LeaveTPSkipped: the command leaves elements which have been skipped by TurboPlayer during playout untouched and does not recall them.
- EndpointStoryIndividually: children of an EndpointStory are recalled one-by-one. Default is to recall them all together.

Executed by: Kernel

Comment: This command is intended to reactivate elements in the rundown, which have been played by accident or for rehearsal only. TurboPlayer will search for the last played or skipped element in the specified rundown and will set it to send state “Planned”. Any other on-air fields and the internal flag which marks an element as played are reset, too.

This command can only be used to reactivate elements bottom-up. It is not possible to reactivate an element in the middle of a sent range. If the bottom-most found element is just playing, the command is not executed and an error message is displayed to the user (if the command was triggered by a certain GUI).

Please note that the first playout of elements might have changed an internal state, which not always can fully be reset. For example, if the user only reactivates a part of a transition or of a group, the times being displayed for this transition / group will be wrong or even missing. Nevertheless, controls assigned to reactivated elements or to a group in which elements are being reactivated are re-executed and an automatically assigned overlaid flag is removed during reactivation.

Please note: there are multiple concepts to address this topic in general: minimum runtime, macro command TP_SetNext and rehearsal mode.

Examples: TP_Recall ("Show", "LeaveUserSkipped")

TP_ResetPlayingEndpointStory (Rundown)

Parameter: Rundown — Only one rundown is supported so far: Show:

This command can be executed if for some reason the play state of an EndpointStory - or better: the state “PlayingEndpointStory” of a rundown – is not reset automatically by TurboPlayer. As a result, the audio routing might be wrong (still a special routing for regionalization might be active). You can call this command to extraordinarily reset the play state. This will cause the internal event “EndpointStoryEnd” to be triggered and – if configured – can end the wrong audio routing.

Executed by: Kernel

TP_ResetPrelisten ( )

This command can be executed if some internal state related to prelisten or PFL becomes wrong. This should not happen but if it does further prelistening might be impossible. This command is some sort of “panic” command to stop any running prelisten / PFL and to reset the internal prelisten states. With some luck prelisten will work afterwards.

Executed by: Kernel + GUI

TP_ResetTransition (Type, Value, Flags,…)

Resets a whole transition with a single command.

A transition here means: two main elements (typically music) and one or multiple transition elements (with the flag Time_StartOnTransitionChannel set). This command will either skip or delete all transition elements. The end fadings of the first main element and the begin fadings of the second main element will be reset. This means: the mark-, link- and fade values will be reset to either the default values (if they are present in the metadata) or to 0 / take duration otherwise. If a fade curve is present, the end / begin part of it will be replaced by simple fades as they are given by the default fade values. If no default fade values are present, the result will be a rectangular fade curve.

With the two parameters Type and Value you specify a single reference element. Normally you would use something like a cursor element or a <next> element of a rundown. The keywords Line and Channel are only present for completeness. The optional Flags can be listed as comma separated additional parameters. The order does not matter, only the presence of a specific keyword.

Parameter:	Type	The rundown list type, one of: "Show", "Jingles", "Stack1".."Stack15", "TheRundown" or: "Line", "Channel". If you use Line or Channel there must be exactly one playing/paused element.
	Value	If Type is a rundown, it is the the full node ID or the kernel rundown position as number (starting with 1) or one of the keywords: "Next", "Cursor<n>", "LastPlaying", "TheElement", If you use one of the cursor keywords <n> denotes the GUI number (starting with 1). If Type is "Line" or "Channel" it is the line / channel number.
	Flags	The following keywords can be used for the flags: - Delete or DelTransition: The transition elements are deleted, not skipped. - Prev, Previous or PrevTransition: The transition before the referenced main element is reset. - Next or NextTransition: The transition after the referenced main element is reset. - Main or RefMain: The command works only if a main element is referenced but nor for a transition element. - RefTransition: The command works only if a transition element is referenced but not for a main element.
Executed by:	Kernel + TreeManager	
Comment:	The keywords “Previous” and “Next” are only relevant if the specified/referenced element is a main element. If it is a transition element it is clear which transition is being meant without one of these flags. If you do not specify one of these two flags, the default is “Next” if the element was specified by “LastPlaying” or by a line or channel specification. In all other cases the default is “Previous”. The keywords “Main” or “Transition” are useful to avoid that a user calls this command if the cursor was set on an element with the wrong type. The macro works for transitions without transition elements, too. In this case the referenced element is one of the two main elements and the fade values of these two elements are reset. In this case the keyword “RefTransition” must not have been set in the macro, of course, because the user cannot select a transition element. So far, this command works only if you have at least one transition line defined (parameter “UseForTransitions”). Without such a line TurboPlayer does not build up information about the elements belonging to transitions.	
Examples:	TP_ResetTransition (Show, Cursor1, RefMain, PrevTransition, Delete)	

TP_SelectNextElement (Type, Position)

Sets the next pointer for a rundown.

Parameter:	Type	One of the rundowns: Show, Jingles, Stack1…Stack15
	Position	The rundown kernel position = the index within the list of preloaded elements. If the index is too small the first element is selected, if the index is too high, the last element is selected. You can also use the special keywords: "first", "last", "next", "prev".
Executed by:	Kernel	
Comment:	This function is different from <TP_SetNext> which is executed by the GUI and moves or copies an element to the current next position. This function does not change the rundown. It simply positions the <next> pointer to one of the elements known to the kernel (aka the preloaded elements). The index starts with 1. Note: If rundown mode is 'sequenced' the <next> pointer cannot be changed with this macro, use TP_SetNext() instead.	
Examples:	TP_SelectNextElement (Show, next)	

TP_SelectMixerSource (Name, Select, Line)

Selects or deselects a mixer source.

Parameter:	Name	Mixer source name (must be a registry key below "MixerSources")
	Select	Can be Select (or 1), Deselect (or 0) or Toggle
	Line	Line number, optional parameter, only for simulated selection (see below)
Executed by:	Kernel	
Comment:	If a select for an already selected source is requested, or a deselect for a not selected source, the command is ignored. Normally the command is sent to the mixer console which performs the assignment of the source to a line. Therefore no line number must be specified. In case you use a mixer console or an interface which cannot handle mixer sources (e.g. GPIO) you can use mixer sources nevertheless (-> "simulated mixer sources" in concepts). You can assign known mixer sources to lines either permanently via the registry (-> TurboPlayer\Lines\...\MixerSource) or with this command. In this case you have to specify the line parameter. TurboPlayer cannot check whether the specified line number is correct, it simply stores the new assignment. Mixer sources assigned by this command will be lost with a reset.	
Examples:	TP_SelectMixerSource (Mic1, Select)	

TP_SendCustomGUIMessage (GUIs, CustomMessage)

Sends an arbitrary message string to the specified GUIs.

Parameter:	GUIs	The GUIs which should receive the custom message. You can specify a star "*" which means: all connected GUIs, or you can list GUI numbers as a comma-separated string, including ranges. Example: "1,2,5-10". Be aware that the message is only sent to GUIs which are actively connected at the moment when the command is being executed.
	CustomMessage	Can be any string. This string is not interpreted by the kernel, it must be interpreted by the receiving GUIs. The standard GUI does not evaluate these custom messages, they are intended for web GUIs connected via TurboPlayerService.
Executed by:	Kernel	
Comment:	Pay attention to enclose the GUI string into quotation marks in order to avoid that the macro interpreter splits it into multiple parameters or an arithmetic expression. (Exception: a single star or a single number.)	
Examples:	TP_SendCustomGUIMessage ("1-4", "<GUIInfo>ABC</GUIInfo>")	

TP_SetAllowedStartComputer (Command, Value)

Modifies the internal list of remote computers which are allowed to execute a relative or sync start.

Parameter:	Command	Can be one of "Set", "Add" or "Del".
	Value	If command is "Set", value can be an empty string to empty the list, a computer name to set a single computer into the list, a comma-separated list of computers (see example for the correct syntax) to set the complete list or "*" to allow all computers. If command is "Add", value can be a single computer name which is added to the list (if not already present). If command is "Del", value can be a single computer name which is removed from the list (if present).
Executed by:	Kernel	
Comment:	The computer name to specify is the Windows computer name. The standard content of the list is a "*" which means: "all computers". If you do not change the list (which is only possible with this command) the start reports of all computers are evaluated. You can change the list to contain certain computers only to achieve that starts on other computers do not trigger a start. Pay attention not to deactivate the own computer. A deactivation might be useful in a very special situation, but in general relative and sync starts should work for the own computer.	
Examples:	TP_SetAllowedStartComputer (Set, "Turbo2,Turbo3")	

TP_SetControlOfElement (Type, Value, Action, Key, Reference, Offset, Parameter)

Special version of "SetDataOfElement" for controls (=generic events, timed during the runtime of an element)

Parameter:

Type	The rundown list type, one of: "Show", "Jingles", "Stack1".."Stack15", "TheRundown" or: "Line", "Channel". If you use Line or Channel there must be exactly one playing/paused element.
Value	If Type is "Line" or "Channel" it is the line / channel number. If Type is a rundown, it is the the full node ID or the kernel rundown position as number (starting with 1) or one of the keywords: "Next", "Cursor<n>", "LastPlaying", "TheElement", "GroupOfNext", "StoryOfNext", "GroupOfCursor<n>", "StoryOfCursor<n>", "GroupOfLastPlaying", "StoryOfLastPlaying", "TheGroupOfTheElement", "TheStoryOfTheElement", “EndpointStoryOfNext”, “EndpointStoryOfCursor”, “EndpointStoryOfLastPlaying”, “TheEndpointStoryOfTheElement”. If you use one of the cursor keywords <n> denotes the GUI number (starting with 1).
Action	Can be: – "Set". Sets the control at the specified position. – "Remove" or "Reset". Removes the control at the specified position. "Key" can be a star "*", which will remove all existing controls of the element. "Reference" can be a star "*", which will remove all controls with the specified key. – "Toggle". Sets the control at the specified position if it is not present for the element and removes it otherwise.
Key	The control key: one of the registry keys below TurboPlayer \ EventsOut
Reference	The reference point, one of: MarkIn, MarkOut, Stop, LinkIn, LinkOut, Intro, Outro Can be omitted for fully specified events or can be a star (see above).
Offset	Optional: the offset from the reference point in milliseconds. Can be omitted for fully specified events.
Parameter	Optional: the parameter for the control. Not used so far: do not specify.

Comment: The registry parameter "Exclusive" of events is automatically executed.
If you use one of the special keywords for the rundown list or the element position, be sure that the corresponding rundown/element is known, otherwise macro processing will be terminated with an error.
The specifications "LastPlaying", "GroupOfLastPlaying" and "StoryOfLastPlaying" look for really playing elements first. If no such element is found, the last element which are just starting is used. This might be important if the function is used within an internal event of type "ElementStarting".
If the global parameter "KernelHasFullData" is not set, the kernel can handle only elements which are known to the kernel, which are only the preloaded elements !
If the parameter is set, other elements and groups can be handled, too. This also means: the group-keywords can only be used if the parameter is set.

Executed by: Kernel

Example:
```
TP_SetControlOfElement ( Show, Cursor1, Set, ARIOn, MarkIn, 0 )
TP_SetControlOfElement ( Jingles,
                         “00000000/00000067/00000004/00000070/0000006B”,
                          Toggle, ARIOn )
```

TP_SetDataOfElement (Type, Value, Field1, Data1, Field2, Data2, Field3, Data3, …)

Sets a field (=XML tagname) of an element

Parameter:	Type	The rundown list type, one of: "Show", "Jingles", "Stack1".."Stack15", "TheRundown" or: "Line", "Channel". If you use Line or Channel there must be exactly one playing/paused element.
	Value	If Type is "Line" or "Channel" it is the line / channel number. If Type is a rundown, it is the full node ID or the kernel rundown position as number (starting with 1) or one of the keywords: "Next", "Cursor<n>", "LastPlaying", "TheElement", "GroupOfNext", "StoryOfNext", "GroupOfCursor<n>", "StoryOfCursor<n>", "GroupOfLastPlaying", "StoryOfLastPlaying", "TheGroupOfTheElement", "TheStoryOfTheElement", “EndpointStoryOfNext”, “EndpointStoryOfCursor<n>”, “EndpointStoryOfLastPlaying”, “TheEndpointStoryOfTheElement”. If you use one of the cursor keywords <n> denotes the GUI number (starting with 1). You can also use the keyword "All" to update all elements and groups of a rundown.
	FieldX	The field to be set. See the BCS technical manual for the correct XML tagnames. If you specify non-standard fields the fieldname must be a valid XML tagname. Limit yourself to letters, numbers, underscore and dots.
	DataX	The data to be set. This can be an empty string ("") to clear a field.
Comment:		You can set multiple fields with a single command. Simply list all field names and the new value as pairs of parameters. If you use one of the special keywords for the rundown list or the element position, be sure that the corresponding rundown/element is known, otherwise macro processing will be terminated with an error. The specifications "LastPlaying", "GroupOfLastPlaying" and "StoryOfLastPlaying" look for really playing elements first. If no such element is found, the last element which are just starting is used. This might be important if the function is used within an internal event of type "ElementStarting". If the global parameter "KernelHasFullData" is not set, the kernel can handle only elements which are known to the kernel, which are only the preloaded elements ! If the parameter is set, other elements and groups can be handled, too. This also means: the group-keywords can only be used if the parameter is set.
Executed by:		Kernel
Examples:		TP_SetDataOfElement (Show, Cursor1, Time_StartMode, Manual) TP_SetDataOfElement (Show, All, SendState, "Planned", Time_RealStart, “”, Time_RealStop, "", Time_RealDuration, "") TP_SetDataOfElement (Jingles, “00000000/00000067/00000004/00000070/0000006B”, “Title”, “This is the new title”)

TP_SetLineParameter (Line, Parameter, Value)

Toggles internal modes

Parameter:	Line	The line number.
	Parameter	One of: LineMode, AutoLineMode, CurrentLineMode
	Value	One of: None, Passive, Active, Active/Variable, Active/OnOff.
Comment:		See the chapter Concepts / line modes above for an explanation of the modes. Pay attention: TurboPlayer holds two parameters for the line mode: one is valid in Automatic activation (=AutoLineMode) and the other one is valid for Passive + LiveAssist activation (=LineMode). Be careful to change the right parameter. CurrentLineMode changes the parameter for the current activation. Do not forget to enclose parameters like "Active/Variable" in quotation marks. Otherwise the parser will see a division.
Executed by:		Kernel / TreeManager
Examples:		TP_SetLineParameter (1, CurrentLineMode, "Active/OnOff")

TP_SetParameter (Name, Value)

Assigns a value to a TurboPlayer parameter.

Parameter: Name — Parameter name

Value — The value assigned to the variable. This can be another expression.

With this command some of the parameters which are set via registry normally can be changed during runtime. You can use the same format for <Value> as in the registry. The following parameters are supported so far:

- BeatExactFadeOut (-> TurboPlayer\Stinger\BeatExactFadeOut)
- BeatExactDuckingTime (-> TurboPlayer\Stinger\BeatExactDuckingTime)
- BeatExactDuckingLevel (-> TurboPlayer\Stinger\BeatExactDuckingLevel)
- SequencedOnSameLine (-> TurboPlayer\SequencedOnSameLine)

Executed by: Kernel

Examples: TP_SetParameter (BeatExactFadeOut, “300,300”)

TP_SetProgram (Program, Flags, ...)

Changes the current BCS program (=service).

Parameter: Program — Name of the program.

Flags — Optional parameters. At the moment two possible keywords are supported: "Reset" and "LoadCurrentShow".
A reset will be done before switching the program and in this case it does NOT include to disconnect from BCS. A reset is not necessary if you either switched the BCS connection beforehand or if you load the current show afterwards.
Loading the current show is done after switching the program.

Comments: This procedure is available since TurboPlayer version 6.0.
Only selecting a new program does not change the content loaded in TurboPlayer, which is an undesired state. The loaded content will still be from the old program, whileas the program selection box in the GUI will already display the new program. In order to not confuse the user, you should always load a new content (e.g. with the flag "LoadCurrentShow") or by calling TP_LoadNode afterwards. Alternatively, you should specify the Reset flag in order to end up with an empty TurboPlayer.
This command cannot be executed if the kernel has loaded at least one single element, which is not in state "stopped". Typically, it can happen that some elements are playing and switching the program is not possible. If this happens, macro execution is stopped with an error report in the log. Therefore it is recommended to first check, whether there is no playing or paused element, e.g. with the check:
DataOfChannel (CountInState, Playing, Paused) == 0
Hint: If an element is only prepared to a channel, it does not prevent this command.

Executed by: Kernel

Examples: TP_SetProgram ("P4", LoadCurrentShow)

TP_SetSubMixerSource (Name, SubName)

Sets a sub mixer source

Parameter: Name — Mixer source name (registry key !)

SubName — One of the sub mixer sources for the specfied mixer source (registry key !)

Executed by: Kernel

Comment: See the chapter Concepts / Mixer sources for more information about sub mixer sources.
Sub mixer sources are not supported for simulated mixer sources.

Examples: TP_SetSubMixerSource (MainMic, MainMicSpare)

TP_SetVariable (Name, Value)

Assigns a value to a global variable.

Parameter:	Name	Variable name
	Value	The value assigned to the variable. This can be another expression, for example: GetVariable (var2)
	This function exists only for backward compatibility for old macros. Now you can assign variables with a '=', like in: $SpecialFlag=1	
	Afterwards this variable can be used within macros. If the name is not yet known, it is added to the list of variables. If the name is known (comparison is case-insensitive) the existing variable is changed. If the value consists of digits only, the variable type is integer, otherwise it is string. Only global variables can be assigned with this function, the assignment to local variables needs a direct assignment.	
Executed by:	Kernel	
Examples:	TP_SetVariable (SpecialFlag, 1)	

TP_ShowError (GuiNo, …)

Displays information in the error window of the GUI.

Parameter:	GuiNo	The number of the GUI to display the error, or a star * for "all GUIs".
	…	A free number of string parameters or expressions which evaluate to a number or a string.
Executed by:	Kernel	
Examples:	TP_ShowError (*, "Next element cannot be started on channel 8.")	

TP_Start (Type, Value)

Starts an element

Parameter:	Type	One of: Show, Jingles, Stack1…Stack15, Line, Channel
	Value	For rundown list types it is either the rundown kernel position (starting with 1) or one of the keywords „Next“ or "NextSystem". For lines and channels it is the number (starting with 1)
Executed by:	Kernel	
Comment:	"Next" starts the element in the rundown which is marked as <next element>.	
	"NextSystem" starts the first element at or below the <next element> which is an internal (=system) element. This keyword is only available for sequenced rundowns.	
Examples:	TP_Start (Show, Next); TP_Start (Line, 4)	

TP_StartApp (CommandLine, Verb, Flags)

Starts a new Windows application.

Parameter:

CommandLine	The command line to execute. In case a verb is specified the pure filename. You should always enclose the CommandLine parameter with the escape sequences \{{ \}} This is recommended to be able to use spaces and quotation marks within this parameter.
Verb	If the verb is empty or not specified the command line is executed. If a verb is specified CommandLine should be a simple filename and the verb is performed on the file (A verb is one of the possible actions which are registered for a file extension). If you specify the special verb * the default action for the file extension is executed.
Flags	An optional string with comma-separated start flags. So far these flags are supported: - NoWindow: The window will not be visible. - MinWindow: The window will be minimized. - MaxWindow: The window will be maximized. - InactiveWindow: The window will not be active. All these window flags can be overridden by the started application, meaning: the application can ignore these values and set its windows state and position itself. Therefore the flags might not work at all and are of limited use. For console applications either "NoWindow" or no windows flag at all can be used. With verbs typically InactiveWindow will not work.

Executed by: Kernel / AppStarter

Comment: There is an extra thread (AppStarter) which starts the new application. Therefore this command does not slow down the kernel in any way. On the other hand, as only a single starter thread exists, multiple requests to start an application will be executed serialized.

Examples:
TP_StartApp (\{{ "C:\Program files\Digas\MyApp.exe" -p1 -p2 -p3 \}}, "", "NoWindow")
TP_StartApp ("C:\Temp\MyDocument.htm", "*")

TP_StartBeatExact (Type, Value)

Starts an element with beat-exact transition, if possible.

Parameter:

Type	One of: Show, Jingles, Stack1…Stack15, Line, Channel
Value	For rundown list types it is either the rundown kernel position (starting with 1) or the keyword „Next“. For lines and channels it is the number (starting with 1)

Executed by: Kernel

Comment: For an explanation see the chapter „CartBeat, CartMotion“ of the „concepts“ above.

Examples: TP_StartBeatExact (Show, Next)

TP_StartEmbedded (List, EmbeddedStartID)

Starts an element with beat-exact transition, if possible.

Parameter:
- List — One of: Show, Jingles, Stack1…Stack15 or “*”
  If you specify a rundown list, TurboPlayer looks for the element to be started in this rundown. If you specify a star, TurboPlayer will look in all existing rundowns for an element with the specified EmbeddedStartID.
- EmbeddedStartID — The EmbeddedStartID of the element to start. This command can only be used if the field with this name is filled in the element to be started.

Executed by: Kernel

Comment: This command is only available in TurboPlayer flavors “Full” and “Lite”.
This command is only executed if the general mode “EmbeddedStart” is active.
This command is intended for reusing broadcast content with live elements in a metadata track. Within this workflow TurboPlayer automatically triggers this command at the subclips which represent the start of a live element. Therefore it should normally not be necessary to use this command outside of such a reuse workflow.
This command triggers a start regardless of the start attributes of the element to be started. Nevertheless it is necessary that the element to be started is within the preload range of the corresponding rundown list. If you have many live elements on a metadata track and you seem to be limited by available MultiPlayer slots, you can use the parameter “TurboPlayer \ CheckSlots” to configure more preloaded elements than there are available MultiPlayer slots.

Examples: TP_StartEmbedded (“*”, “89CD4834-23EB-46C4-9FB9-89A9D6A7D2F8”)

TP_StartStop (Type, Value)

Starts or stops an element (function toggles play state)

Parameter:
- Type — One of: Show, Jingles, Stack1..Stack15, Line, Channel
- Value — For rundown list types it is the rundown kernel position (starting with 1) or “NextAll” (or "AllNext"), otherwise the line / channel number (starting with 1). The special keyword "NextAll" for rundown performs a start-next if no element at all is running in the specified rundown, or a stop-all if at least one element is running.

Executed by: Kernel

Examples: TP_StartStop (Jingles, 1); TP_StartStop (Channel, 1); TP_StartStop (Show, Next)

TP_Stop (Type, Value)

Stops an element

Parameter:
- Type — One of: Show, Jingles, Stack1…Stack15, Line, Channel
- Value — For rundown list types it is the rundown kernel position (starting with 1) or the keyword "All". For lines and channels it is the number (starting with 1) or the keyword "All"

Executed by: Kernel

Comment: If you want to stop all running elements use TP_Stop (Channel, All) and not "Line", because running live elements do not have a line on which they are running – only a virtual channel.

Examples: TP_Stop (Jingles, 5); TP_Stop (Jingles, All); TP_Stop (Line, 1)

TP_StopGroup (Rundown, Time)

Performs a stop (or a fadeout) for all running elements within the group of the last playing element in the specified rundown list. Elements within a stopped group which have not yet been played are skipped.

Parameter:
- Rundown — One of: Show, Stack1..Stack15
- Time — The fadeout time in milliseconds. This parameter is optional, default is 0.

Comment: If there are multiple running groups, the eldest one is stopped. If the last running element is not within a group, it is simply stopped or faded out. If there is no running element, the active group is stopped (that is the last group in which an element was started). If there is no active group, the "next element" is used instead – if it is within a group.

Executed by: Kernel, TreeManager

Examples: TP_StopGroup (Show, 1000)

TP_ToggleListMode (List, Type, Value)

Toggles internal modes

Parameter:	List	One of: Show, Jingles, Stack1..Stack15
	Type	One of: AllowRearrangement, AutoPrepare, AutoDelete, ClearOnReset, PreloadAroundCursor, Rundown, SingleElement, Stack, Track
	Value	For "RundownMode" it can be: Random, Sequenced, Once. For "StackMode" it can be: Clipboard, Free, Track, Endpoints. For "Track" it is the track number For all other modes it can be "On" or "Off"
Comment:	See the chapter Concepts / list modes above for an explanation of the modes. The stack mode can only be changed for stacks. (Changing the stack mode for the show rundown is possible, too, but be careful when doing it. The behaviour of TurboPlayer might no longer be what you might expect.) Some mode changes are only possible, if they have not been disabled in the registry for the corresponding list. The mode "PreloadAroundCursor" can only be switched on or off. It is not possible to change the number of preloaded elements in front of / behind the cursor element as it is defined in the registry. Pay attention when changing the track number for a rundown, especially for the show rundown. The current track number of a rundown will be used implicitly in operations to load the rundown, e.g. if the user selects a new main show with the dropdown field in the GUI, the actually loaded track might not be what the user expects. Changing the stack mode to/from “Endpoints” should be avoided. If you really need to do it, you should perform a rest by loading a new main show afterwards. Otherwise, the content might not be filled correctly.	
Executed by:	Kernel / TreeManager	
Examples:	TP_ToggleListMode (Stack1, RundownMode, Sequenced)	

TP_ToggleMode (Type, Value, Timeout)

Toggles internal modes

Parameter:	Type	One of: Activation, AdjustLoudness, AssignOverlaid, AutoLoadJingles, BeatExact, ButtonStart, EmbeddedStart, ExtendedShowScanning, FaderStart, FaderStopsPFL, FreeShowlist, Ghost, IgnoreMarkOut, InsertFillers, OnAir, Pause, PlayFadings, SelectSources, ServerConnection, SyncStart
	Value	For the activation mode possible values are: L(ive), A(utomatic), P(assive). For all other modes possible values are: On or 1, or: Off or 0. For the server connection possible values are: C(onnected), S(tandalone) or R(ehearse). For all modes the letter/word (T)oggle can be used which changes between on and off or cycles through the modes, respectively.
	Timeout	This parameter is optional and only used, if the activation mode changes from passive to LiveAssist or from Passive to Automatic. Such a change requires a confirmation from the BC server or from any other TurboPlayer which is active. If either the answer is received within the timeout time or the timeout elapses, the new mode is established. If timeout > 0 the active TurboPlayer will display a request message box. If timeout = 0 the new activation mode is established immediately.
Comment:	See the chapter Concepts above for an explanation of the modes.	
Executed by:	Kernel	
Examples:	TP_ToggleMode (Activation, Auto); TP_ToggleMode (Pause, On)	

TP_TreatSpecial (Rundown, Elements, Operation, Option)

Performs an operation on special elements (text, info).

Parameter: Rundown — One of: Show, Stack1..Stack15

Elements=	Operation=	Description:
Text	Stop	Stops all running text elements. Text elements must be playing as “live elements”.
Text	Sent	Sets the specified text elements to SendState=Sent.
Text	Skip	Sets the specified text elements to SendState=Skipped.
Info	Sent	Sets the specified info elements to SendState=Sent.
Info	Skip	Sets the specified info elements to SendState=Skipped.

Option — An option can be specified for the Sent/Skip operations. It denotes the bottom-most element in the rundown which will skipped / be set to sent. Possible values are:

- Next: If the <next element> is a special element it will be treated.
- PreNext: Special elements in front of the <next element> will be treated.
- NextOrPre: Meaning: “Next”, if <next element> is not special: “PreNext”.
- PreEldestPlaying: Special elements in front of the topmost playing.
- PreNewestPlaying: Special elements in front of the bottommost playing.

Default (if option is not given) is “NextOrPre”.

Comment: The “stop” operation is intended for virtually playing elements, the other operations should be used if special elements are not played.
The media type determines whether an element is text, info or something else.
TurboPlayer starts with the specified first element. Then TurboPlayer moves backward in the rundown and treats all / multiple special elements. TurboPlayer stops when the first special element with state Sending / Sent / Skipped is encountered.

Executed by: Kernel, TreeManager

Examples: TP_TreatSpecial (Show, Text, Stop), TP_TreatSpecial (Show, Info, Sent, PreNext)

TP_TriggerBUSFilecheck( )

This command sends a file-check request for all loaded rundowns (shows, jingles, additional nodes…) via generic BCS message. There must be at least one running BUS Filecheck task running in mode “Externally triggered”, which can receive the message and can start working. This task must be configured for the same service as TurboPlayer has loaded and the requested node types must be configured to be checked.
TurboPlayer does not show any success or failure message in a GUI. Reports appear only in the technical log files. Anyhow, even if the message can be delivered correctly, this does not mean that BUS will operate as intended. The user needs to monitor the file state of the elements in TurboPlayer to see a progress or a success.

Executed by: Kernel + TreeManager

TP_UpdateEvents ( )

This command is for testing purposes only. It reloads the out-, internal- and timed-events and the kernel re-reads the key shortcuts. Changes of already loaded events (e.g. a changed macro) are loaded into memory and new events are loaded, but existing events which were deleted from the registry are not removed.

Executed by: Kernel + GUI

4.5.6 Internal Events

Internal events are executed automatically by the engine in some cases. You can specify any macro which is executed if the corresponding event occurs. In the registry these events are listed below the key "TurboPlayer\EventsInternal". Two values must be present: the "Event" parameter specifies in which case the command should be executed, and the "Command" parameter specifies the macro.

Within the command string of internal events, you have access to some special state variables which are not available otherwise. E.g. the variables %TheRundown and %TheElement specify the rundown and the element which are just being treated. With this knowledge the example of a macro from above can be modified:

```
if ( %TheRundown == Show )
{
    if ( DataOfElement ( %TheRundown, %TheElement, Class ) $= "Music" )
    {
       SwitchLightOn(5)
    }
    else
    {
      SwitchLightOff(5)
    }
}
```

This macro can be entered for the internal event "NextElementChanged", which means it will be called whenever the next element cursor is set to a different element. First, the correct rundown type is checked. Next an external lamp is switched on, whenever the class of the next element is music, otherwise the lamp is switched off. Of course, there must be an IO module which is configured to handle the light on/off commands.

Especially for the event “NextElementChanged” you might wonder whether %TheElement specifies the old or the new next element: as the event is triggered after the <next> pointer was set to a different element (therefore the event is called “changed”) it is the new next element.

Internal events available:

Internal event name	Called when ?
Init	The first available event during startup. It is triggered when the RundownKernel (which executes all macros) has read its settings, started the audio engine and prepared its communication links. Other components are not available yet. Therefore, you must be very careful because only very few commands are available. Typically, you should only set variables or some internal modes like the free-showlist mode.
Startup	Called during startup of the rundown kernel but later than "Init" (about 5-7 seconds later). A lot more components are available and initialized than during "Init", but nevertheless it is not guaranteed that all parts of TurboPlayer are fully initialized when this event is triggered. Therefore, some commands may not work as intended. If you need an initialization when TurboPlayer is fully started you should use the event “Reset” and make use of a global variable to ensure that the initialization is done only once. (The variable is necessary because the reset event is called for every explicit show change.)
Reset	Called after a reset has been executed by the rundown kernel (loading a show initiates a reset)
Exit	Called before the engine begins with the termination. If such an event is defined, the engine waits approx. 300 ms until termination continues. You have this time to execute any shutdown commands and all engine components are still present and working.
GUIConnected	A GUI has connected to the engine. This is an old event type. It is triggered when a GUI calls “GetFullInfo”. This is only done for a first connect but not for a reconnect after a disturbed connection. The advantage of this event is that the GUI is really, fully connected and is ready to accept commands. This is slightly different for the other event: GUIConnStateChanged.

Internal event name	Called when ?
GUIConnStateChanged	The connection state of a GUI has changed. The possible states are: Disconnected, Disturbed and Connected. You can use the variable %TheGUI to get the GUI number and the functions IsGUIConnected (%TheGUI) or DataOfGUI (%TheGUI, ConnState) to query more information. This event is triggered when the IP connection to the message box in kernel is established or lost. It is not guaranteed that there is a connection to the message box of TreeManager at the same time and/or that the GUI is already ready to receive commands. Possibly you need to delay things you want to do.
ModeChanged	Triggered, when one of the internal modes has changed. This includes the modes of a rundown list. (See the variable %TheMode below)
ActiveTreeChanged	If a rundown list has multiple BCS nodes loaded (e.g. the show list has the current, the prev and the next show loaded) TurboPlayer has exactly one active node (here: “tree”) in each rundown. This active tree is set when a rundown list is initially loaded and it is automatically advanced when the first element in the new tree is being started. This event is triggered for each rundown. If you are only interested in changes of the active show you must filter for: %TheRundown == Show.
CursorChanged	when one of the cursors of a GUI was set to an element
EmbeddedStartHit	when the command TP_StartEmbedded is executed and finds at least one element, which can be started.
EmbeddedStartMiss	when the command TP_StartEmbedded is executed but finds not a single element, which can be started.
NextElementChanged	when an element has been declared <next> element
MetaDataOfNextElementChanged	when the metadata of the <next> element changes. Hint1: only the fields the kernel knows are watched. Hint2: as the times of elements change with every start/stop this event might occur more often than expected.
ControlChanged	At least one of the controls (events) of an element has been changed. If you want to surevey groups and not-preloaded elements, the registry flag "KernelHasFullData" must be activated.
LoadStateChanged	when an element was loaded, unloaded or load failed. External elements (with a mixer source) and live elements are virtually preloaded and generate this event, too. Be aware that this event will be triggered with a transition “Loaded” -> “Loaded” if an element is reloaded (e.g. due to a changed file list or a changed mark-in). If an element is deleted or drops out of the preload range the element is normally moved to the rundown list of type “Deleted” before the element is really unloaded by a background task.
PlayStateChanged	when an element was prepared, started, paused or stopped. This is in parts overlapping with the next 4 events, but they are more specific.
ElementStarting	when an element is being started (before start event). Pay attention not to do a lot of stuff in a macro triggered by this event because TurboPlayer will continue with the start sequence only after the macro execution has finished.
ElementStarted	when an element has been started (after start event)
ElementStopping	when an element is being stopped (before stop event, not triggered if element runs till end-of-take). Pay attention not to do a lot of stuff in a macro triggered by this event because TurboPlayer will continue with the stop sequence only after the macro execution has finished.
ElementStopped	when an element has been stopped (after stop event)
UnexpectedStart	when a spontaneous element is being inserted as a reaction to an unexpected opening of an external line. This event is triggered after a line was opened but before the spontaneous element is created or started, therefore no element data (like an ID) is available yet.

Internal event name	Called when ?
LineStateChanged	when a line was opened or closed or when the mixer source of a line was changed.
PrelistenStateChanged	when the prelisten state for an internal element is being activated or deactivated
PFLStateChanged	when the PFL state for a line is being activated or deactivated
SyncStartHit	when a sync-start is executed. The following conditions must be TRUE: - SyncStart mode is active, - a SyncStartField has been defined, - the started element has an ID in this field. This event is called once for each found / started element. Pay attention with start messages from the own computer: if necessary, deactivate sync-starts from the own computer with the command TP_SetAllowedStartComputer.
SyncStartMiss	Like before, but this event is called when no startable element is found. The event is triggered for all used rundowns without an appropriate element once.
EndpointStoryBegin	when the first element in an EndpointStory starts playing. The event is handled as pre-start operation, meaning: it is executed typically 100 ms before the real start (or the OpenDelayActive/Passive time of the line). This event can be used to perform a switching of the audio routing if the EndpointStory is used for regionalization.
EndpointStoryEnd	when elements in an EndpointStory were playing and all elements have been stopped. This includes elements running on the local TurboPlayer but also all know elements running on a remote TurboPlayer. This event can be used to perform a switching of the audio routing if the EndpointStory is used for regionalization. So far there is no complicated logic, which keeps an EndpointStory as playing when multiple elements per region are scheduled and there are gaps in between, which cause an interval, in which no element is played at all. The logic is very simple: if no element is playing the event is triggered.

4.5.7 Special state variables:

These variables are available in internal events or if an active/current object exists. For example: the variables %TheRundown and %TheElement will be defined if a macro is executed as a control which sticks to an element.

If you try to access a variable which does not exist, execution of the macro is stopped. You can test for the existence of variables with the function: varexist (%...)

Hint: in elder versions of TurboPlayer this information was only supported via special functions. This is still supported but you should use the variable names now. The name itself was not changed, e.g.: TheRundown() is now: %TheRundown

%TheRundown

Is a placeholder for the rundown which is just being treated

Return A string The name of the rundown list: Show, Jingles, Stack1..Stack15, Prelisten.

Comment: Available in all internal events for an element: CursorChanged, NextElementChanged, ControlChanged, LoadStateChanged, ElementStarting, ElementStarted, ElementStopping, ElementStopped, UnexpectedStart, PrelistenStateChanged, PFLStateChanged (in some cases), SyncStartMissed, EmbeddedStartHit, EndpointStoryBegin, EndpointStoryEnd. Besides a ModeChanged for a rundown (!) will set this variable.

Executed by: Kernel

%TheElement

Is a placeholder for the element which is just being treated

Return A number The element position within the preloaded elements (starting with 1)

Comment: Available in all internal events for an element: CursorChanged (only available for preloaded elements), NextElementChanged, ControlChanged, LoadStateChanged, ElementStarting, ElementStarted, ElementStopping, ElementStopped, PrelistenStateChanged, PFLStateChanged (in some cases), EmbeddedStartHit, EndpointStoryBegin.

Executed by: Kernel

%TheNodeId

Is a placeholder for the full node ID which is just being treated

Return A string The full node ID, e.g: 00000000/00000067/000007DF/0000000A/00000005/00000067/00000002/00000065/00000084

Comment: Available in all internal events for an element: CursorChanged (only available for preloaded elements), NextElementChanged, ControlChanged, LoadStateChanged, ElementStarting, ElementStarted, ElementStopped, PlayStateChanged, PrelistenStateChanged, PFLStateChanged (in some cases), SyncStartHit, EmbeddedStartHit, EndpointStoryBegin.

Also available for: ActiveTreeChanged (here it is the ID of the show without a track below) , CursorChanged

Executed by: Kernel

%TheLine

Is a placeholder for the line which is just being treated

Return A number The line number (starting with 1)

Comment: Available in internal start/stop events: ElementStarting, ElementStarted, ElementStopping, ElementStopped, UnexpectedStart, LineStateChanged, PrelistenStateChanged , PFLStateChanged, EndpointStoryBegin.

Executed by: Kernel

%TheChannel

Is a placeholder for the channel which is just being treated

Return A number The channel number (starting with 1)

Comment: Available in internal start/stop events: ElementStarting, ElementStarted, ElementStopping, ElementStopped, PrelistenStateChanged, PFLStateChanged, UnexpectedStart, EndpointStoryBegin , SyncStartMissed (the local channel on which a start would have been performed, or 0 if unknown).

Executed by: Kernel

%TheGUI

Is a placeholder for the GUI which just did an action

Return A number The GUI number (starting with 1)

Comment: The GUI number can be queried in the following events: GUIConnected, GUIConnStateChanged, CursorChanged.

In the following events the GUI number might possibly be available if the event was initially triggered by a specific GUI: ElementStarting, ElementStopping, NextElementChanged, PrelistenStateChanged, ModeChanged. In these cases you should check the existence of %TheGUI with the varexists function. Anyhow, the presence of %TheGUI is not always reliable. For example, the ElementStarting might be delayed if an element must be queued/joined. If MultiPlayer later starts the element the information about the GUI which initially triggered the start is not available anymore.

Executed by: Kernel

%TheMode

Is a placeholder for the mode which was just changed

Return	A string	The changed mode (Activation, ButtonStart, FaderStart, Ghost, IgnoreMarkOut, InsertFiller, OnAir, Pause, PlayFadings, SelectSources, ServerConnection, SyncStart, EmbeddedStart, AutoLoadJingles, AdjustLoudness) or one of the modes of a rundown list (AutoPrepare, AutoDelete, ClearOnReset, Rundown, Stack, Track)
Comment	Available in the internal event: ModeChanged	
Executed by:	Kernel	

%TheCause

Is a placeholder for the source which triggered the event

Return	A string	These values are defined: – Unknown (source is not known) – GUI (triggered by a GUI) – Line (source is a line command, typ. fader-open or –close) – Button (a key-shortcut) – EventIn (triggerend by Event-In) – EventOut (triggered by a generic event) – Timer (triggered by a timer) – Internal (triggered by some TurboPlayer-internal rule) – External (used for unexpectedly started elements) – Stinger (for stops triggered by a stinger)
Comment:	Available in internal events: NextElementChanged, ElementStarting, ElementStopping, ElementStopped, UnexpectedStart, ModeChanged, LineStateChanged (only open/close), PFLStateChanged	
Executed by:	Kernel	

%TheMinRuntimeExceeded

Is a placeholder for the fact whether the minimum runtime was exceeded when an element stops

Return	A number	1 if the minimum runtime was exceeded, otherwise 0.
Comment	Available only in the internal event: ElementStopped !	
Executed by:	Kernel	

4.5.8 Examples

Change a mode during startup

The general TurboPlayer modes (like AssignOverlaid) cannot be set directly by a registry parameter. If you want to change them permanently it is necessary to write a little macro which is executed during startup. Therefore you have to create a new registry key below the key “TurboPlayer\EventsInternal” and you have to set the parameter “Event” to “Startup”. The macro in parameter “Command” is very simple:

```
TP_ToggleMode ( AssignOverlaid, On )
```

Switch TurboPlayer to passive mode if the last GUI disconnects

The idea is to automatically switch TurboPlayer activation mode to “Passive” when there are no longer any GUIs being connected. This only works if you are using only extra GUIs. If you run the full TurboPlayer.exe, which has an internal GUI, this internal GUI will never disconnect unless you terminate the whole program.

In order to fullfill the wish, you need to create an internal event in a new registry key below the key “TurboPlayer\EventsInternal” and you have to set the parameter “Event” to “GUIConnStateChanged”. The macro in parameter “Command” could be:

```
if ( NumberOfConnectedGUIs() == 0 )
{
    TP_Log ( "Switching TurboPlayer to passive mode because there is no longer
              a GUI connected." )
    TP_ToggleMode ( Activation, Passive )
}
```

So far the macro would switch TurboPlayer to passive mode even if a GUI only looses the connection (the state becomes “Disturbed”) but does not disconnect intentionally. If you want disturbed connections to be counted as connected ones, you need to change the call in the first line:

```
if ( NumberOfConnectedGUIs( "Connected, Disturbed" ) == 0 )
{
   ……
}
```

Conditional start

This example demonstrates how to make a function which starts the next element from the show rundown – but only, if the element is a commercial:

```
if ( DataOfElement ( Show, Next, Class ) == Commercial )
{
    TP_Start ( Show, Next )
}
else
{
    TP_Log ( "Start not executed because next element is not a commercial" )
}
```

Start multiple jingles

This batch demonstrates how to always start jingle number 1 and 5 simultaneously. It is entered as command for the internal event <ElementStarted>.

```
if ( %TheRundown == Jingles )
{
    if ( %TheElement == 1 )
    {
        TP_Start ( Jingles, 5 )
    }
    else if ( %TheElement == 5 )
    {
        TP_Start ( Jingles, 1 )
    }
}
```

Convert console key (pulsed GPI event) to start/stop

Let's assume you have a mixer console with keys. When you press such a key, the console sends a GPI which is pulsed. Now you want to use the key for starting / stopping a specific jingle. The problem is, in the IO module you cannot directly generate a start/stop command, because the module can generate these commands only for lines/channels. Therefore we let the IO module generate a generic event instead of a start/stop whenever the key is pressed. And in the event we make a start or a stop, depending on the current play state of the jingle:

```
if ( DataOfElement ( Jingles, 3, PlayState ) $= "Play" )
{
    TP_Stop ( Jingles, 3 )
}
else
{
    TP_Start ( Jingles, 3 )
}
```

Allow starting of jingles in specific jingle groups only

This macro demonstrates how to program keys in a way that they start or stop a jingle only if one of the specified jingle groups is being loaded. You have to define one macro for each jingle position and to assign these macros to the jingle keys on the pad instead of the normal start/stop calls.

If the user has the wrong jingle tab loaded, a message is displayed. In this example you can also see the use of an escape sequence because the second parameter contains quotation marks.

```
if ( ( DataOfRundown ( Jingles, Name ) $= "Afternoon" ) ||
     ( DataOfRundown ( Jingles, Name ) $= "Morning" ) )
{
    TP_StartStop ( Jingles, 1 )
}
else
{
    TP_ShowError ( 1, \{{Jingletab "Afternoon" or "Morning" must be loaded\}} )
}
```

Open microphone fader

This batch demonstrates how to open or close a fader for a microphone with a mixer source which is not permanently selected. The first "if" checks if the mixer source has already been selected. If not, the source is selected and a wait of 250ms is done, to enable the mixer console to assign the source. The second part opens or closes the fader, depending on its current state and sets the background color for the corresponding button (if such a button has been defined).

```
if ( DataOfLine ( Mic1, OpenState ) == "" )
{
    TP_SelectMixerSource ( Mic1, Select )
    Sleep ( 250 )
}
if ( DataOfLine ( Mic1, OpenState ) $= "Close" )
{
    TP_MoveFader ( Mic1, 0, 0, Yes )
    TP_ChangeUserButtonProp ( 1, Mic1Button, ColorBkgnd, 255, 0, 0 )
}
else
{
    TP_MoveFader ( Mic1, 0, -99, Yes )
    TP_ChangeUserButtonProp ( 1, Mic1Button, ColorBkgnd, 192, 192, 192 )
}
```

Jingle ducking

We can extend the previous macro to achieve a ducking of any running jingle whenever the microphone fader is being opened. (Changing the user button properties is omitted here) Here you can see the use of a loop: we assume that channel number 4, 5 and 6 are the jingle channels and we iterate among them. With the if-statement within the loop we check whether there is an element playing on the channel. If so, we duck the channel by 10 dB.

```
if ( DataOfLine ( Mic1, OpenState ) == "" )
{
    TP_SelectMixerSource ( Mic1, Select )
    Sleep ( 250 )
}
if ( DataOfLine ( Mic1, OpenState ) $= "Close" )
{
    TP_MoveFader ( Mic1, 0, 0, Yes )
    %channel = 4
    while ( %channel <= 6 )
    {
        if ( DataOfChannel ( %channel, PlayState ) $= "Play" )
        {
            TP_MoveFader ( %channel, 2000, -10, FALSE )
        }
        %channel = %channel + 1
    }
}
else
{
    TP_MoveFader ( Mic1, 0, -99, Yes )
    %channel = 4
    while ( %channel <= 6 )
    {
        if ( DataOfChannel ( %channel, PlayState ) $= "Play" )
        {
            TP_MoveFader ( %channel, 1000, 0, FALSE )
        }
        %channel = %channel + 1
    }
}
```

Activating a second prelisten

Let's assume your mixer console supports a single prelisten channel. But if a second GUI wants to prelisten, something in the console must be switched. Therefore, we use the following macro to detect whether the GUI2 starts prelistening (which is always on channel 6 in this example). If so, we execute the command "PrelistenGUI2". This command is not known to TurboPlayer and is no defined event-out. Therefore, it is sent to all IO modules and it must be configured that one of the IO modules triggers the switch of the mixer console when it receices this command. The macro must be entered as internal event for "PrelistenStateChanged".

```
if ( TheChannel() == 6 )
{
    if ( DataOfChannel ( 6, PrelistenState ) == 1 )
    {
        PrelistenGUI2 ( On )
    }
    else
    {
        PrelistenGUI2 ( Off )
    }
}
```

Make a user button in the GUI blink

Here you can see how to change the properties of a GUI user button in a way that the button blinks. You have to enter this macro as a timed event which should be called by a periodic timer of 500ms e.g. To start or stop blinking you have to call the command TP_ActivateTimer for this event. Please note the use of a macro variable to store the current state of the button. At the very first call of the macro this variable will not be defined yet and an initial value is assigned to the variable. The macro is quite simple because it only changes the text and background color of the button.

```
if ( ! varexist ( $ButtonIsHigh ) )
{
    $ButtonIsHigh = 0
}
if ( $ButtonIsHigh )
{
    TP_ChangeUserButtonProp ( *, ButtonID, ColorBkgnd, 255, 0, 0 )
    TP_ChangeUserButtonProp ( *, ButtonID, ColorText, 0, 0, 0 )
    $ButtonIsHigh = 0
}
else
{
    TP_ChangeUserButtonProp ( *, ButtonID, ColorBkgnd, 255, 255, 255 )
    TP_ChangeUserButtonProp ( *, ButtonID, ColorText, 128, 0, 0 )
    $ButtonIsHigh = 1
}
```

Semi-automatic update of start times

In case TurboPlayer does not play any element but waits for the next start, no automatic recomputation of the permanently changing start times of future elements is done. This is not done to avoid a too high load for the BCS and the connected clients, because in BroadcastSystem all data changes must be handled by BCS and will be distributed to all interested clients. The generated load of doing updates e.g. every second is too high, normally. But let's assume you still want to have these automatic time updates in special situations. Therefore it is recommended to do it semi-automatically: the user gets a button or a shortcut to activate or deactivate the updating. The updating itself is done in a timed event. In case the user forgets to switch updating off, an internal event is triggered with every element start and deactivates the timed event.

(1) The timed event to perform the update (name it "AutoUpdateStarttimes", e.g.):

```
TP_UpdateStarttimes ( 1, Show )
```

This event also needs a parameter "ActiveAtStartup" which should be false, and a "Period". Try with a period of 1000 ms but increase the interval time if the load is too high.

(2) The generic event to activate or deactivate the timed event:

```
if ( ! IsTimerActive ( AutoUpdateStarttimes ) )
{
   TP_ActivateTimer ( AutoUpdateStarttimes, TRUE )
}
else
{
   TP_ActivateTimer ( AutoUpdateStarttimes, FALSE )
}
```

This event is made available via button or shortcut.

(3) The internal event to switch off the updates when an element starts:

```
if ( %TheRundown $= "Show" && IsTimerActive ( AutoUpdateStarttimes ) )
{
    TP_ActivateTimer ( AutoUpdateStarttimes, FALSE )
}
```

This must be executed with: "Event"="ElementStarted" and deactivates the automatic updating of start times in case the user forgot to do it himself.

Special toggle prelisten function

This macro set can be used to have a function (button) which toggles prelistening. The first call starts prelistening at the begin of the selected take. If nothing happens the macro changes to prelistening of the end of the take after 5 seconds. The same happens if the TogglePrelisten macro is called during prelistening of the begin. If the macro is called during prelisten of the end, prelistening is stopped. (= a Dalet function)

These macros illustrate some details. The big problem with this functionality is that the user can call the main macro (4 below) at any time. For example: if he executes the macro two times while the begin is being prelistened, TurboPlayer will first try to start a prelistening at the end but the second command can happen before this start was successful. Therefore, TurboPlayer can be in any intermediate state (e.g. stop is pending, unload is pending, preload for new position is pending, start is pending). It is impossible to treat all these states. Therefore, we use the following solution here: a global state variable ($PrelistenState). This variable allows to set a state as "requested" if the user wants to perform an action (stop) while another action (second start) is just pending. In addition, we use the internal events TurboPlayer calls when a channel started or the prelisten state changed. In these internal events we check the $PrelistenState variable and perform the requested action. If nothing was requested, we can simply set the state to "playing". So, the used states are:

- <empty> Prelisten is not active
- begin_pending A play command for the take begin is just being executed
- begin_playing Play command for begin of take was successful, TP is playing
- end_requested User wants to prelisten at end but the start action was delayed
- end_pending A play command for the take end is just being executed
- end_playing Play command for end of take was successful, TP is playing
- terminate_requested User wants to terminate prelistening but the exit action was delayed

Here we must use both internal events "PlayStateChanged" and "PrelistenStateChanged". This is necessary due to the logic of TurboPlayer when it restarts a running prelistening at a different position in the take. In this case the prelisten state remains active and only a stop/start of the channel is executed.

Another interesting thing is the usage of a global counter and a local variable to avoid that an unwanted change to prelisten-end is executed after the sleep. You should always be aware that during a sleep TurboPlayer continues running and other macros can be executed. Here it could happen that someone stops prelistening during the sleep, or perhaps the user even starts another prelisten during sleep. So we have to assure that we continue execution after the sleep only if we are still in the "same prelisten".

(1) An internal event of type "Reset" to initialize the used variables:

```
$PrelistenChannel = 6    // <-- enter the correct prelisten channel here
$PrelistenState   = ""
$PrelistenCounter = 0
```

(2) An internal event of type "PlayStateChanged" to set the prelisten state and perform "requested" actions:

```
if ( %TheChannel == $PrelistenChannel &&
     DataOfChannel ( $PrelistenChannel, PlayState ) == "Playing" )
{
    if ( $PrelistenState == "begin_pending" )
    {
        $PrelistenState = "begin_playing"
    }
    else if ( $PrelistenState == "end_pending" )
    {
        $PrelistenState = "end_playing"
    }
    else if ( $PrelistenState == "end_requested" )
    {
        $PrelistenState = "end_pending"
        TP_Prelisten ( 1, FastPrelisten, End )
    }
    else if ( $PrelistenState == "terminate_requested" )
    {
        TP_Prelisten ( 1, Exit, NoSave )
    }
}
```

(3) An internal event of type "PrelistenStateChanged" to set the prelisten state in other cases:

```
if ( TheChannel() == $PrelistenChannel )
{
    if ( DataOfChannel ( $PrelistenChannel, PrelistenState ) == 1 )
    {
        if ( $PrelistenState == "" )
        {
            $PrelistenState = "begin_playing"
        }
    }
    else
    {
        $PrelistenState = ""
    }
}
```

(4) An event-out "TogglePrelisten":

```
if ( DataOfChannel ( $PrelistenChannel, PrelistenState ) == 0 )
{
    // Start prelisten at begin of take:
    $PrelistenState = "begin_pending"
    TP_Prelisten ( 1, FastPrelisten, MarkIn )

    // To avoid changing to end if another prelisten
    // has been started during sleep:
    $PrelistenCounter = $PrelistenCounter + 1
    %MyCounter = $PrelistenCounter

    // Wait time to change to prelisten end 5000 ms:
    Sleep ( 5000 )

    // Call ourself to change to prelisten of take end:
    if ( DataOfChannel ( $PrelistenChannel, PrelistenState ) == 1 &&
         $PrelistenState == "begin_playing" &&
         %MyCounter == $PrelistenCounter )
    {
        TogglePrelisten ( )
    }
}
else // Prelisten is active
{
    if ( $PrelistenState == "begin_pending" )
    {
        $PrelistenState = "end_requested"
    }
    else if ( $PrelistenState == "begin_playing" )
    {
        $PrelistenState = "end_pending"
        TP_Prelisten ( 1, FastPrelisten, End )
    }
    else if ( $PrelistenState == "end_playing" )
    {
        $PrelistenState = "terminate_pending"
        TP_Prelisten ( 1, Exit, NoSave )
    }
    else
    {
        $PrelistenState = "terminate_requested"
    }
}
```

Playout takeover of a fallback TurboPlayer

In installations with two TurboPlayers the second TP can be configured as a passive fallback player which is able to continue the playout if the first player fails for some reason. As a helper for this case the macro command TP_AutoSetMarkIn exists (see above). Here we want to discuss how to write macros for a “takeover” button. If pressed on the fallback TurboPlayer, the last elements running in the show rundown and in stack 1 and 2 should be started at the correct position on the fallback TurboPlayer. In addition, a macro should be sent to the main TurboPlayer to stop it and switch it to passive activation mode in case that the main TurboPlayer is still working partially (meaning: it is not completely dead but operating with problems).

(1) The macro executed when the “takeover” button is being pressed:

```
// Call a second macro to silence the main TurboPlayer:
RemoteStopMainTurbo()

// Compute and set new mark-in for the last playing element in desired rundowns:
TP_AutoSetMarkIn ( Show,   "Sending+Sent", 1 )
TP_AutoSetMarkIn ( Stack1, "Sending+Sent", 1 )
TP_AutoSetMarkIn ( Stack2, "Sending+Sent", 1 )

// Now let’s make the fallback TurboPlayer active:
TP_ToggleMode ( Activation, Automatic )

// Time is needed to transfer the new metadata to the kernel:
Sleep ( 100 )

// Start all rundowns - but only if there is a <next> element to be started
// and if the element was shortend beforehand (visible in title):
if ( DataOfRundown ( Show, Next ) > 0 &&
     DataOfElement ( Show, Next, Title ) $= "[Modified MarkIn/Out]" )
{
    TP_Start ( Show, Next )
}
if ( DataOfRundown ( Stack1, Next ) > 0 &&
     DataOfElement ( Stack1, Next, Title ) $= "[Modified MarkIn/Out]" )
{
    TP_Start ( Stack1, Next )
}
if ( DataOfRundown ( Stack2, Next ) > 0 &&
     DataOfElement ( Stack2, Next, Title ) $= "[Modified MarkIn/Out]" )
{
    TP_Start ( Stack2, Next )
}
```

(2) The second macro (called RemoteStopMainTurbo in the registry) will be sent to the main TurboPlayer and will stop everything. The reason why we use a second macro is the first line: we want to transfer the execution of the remainder of the macro to the main TurboPlayer.

```
TransferExecution ( MAINTURBO )

// Switch main TurboPlayer to passive mode:
TP_ToggleMode ( Activation, Passive )

// Temporarily disconnect from BCS. This is done in order to achieve
// that the main TurboPlayer will not set the played elements to “sent”.
// All metadata changes of the main TurboPlayer will only be locally/temporary.
TP_ToggleMode ( ServerConnection, Rehearse )

// Now we can stop everything which is running:
TP_Stop ( Channel, All )

// After some wait time the main TurboPlayer can reconnect to BCS.
// As we did use the rehearse mode, TurboPlayer will throw its own data away
// and will fetch all rundown from BCS again.
Sleep ( 2000 )
TP_ToggleMode ( ServerConnection, Connected )

// Main TurboPlayer is now in passive (watching) mode.
// The same macros could be used vice-versa in order to switch playout
// back to main TurboPlayer.
```

Control a GUI recording for external / spontaneous elements

Per definition external elements are elements for which no audio is available during scheduling. The audio is only available during playout. Therefore, it might be desirable to record the audio in the background while a show is running. This makes the audio available for any kind of later reuse. Since version 5.2 TurboPlayerGUI can use MultiRec to make recordings onto elements. Typically, these elements are placeholders, but in this use case they are external elements. These external elements can be scheduled in advanced, or they might be created by TurboPlayer as spontaneous elements when an external fader is opened and no corresponding external elements is scheduled as <next> element. Here is a set of macros which allow to do a GUI recording with the help of the command TP_ControlGuiRecording:

(1) We need an internal event with the trigger “ElementStarting”:

```
if ( %TheLine == 7 &&
     DataOfLine ( %TheLine, OpenState ) == Open )
{
    // We must not trigger a start for spontaneous elements.
    // This is done in internal event for "UnexpectedStart".
    if ( DataOfElement ( %TheRundown, LastPlayingExternal,
                         IsSpontaneousElement ) == 0 )
    {
        TP_ControlGuiRecording ( 1, Start, Show, LastPlayingExternal, "", Yes )
    }
}
```

Here we assume that it is GUI number 1 (the first parameter of TP_ControlRecording) which performs the recording and we assume that line 7 is always used – e.g. because it is a fixed microphone line. If the line number can be variable, you could instead use something like this query to find the correct mixer source:

```
if ( DataOfLine ( %TheLine, MixerSourceName ) == "MainMic" …
```

(2) In order to handle the case of a spontaneous element we need an extra internal event. This is necessary because “ElementStarting” is being triggered much later (after BCS has inserted the spontaneous element into the rundown). In order to not lose the beginning of the audio we need to start the recording as early as possible. Therefore, we use the event “UnexpectedStart” for the next macro:

```
if ( %TheLine == 7 &&
     DataOfLine ( %TheLine, OpenState ) == Open )
{
    TP_ControlGuiRecording ( 1, Start, Show, LastPlayingExternal, "", Yes )
}
```

The second if-condition in macro (1) assures that no second start is triggered when the spontaneous element has been inserted into the rundown and TurboPlayer begins to play the element virtually.

(3) Last, we need a macro to stop recording when the external line is closed. This can be done with a single macro for spontaneous and scheduled elements in an internal event with trigger “ElementStopped”.

```
if ( %TheLine == 7 &&
     DataOfLine ( %TheLine, OpenState ) == Closed &&
     %TheMinRuntimeExceeded )
{
    TP_ControlGuiRecording ( 1, StopSave )
}
```

In case a spontaneous element is stopped after a very short time (this is independent of a configured minimum runtime) TurboPlayer removes the spontaneous element from the rundown. In case a scheduled external element is played for less than the configured minimum runtime TurboPlayer resets the element to planned. In both cases the GUI recording will be stopped automatically with a second parameter “StopCancel”. These cases need to be treated by the macro and therefore we have the query of the %TheMinRuntimeExceeded state variable.

Displaying a warning if elements cannot be loaded

This is an example for using defined macro functions and the command TP_ForNodesDo, which were both introduced in TurboPlayer version 6.

Normally, the user can see the load state of the elements in the preload range by the background color of these elements in the rundown view and an additional flash symbol. But if you have a big preload range only a small fraction of this range might be visible. For this example, we want to make sure that the user is informed if one of the elements in the preload range could not be loaded, though the corresponding element is not visible. Therefore, we want to place a user button on a prominent place in the GUI. Clicking the "button" should not trigger any action, but the text and the background color should clearly signal the overall load status.

The idea for this example is to execute a macro regularly as a timed event (e.g. every second or every few seconds) and to use the command TP_ForNodesDo to iterate over the whole preload range in order to detect elements, which couldn't be loaded. (A hint: this is only one example to solve this task. Another possibility would be to evalute the load state in an internal event of type "LoadStateChanged" and to track the overall load state in global variables.) Here is the macro (assuming that a user button in the GUI(s) with button ID "WarnButton" does exist:

```
// We must make sure that the global variable loadFailed does exist. (Alternatively,
// the variable could be initialized in an internal Reset/Startup macro.)
if ( ! varexist ( $loadFailed ) )
{
    $loadFailed = "false"
}

// Remember the old state, whether a load had failed the last time:
%loadFailedOld = $loadFailed

function CheckLoadState ( rundown, index, ID )
{
    if ( DataOfElement ( %rundown, %index, "LoadState" ) $= "Load failed" )
    {
        // We cannot use the return value for the result of the evaluation.
        // Therefore, we use a global variable:
        $loadFailed = "true"
        return false
    }
    return true
}

// Only look for load-failed elements if something is being loaded:
if ( DataOfRundown ( Show, KernelElements ) > 0 )
{
    // Check all elements in preload range:
    $loadFailed = "false"
    TP_ForNodesDo ( Show, first, last, down, false, CheckLoadState )
}
else
{
    $loadFailed = "false"
}

// Only if the status changed, call the function to display
// the new status in a user button of the GUI(s):
if ( $loadFailed != %loadFailedOld )
{
    if ( bool ( $loadFailed ) )
    {
        TP_ChangeUserButtonProp ( *, "WarnButton", "ColorBkgnd", 255, 0, 0 )
        TP_ChangeUserButtonProp (*, "WarnButton", "Text", "ATTENTION: Load failed!")
    }
    else
    {
        TP_ChangeUserButtonProp ( *, "WarnButton", "ColorBkgnd", 128, 128, 128 )
        TP_ChangeUserButtonProp ( *, "WarnButton", "Text", "Load state o.k." )
    }
}
```

Bulk skipping of spare elements in overplanned shows

This is an example for using defined macro functions and the command TP_ForNodesDo, which were both introduced in TurboPlayer version 6.

Let's assume that we are always planning too many elements into each show, because there should be enough spare material available for unexpected situations. By definition, most of these extra elements will not be used in a regular situation. Now we want to give the user who runs the show the possibility to skip all these spare elements with one mouse click when reaching the end of the show. Therefore, we want to have a macro, which can identify the spare elements at the end of the currently active show and skip them. For this task the command TP_ForNodesDo comes in handy. It allows to iterate over all nodes of the active show. Then we can write a callback function for this command, which identifies the spare elements and skips them. Identifying the spare elements is done by looking to the start timestamp of each element: if the scheduled start of the element is after the end of the show, the element is not needed, obviously. Here is a macro, which handles the task:

```
function SkipSpareElements ( rundown, index, ID, showEndDate, showEndTime )
{
    // Only skip elements with send state "Planned" or "Cleared"
    // (e.g. not already running ones):
    %sendState = Data ( "SendState" )
    if ( Tagname() == "Element" &&
         ( %sendState == "Planned" ||
           %sendState == "Cleared" ) )
    {
        // Compute start date and time as numbers from start timestamp for cur. node
        // Format of a timestamp is "YYYY-MM-DD HH:MM:SS.mmm"
        %start = Data ( "Time_Start" )
        %startDate = ConvertDate ( substr ( %start, 0, 10 ) )
        %startTime = ConvertTime ( substr ( %start, 11, 12 ) )

        // Now skip all elements, which have a start timestamp after
        // the end of the active show:
        if ( %startDate > %showEndDate ||
             ( %startDate == %showEndDate && %startTime >= %showEndTime ) )
        {
            TP_SetDataOfElement ( %rundown, %ID, "SendState", "Skipped" )
        }
    }
    return true
}

// Compute end date and end time of active show as numbers:
%endOfActiveShow = DataOfShow ( "Show", "ActiveShow", "", "Time_Stop" )
%showEndDate = ConvertDate ( substr ( %endOfActiveShow, 0, 10 ) )
%showEndTime = ConvertTime ( substr ( %endOfActiveShow, 11, 12 ) )

// Iterate over all nodes of the active show and call function SkipSpareElements:
TP_ForNodesDo ( Show, FirstInActiveTree, LastInActiveTree, Down,
                FullTree, SkipSpareElements, %showEndDate, %showEndTime )
```

5 REGIONALIZATION AND REMOTING

This chapter is only a supplement to the chapter with same title in BCSTechManual. Please read the chapter in the BCS manual first to learn something about the general ideas and the terms used. Here we only discuss additional topics concerning TurboPlayer.

5.1 Multiple rundowns

5.1.1 Tracks and stacks

In DigaBroadcastSystem regional elements are stored in separate tracks. TurboPlayer supports playing multiple tracks at the same time with the help of stacks. Stacks are simply multi-purpose rundown lists in addition to the main show rundown and the jingle list. In principle there is no big difference between each of the rundown lists, how a rundown behaves depends on the configuration. Here we focus on stacks which are configured as rundowns for additional tracks of the loaded shows.

The configuration is done in the DigaSystem registry below the key:
`TurboPlayer\RundownLists`
You can create subkeys called "Stack1", "Stack2"… Below these subkeys are the parameters for the stacks. Please see the chapters 1.4.6 and 1.4.7 as well, they describe the rundown lists and the possible settings in general. Here we specify the following settings:

- `StackMode=Track` This setting tells TurboPlayer to fill the list with the specified tracks of the loaded shows automatically.
- `Track=<Track number>` This is the assignment of the stack to a track.
- `RundownMode=Sequenced` This parameter is needed to make the stack behave like the show rundown, meaning: the list is played from top to bottom, and each element can be played only once.

In addition, you must assign logical channels to the stacks to be able to play elements, and if you want to see the stacks in the GUI, you must configure stack windows in the surface. In this configuration TurboPlayer will automatically load all specified tracks for the loaded shows and distribute them to their assigned stack rundown. Note that the additional tracks do not necessarily follow the main track while it advances. That means if all elements of the main track have been played, the current show changes. At the same time there might still be planned elements in one of the additional tracks and such a track will not advance together with the main track. It can be enforced that the additional tracks are always from the same shows as the main track by setting the parameter:
`RundownLists\TracksFollowShow=1`
With this setting it no longer matters if there are still planned elements in one of the tracks. The current show will be advanced for all tracks simultaneously.

So far, this description applies to regional playout from one studio as well as for remoting. Especially if you assign logical channels to all rundowns, you are probably going to play the elements from a single location. But now let's have a look at remoting.

5.1.2 Remote links

Here we have the case that some of the loaded shows – or more interesting: some of the tracks – have a remote link set. That means, the data TurboPlayer loads, is hosted on a remote system. This might be a problem, of course. First of all, TurboPlayer must have write access. Otherwise, it cannot play the elements correctly. Secondly, it is not possible for multiple TurboPlayers to play the same elements, though this sometimes is desirable. And thirdly, the delay between TurboPlayer and the remote BCS could be so long that the whole system becomes very slow. Mostly this is a visual problem, because the elements in the rundown list are updated slowly. Time critical commands like start and stop should not be affected, because they are handled completely within TurboPlayer. But sometimes (e.g. the start of an unexpected external element) TurboPlayer depends on the answer of the BCS and therefore there can be no guarantee that the delay is short enough for a correct operation. If you want to use such a remote configuration, you must evaluate and test your whole system whether it is fast and stable enough or not.

To solve some of the problems there is a feature called de-linking. To configure the behaviour of TurboPlayer the parameter:

`TurboPlayer\RemotePlayMode`

exists. Here are the possible settings:

- "No": starting elements in remote shows / tracks is not possible, an error message is displayed.
- "Yes": the remote link is ignored, the element is played like a local element.
- "Delink silent": when the first element of a remote show / track is started, TurboPlayer automatically removes the remote link. The show / track will be played from the local server and changes are no longer reflected from / to the remote system.
- "Delink ask": a message box appears, telling the user about the remote link and asking, whether he wants to remove the link. The user can either remove the link, or starting elements is not possible.

Please note: the problems above do not exist, if you use the stacks read-only. For example, let's assume we are the central system and there are some locations using an own TurboPlayer for a regional playout. If you want to monitor from the central TurboPlayer which elements are just being played at all locations, you can add some stacks to the GUI. Each of them would be loaded with the remote track for one location, but do not assign logical channels to the stack. So you have a read-only stack in which you can see how the remote TurboPlayers advance along their tracks.

5.2 Inter-Turbo communication

5.2.1 Communication types

There are two possible communication types which can be used for messages between TurboPlayers. The first type is the direct communication via TCP/IP and the second type is via BCS broadcast messages.

Direct communication is configured in the DigaSystem registry in the key:

TurboPlayer\Communication\RemoteTurboPlayers

Each subkey represents one remote TurboPlayer. As for the internal communication, ThreadMessageBoxes are used here, but for remoting only TCP/IP is supported as protocol. As usual, you have to specify the address / port of the endpoint. See the parameter description for more information. If remote connections are configured, TurboPlayer will try to establish the specified connection during startup and continuously later on, when it is lost. There is also a special GUI window you can add to the surface which shows the connection state of all configured TurboPlayers.

Communication via BCS broadcast messages is normally enabled, but you can disable it with the parameter:

TurboPlayer\Communication\RemoteTurboPlayers\SendBroadcastMessages

Normally, this indirect type of messaging should not be used, the direct link is faster – as it is direct – and safer – as the connection state is permanently surveyed and can be displayed to the user. But in case you do not want to open additional ports on your on-air client, you can use the BC servers as relay. Please note two things: First, it does not matter if you have both communications enabled. The exchanged messages are designed in a way that if a message is received twice, the second one does no harm. Second, there is an authentication for BCS broadcast messages. This is useful, because a TurboPlayer can be fully remote-controlled with these messages. The mechanism implemented uses the current time – therefore you must pay attention to a correct synchronization of the system clocks. Authentication is enabled by default but can be disabled with the parameter:

TurboPlayer\Communication\GenericMessageAuthentication

5.2.2 Next info

The information about the next element is something that is being permanently exchanged between TurboPlayers. Whenever the next element in any rundown changes, all other TurboPlayers are informed. A TurboPlayer that receives the information will store it internally (you can see it in the state dump of the TreeManager) and if there are any elements, which have a relative link to one of the known next elements, they can be preloaded. This preloading is necessary, because when an element becomes next element, it can be expected that this element will start at any moment. Therefore, other dependent relative elements must be prepared as well.

5.2.3 Load state info

Since version 6.1 it is also supported to send load state messages, which inform about preloaded elements (or unloaded ones or if a load has failed). This needs to be activated with the parameter:

TurboPlayer\Communication\RemoteTurboPlayers\SendLoadStateMessages

This information is not needed by the remote TurboPlayer engines, but it is needed if you want to visualize the load state of remotely played elements in the rundown view of a TurboPlayerGUI with “Play info” column (which can display the remote load state, too). Sending the load state is only one of the possible information messages. Sending messages must still be activated in general with the parameter:

TurboPlayer\Communication\RemoteTurboPlayers\SendMessagesForRundown

5.3 Relative and synchronized / slave starts

In regionalization (and another case mentioned below) it is sometimes necessary to start elements on different tracks simultaneously. This is typically the case for the first element of a regionalized block. When the element on the main track starts, corresponding elements on other tracks should start too. TurboPlayer supports this sort of simultaneous start. First, we describe the underlying mechanism and afterwards the details and necessary settings.

In a complex installation there might be the problem that there are multiple TurboPlayers running and each of them has only its individual tracks loaded. In such an installation the TurboPlayer which starts the main element does not know whether there are dependent slave elements. And the TurboPlayers which want to start the slave elements have no knowledge about the main element and cannot survey the main track. Due to this problem the technical solution is that each TurboPlayer, which starts an element, sends out a broadcast message to all other TurboPlayers with information about the just started element. The TurboPlayers with the slave elements will receive the start message and will examine the information in it. If the start happened in the same program / service and if there are any slave elements which are ready, they are started.

Some detail explanations:

- What makes an element be a “**slave element**” ? There are multiple answers:
  1) The element has a **relative start**. This means:
     - <Time_StartType> = “Relative”
     - <Time_StartMode> = “Relative”
     - <Time_ReferenceID> is present
     - <Time_ReferenceGUID> is present.

     These values can be set in DigAIRange when you select a relative start. Both reference values are a link to the master element, the first one is a relative path to the ID of the master element, the second one is the exact value of the field <GUID> of the master element.
  2) An element has start mode **SyncStart**. This signals that the element should be started synchronized with another element. The criterium is that the master and the slave elements have the same content in the SyncStartField. This field needs to be configured in the DigaSystem registry in parameter:
     `BroadcastSystem\SyncStartField`
     Though you can use any of the existing regular or custom fields, it is recommended to define an extra field for this purpose, e.g. “SyncStartID” or “SSID”. Using an extra field ensures that the field has exactly one meaning and that there are no random matches or mismatches. The disadvantage is that the user has to fill it out manually. This might be a challenge, though the configured field is visible in the general submask for an element in DigAIRange and the user does not need to make the field be unique for the whole world and all times. The field need not even be unique at all. If the master elements and the slave elements have the same order (meaning: no mixup of the positions) then the SyncStartID could even always be the same, because TurboPlayer will always start only a single slave element for each rundown but not multiple ones. If you use the SyncStart mode in conjunction with EndpointStories (see below) then BCS will compute the SyncStartField automatically.
  3) Amendment: the start mode SyncStart for elements is only available since version 6.1. Earlier versions could execute a synchronized start even without this start mode. Therefore, a special TurboPlayer mode “**SyncStart**” had to be active. If this mode is active, TurboPlayer acts as described above in (2) – but for every element, not only the ones with start mode SyncStart. There is one difference: if the TurboPlayer mode SyncStart is active, TurboPlayer will not only make use of the SyncStartField, but of the field “GUID”, too. If master and slave element have the same GUID they are started synchronized. If two elements are in different rundowns / tracks, they will never have the same GUID. In BroadcastSystem this will happen only if one of the rundowns is created with the BUS sync task (used to create identical rundowns for simultaneous playout). We do not recommend to use the old TurboPlayer mode SyncStart anymore. Since version 6.1 you should always use the new start mode SyncStart for elements. This makes the intended start visible in the rundown and there are no secret starts executed in the background. There is even a parameter, which allows to deactivate the old behaviour:
     `TurboPlayer\SyncPlayOptions`

The default value is <empty>, which means: the old behaviour is still active. But if you set this parameter to “Strict” (aside of other possible options), TurboPlayer does no longer start arbitrary elements synchronized. Then the elements must definitely have start mode SyncStart.

- And what does mean “the elements are **ready**” ? It does not mean that the slave element is the <next> element. The slave element might be somewhere down in the rundown and can be started, nevertheless. In doing so, previous elements might be skipped as usual. As the messages are evaluated by the kernel, the slave elements must be known by the kernel. This means: the elements must be in the range of preloaded elements. Elements with a relative start are preferred when the preloaded elements are selected (like with an absolute start). This is not true for SyncStart elements.
- Both types of start work on multiple TurboPlayers via network communication but also on a single computer. The tracks with the slave elements might be loaded on different TurboPlayers or into multiple stacks of a single TurboPlayer. If you use multiple computers the rundown for the master and slave element can be the same, e.g. the show list. On a single computer the slave elements for one master element must all be distributed to individual rundowns.
- Relative starts work only in one direction: the slave element is always the element with the relative start attributes. In contrast, the SyncStart might work in both directions because none of the elements is different. This might be useful sometimes, but it is only true if you use the old TurboPlayer mode SyncStart (see above). If you use the start mode SyncStart for individual elements the synchronization works only in one direction: a “normal” element triggers the start of a SyncStart element.
- Pay attention: in a setup with two (or more) active TurboPlayers, which load the same track, you will encounter problems. (Typically, this setup is used with multiple studios operating alternating on the same rundown.) Here you must not use the SyncStart feature ! Depending on the exact timing it will happen in some cases that multiple TurboPlayers start an element and in other cases only one TurboPlayer starts it. The solution is to have only a single active TurboPlayer because in passive activation sync starts will normally not be executed. (You can change this with the “StartStopPassive” parameter for special purposes.)
- Neither relative nor sync starts work across different services. All elements must be part of one service.
- Relative starts can have an **offset**, meaning the slave element starts after a certain delay. This is not possible for the SyncStart. Here the elements are started together.
- In the start message the number of the channel within the rundown is included. The receiving TurboPlayer will try to use the “same” channel of its rundown. For example: if the master element is started on the second channel, the slave element will start on the second channel of its rundown too. This applies only if the channel on the slave system is available. If it is not, the normal channel selection is done. This feature might be useful if you want to control the faders of the regionalized channels together with the master fader.
- The achievable **precision** depends on the setup. The slowest start will be for a remote TurboPlayer with a slow network connection and messages via BCS. Here you can expect delays of some 100 ms. The fastest slave start can be achieved on a single TurboPlayer when possible, pre-start operations (like opening a fader) are eliminated. Here you can expect a delay of a few ms only.

5.3.1 Limiting sent messages

By default, TurboPlayer sends information about the next element and start operations for all elements in all rundowns. This can produce a high number of messages being sent in the network, especially in installations with many TurboPlayers. In most cases it is not necessary that this big amount of information is distributed within your network. For example, in an installation with many remote TurboPlayers, which all depend on the start of a single central TurboPlayer only, there is no need for the remote TurboPlayers to send out any information. The parameter:

`TurboPlayer\Communication\RemoteTurboPlayers\SendMessagesForRundown`

can be used to switch off the sending of the messages. Besides, you can limit it to necessary rundowns – e.g. the showlist only.

5.4 Using EndpointStories for regionalization

5.4.1 Introduction

Using EndpointStories for regionalization is a topic for many modules – nearly all of BroadcastSystem, but also for DatabaseManager/ContentManager. You can find an explanation of the basic concept and some descriptions for the other modules in BCSTechManual in chapter 11.4. Please read that chapter first!

Important: the required features were introduced in version 6.1 of the modules of BroadcastSystem. Older versions do not support EndpointStories.

5.4.2 Stacks and stack mode Endpoints

Stacks can be configured to operate in stack mode “Endpoints”. In this mode changes of the rundown by the user are not possible. The content of such a stack is filled and modified only by TurboPlayer. This is done by evaluating the rundown list “Show”. Elements, which have the right DistributionEndpoint (=region) are picked and are copied to the stack. TurboPlayer must be informed about the right DistributionEndpoint by setting the parameter:

`TurboPlayer\RundownLists\Stack<n>\DistributionEndpoints`

The name is in plural, because you can specify a comma-separated list of multiple endpoints. Nevertheless, you can use the alternative value names “Regions” or “Region”.

TurboPlayer will not only pick the regionalized elements, but the surrounding EndpointStories and possibly groups, too. This is necessary to keep the right node IDs for each level. There is also a permanent synchronization for the copied nodes running. Any BCS notifications, which modify one of the regionalized elements, are applied to the element in the show rundown and at the same time to the copy of the element in the stack.

As the show rundown should not play any of the regionalized elements, it is not necessary that they are preloaded for this rundown. Only for the right stack they should be preloaded, of course. This can be achieved by configuration, in which there are two possibilities:

1) You can set the parameter:
   `TurboPlayer\RundownList\Show\DistributionEndpoints=TurboPlayer`
   The name of this special endpoint is fixed. Elements, which are not assigned to a region – so called “neutral elements” – have either the endpoint “TurboPlayer” or an empty one (which means the same). With this setting only the neutral elements are preloaded for the show rundown. This setting is the right one if you are playing the regionalized elements from the same TurboPlayer as the regular and the neutral elements.
2) You can set the parameter:
   `TurboPlayer\RundownList\Show\PreloadedElements=-1`
   In this case TurboPlayer will not preload any element for the show rundown. Then you are not able to play any of the neutral elements from this TurboPlayer, of course. Therefore, this setting is the right one if you have an extra TurboPlayer running in the background, which should only play the regionalized elements but nothing else. In this case you will need a communication channel between the main and the regio TurboPlayer, so that the main TurboPlayer can remote-control the regio TurboPlayer. (This is described below.)

When TurboPlayer distributes the elements for the regions onto the stacks, it can happen that the result is an empty EndpointStory in one of the stacks. This means that no element was scheduled for this region. If such an EndpointStory is played, there will probably be silence for the affected region. In order to prevent this silence, you can make use of the parameter:

`TurboPlayer\RundownLists\Stack<n>\FillNeutralElement=true`

Then TurboPlayer will pick the neutral element(s) of the EndpointStory and make a copy in the affected stack. As a result, the neutral element will be played multiple times simultaneously – once for the show rundown and one or more times for the affected stack(s). You will see this fact in the protocol track of BCS – such an element appears more than once.

Regarding the protocol track there is nothing special. As usual, you can use the parameter:

TurboPlayer\RundownLists\Stack<n>\ProtocolTrack

to define whether a protocol is written and to which protocol track. This can be the same track for all regio stacks and the same track as for the show rundown.

Advancing the shows for stacks in mode Endpoints does not need special attention. It is always strongly coupled to the advancement of shows within the show rundown. The parameter “TracksFollowShow” is not applied here.

In principle, you can change the stack mode to/from “Endpoints” during runtime of TurboPlayer, but it is not recommended to do this. If you really do it, you should afterwards perform a full show loading including a TurboPlayer reset. Otherwise, the content of rundowns might not be filled properly.

5.4.3 Synchronized start / pause / stop

For regionalization it is necessary by definition that elements for different regions play at the same time. Therefore, TurboPlayer supports a synchronization of the start of elements. Since version 6.1 the start mode “SyncStart” can be used for this purpose. If you use EndpointStories, the SyncStart mode will be assigned by BroadcastServer automatically. Typically, there is a first neutral element (not intended for any specific region), which can have any arbitrary start mode, like e.g. Manual, Sequenced or StartOnTime. The very first element for a specific region (=DistributionEndpoint) will get the start mode SyncStart assigned by BCS. If needed, you can put multiple elements per region into an EndpointStory. In this case all further elements will get the start mode Sequenced. Any other start modes than these two are not supported so far. The start mode SyncStart needs a common ID for the elements, too. The field for this ID must be configured in parameter:

BroadcastSystem\SyncStartField

BCS will take care of this field, too, and the user does not have to fill it out. For more information, please see chapter 11.4 of BCSTechManual.

When elements have a start mode SyncStart, you do not have to do anything special, like activating the TurboPlayer mode “SyncStart”. This TP mode was existing before the start mode for elements was introduced with version 6.1. It is not needed and it is even recommended to disable a synchronized start of elements without having set a start mode SyncStart for them. Normally, you should not have to take care about anything for the synchronized start to work, except of basic things, like that a stack for the region must exist and elements in that stack must be preloaded.

So far, we have only talked about a synchronized start. Sometimes it is desirable to handle pausing and stopping of elements in a synchronized way. This is typically the case for shows in live-assist mode. If the talent starts a regionalized leg by starting the first neutral element and then has to either pause or stop the playout then it should be sufficient to pause/stop the neutral element and all regionalized elements should do the same. This behaviour can be activated with a registry parameter:

TurboPlayer\SyncPlayOptions

The value can be a comma-separated list of keywords. These three keywords are defined so far:

- **Pause**: If "Pause" is set, elements, which were started with a SyncStart, are also paused synchronized. The same is true for a restart after a pause.
- **Stop**: If "Stop" is set, elements, which were started with a SyncStart, are also stopped synchronized, when the stop was triggered explicitly/manually by the user. If a master element is stopped regularly (by running till the intended end) the stop is not synchronized. In this case the slave elements will run till their intended end, too.
- **Strict**: If "Strict" is set, TurboPlayer will only start elements synchronized, which have the start mode "SyncStart". Without this option TurboPlayer can start any element synchronized when the TurboPlayer mode “SyncStart” is active and elements have the same content in the specified SyncStartField. This is the old behaviour when the start mode SyncStart for elements was not yet available and it should no longer be used.

Another topic is to start the first regular element outside of an EndpointStory after all elements inside the EndpointStory have stopped playing. Therefore, TurboPlayer supports a special InEvent, which is called "**TP_AllRegioChannelsStopped**". It is triggered when the last element in an EndpointStory has finished playing and you can set an external start for this InEvent for the first element outside of the EndpointStory. Please see chapter 4.2.2 for more information about this InEvent.

5.4.4 Using regio TurboPlayers running in background

You can configure a single TurboPlayer so that it can handle regionalized playout itself. This requires to provide one or more stacks and one or more channels for that purpose. Sometimes this is not desirable or not even possible, e.g. because you have to server too many regions to reasonably be handled by a single TurboPlayer. In this case you can make use of one or more extra TurboPlayers, which run in the background, which handle only the regionalized playout and which are remote-controlled by the main/master TurboPlayer. Hint: as such a solution includes remote-controlling, we sometimes refer to the background TurboPlayers as “remote” – though they are not in a different location far away.

The basic requirement for the remote-controlling is to connect the main TurboPlayer to the regio TurboPlayers with the inter-Turbo communication, as it is described above in chapter 0. In that chapter it is described that there can be two types of message exchange: (1) a direct TCP/IP connection between two TurboPlayers and (2) a communication via BCS generic broadcast messages. When you want to remote-control a TurboPlayer for a synchronized start it is a MUST to do it with a direct connection. Otherwise, you will lose any chance for a shortest possible delay between the master- and the slave start. The second way via BCS is a nice thing to use in addition. This gives a slightly higher chance that a start even works in case of communication problems on the direct line. By the way: it will not happen that a message is handled twice if it is received via both ways of communication. Each message has a unique ID and a receiving TurboPlayer will handle each message only once.

Pay attention that the computer clocks of all computers are synchronized as best as possible. At a minimum they should have a synchronization, which has an accuracy of 1 second. Please see chapter 6.5 “Play Info” below for hints how to achieve an even better time synchronization.

In the described setup the intention is that the main TurboPlayer only plays regular elements and neutral elements of an EndpointStory, but no elements dedicated to a specific region. The regio TurboPlayers in contrast, should only play the regionalized elements, nothing else. The first wish can be achieved by setting the parameter:

```
TurboPlayer\RundownLists\Show\DistributionEndpoints=TurboPlayer
```

This causes the main TurboPlayer to only load regular or neutral elements, because they have either the DistributionEndpoint “TurboPlayer” or this field is not set at all (which has exactly the same meaning). On the regio TurboPlayer(s) you set the parameter:

```
TurboPlayer\RundownLists\Show\PreloadedElements=-1
```

These TurboPlayers should load the same show content form BCS as the main TurboPlayer, but they should not preload any element for the show rundown. If elements are not preloaded, they will not be played either. Regionalized elements should only be loaded and played for the corresponding stack. Therefore, you set up stack mode “Endpoints” and the correct region for each stack as it is described above.

You can think about using the parameter:

```
TurboPlayer\SlavedShowLoading=true
```

for the regio TurboPlayers. This setting will make the regio TurboPlayers always follow with their loaded shows the main TurboPlayer – even if there are some elements in their rundowns, which have not yet been played.

5.4.5 Switching of the audio routing during regionalized legs / fades

Depending on your setup, it might be necessary to change the audio routing when a regionalized leg starts and when it ends. For example, the regionalized outputs might be switched to the regionalized channels of TurboPlayer when the regionalized leg starts and they are switched back to the main output channels when the leg ends.

TurboPlayer has two internal events, which support such a switching:

- **`EndpointStoryBegin`**
- **`EndpointStoryEnd`**

Please have a look to chapter 4.5.6 “Internal Events” for more information.

The begin event is triggered **before** the first element in an Endpoint starts and the end event is triggered **after** the last element in an EndpointStory stops. The time is typically 100 ms before start, but this can be more, depending on the type of the start and the line parameters OpenDelayPassive or OpenDelayActive. Normally, this should be enough time to change the audio routing so that the begin of the first region elements is not cut off.

TurboPlayer is not capable of directly changing the audio routing. You need to write a macro command for these two internal events, which do the switch. For example, you could call an Ember_SetParameter function of the TIOEmberPlus.dll, which would then relay the information to an EmBER+ device by changing any arbitrary node in the EmBER+ tree.

When using audio switching it is not possible to executes fades between elements outside and inside of EndpointStories. There should even be a small gap in between of these elements so that the switch of the audio routing can be done without audible artifacts. This requires a correct scheduling so that the last regular element outside of an EndpointStory is clearly finished, then there is a gap and only then the first element inside of the EndpointStory should start. For a manual start the talent must be aware of this and must not try to perform any fades on the border to the EndpointStory. For an automatic start you need to set the start attributes accordingly.

If you want to perform fades into and out of an EndpointStory you cannot use a switching of the audio routing. This must be done with a static audio routing, which is supported by TurboPlayer, too. You need to set up a routing like this:

- **`Sum for the regular output:`**
  - **`Regular TP channel 1`**
  - **`Regular TP channel 2`**
  - **`Regular TP channel 3`**
  - **`TP channel 4, reserved for neutral elements`**
  - **`TP channel 5, reserved for neutral elements`**
  - **`TP channel 6, reserved for neutral elements`**
- **`Sum for the output to region x:`**
  - **`TP channel 7 for region x`**
  - **`TP channel 8 for region x`**
  - **`Regular TP channel 1`**
  - **`Regular TP channel 2`**
  - **`Regular TP channel 3`**

A regular element is one of the elements outside of an EndpointStory, which should be audible in the main output and in all regions, too. A neutral element is one of the elements inside of an EndpointStory, which should be played on the main output meanwhile regionalized elements are played on the outputs for each region. As you can see, the idea is to have extra channels, which are reserved only for the neutral elements inside of an EndpointStory. These special channels are statically mixed only to the main output, whereas the regular channels are statically routed to both outputs: to the main one and to the regionalized ones.

In order to make it easier for the user, you should have as many channels for neutral elements as there are regular channels and you should assign them to the same lines/faders. For example: Line-1 / fader 1 should have channel 1 and channel 4 assigned. This allows the user to open fader 1 and he does not have to care, whether he is starting a regular element or a neutral element inside of an EndpointStory.

Here is an example of the timing when an EndpointStory starts:

- The last element in front of an EndpointStory is playing and starts to fade out. As it is playing on a regular channel, this element is audible on the main output and on the output for all regions.
- The first neutral element – and with it all regionalized elements – inside the EndpointStory starts. The neutral element starts on a special channel, which is reserved for neutral elements. Therefore, this is only audible on the main output. The regionalized elements start synchronized, but on the channels reserved for the corresponding region. They are only audible on the output for a region.
- When the last regular element has faded out it stops. The playing neutral element is only audible on the main output.

Such a setup ensures that no listener hears a crossfade between three elements, only between the last regular element and “his” regionalized or neutral element.

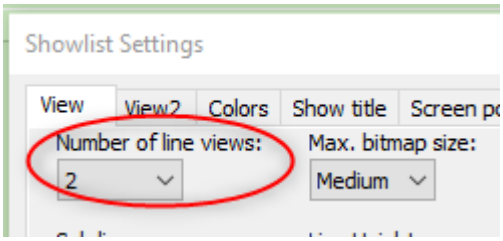
In case no element was scheduled for a specific region, this would result in silence for this region. To avoid that, TurboPlayer has the parameter:

`TurboPlayer\RundownLists\Stack<n>\FillNeutralElement=true`

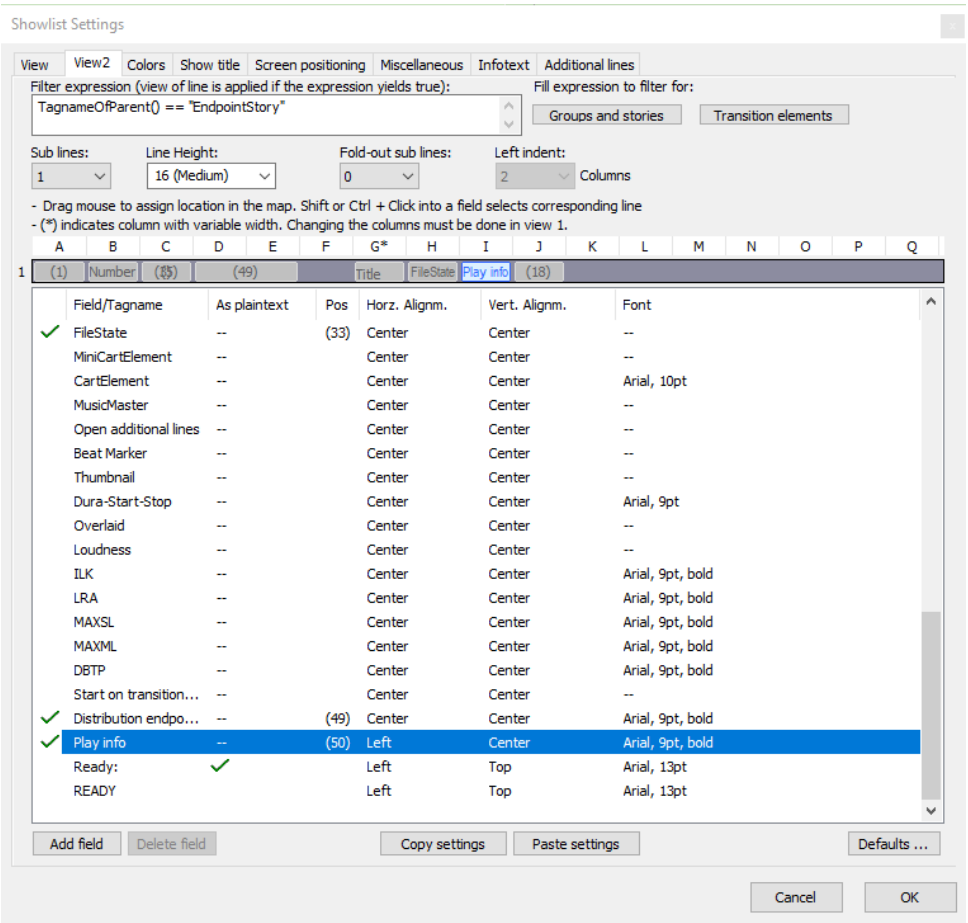
If set, TurboPlayer will make a copy of the neutral element(s) for each region, which would otherwise have an empty regionalized leg. As a result, there will no silence for any of the regions.

5.4.6 Visualization of regio elements and of load and play info for remote TurboPlayers

First of all, there might be the wish that elements, which are not handled by the main TurboPlayer are displayed smaller as the regular elements. This makes it immediately visible to the user that there is something special about these elements. You can achieve a different setting for elements in an EndpointStory by adding a second “line view” in the settings for the show view:



This allows you to define for which elements the second line view should be applied and to adjust these different settings. Here is an example. You can also see, which condition you need to configure to make the second line view be applied to all child elements of an EndpointStory.



In case you have set up the different TurboPlayers as it was described in the last chapter, you might have the wish that it should be possible to see the status of the background TurboPlayers somehow in the main TurboPlayer. Otherwise, the users would have to e.g. make use of one or multiple instances of Remote Desktop in order to monitor the background TurboPlayers.

TurboPlayer supports the visualization of the load state and of play information for elements in a rundown, which are handled by a different TurboPlayer. This is done in an extra field/column, which you can configure to appear in the view of the show rundown. This is the field “Play info”. This field will need quite a lot of horizontal space, because it does not only have a background color, which represents the load and/or play state, it displays the playtime, the remain time and additional information, too.

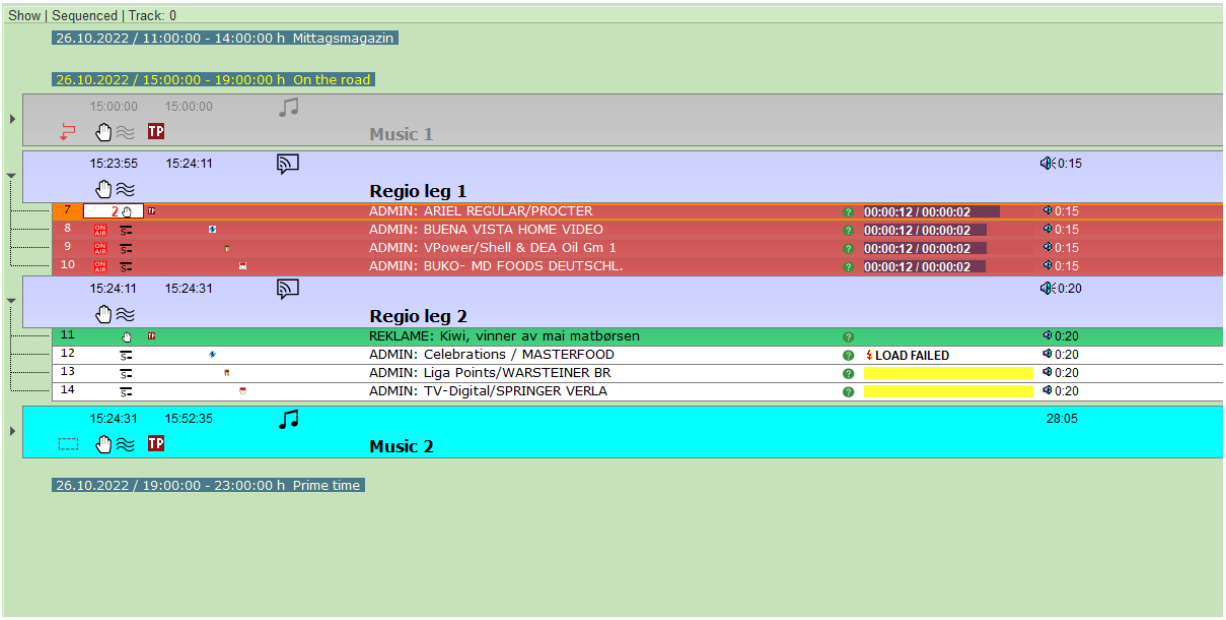
For the question “how is the information about the load state and the play info transmitted to the main TurboPlayer?” we have to distinguish between play info and load state. Both types of information are transmitted via different means.

1. The play info is sent by TurboPlayers to BCS via special messages. BCS collects this information from possibly multiple TurboPlayers and then sends out regular notifications to all interested clients. Not only TurboPlayer, also DigAIRange can display this information in the rundown view. The information contains the play state (playing, paused, restarted, stopped), the play- and the remain time (if known) and some additional information (mark-out will be/was ignored, element is looping, etc.). This is displayed in textual form. The background of the field has a progress bar, visualizing the play-/remain time.
   Sending out play info is only done if it was configured. This happens with the parameter:
   `TurboPlayer\Communication\RemoteTurboPlayers\SendMessagesForRundown=...`
   The parameter must name the rundown lists for which messages should be sent out. This can either be a comma-separated list, like “Show,Stack1” or you can use a star “*” in order to send messages for all rundowns. On the receiver side no extra configuration is necessary. It is sufficient to add the “Play info” field/column to the rundown view and the GUIs will request the needed information from

the engine and/or BCS automatically. Hint: the same mechanism is used if you configure one of the player windows of TurboPlayer to display play info of remote TurboPlayers by using a channel number 10000 or higher.

2. The load state information is sent from the background TurboPlayer to the main TurboPlayer. This can either be done with a direct connection of the affected TurboPlayers (which is strongly recommended) or via generic broadcast messages of BCS. By default, a TurboPlayer does not send out load state messages. This is only done if you set this parameter:
   `TurboPlayer\Communication\RemoteTurboPlayers\SendLoadStateMessages=true`
   Sending messages for a specific rundown must already have been activated with the other parameter "SendMessagesForRundown". Load state messages are only used by TurboPlayers. Other BCS clients cannot make use of them.
   The receiving TurboPlayer will display the load state within the "Play info" field as long as the remote element is not playing. Then the background color of the field will have the color for preloaded elements. If the load did fail, a flash symbol and the words "LOAD FAILED" will be displayed. If the connection to the corresponding remote TurboPlayer is lost, the words "CONNECTION DISTURBED" will be displayed. This makes it clearly visible to the user that there is a problem with one of the remote regio elements and he can decide what to do about it, like: play the EndpointStory nevertheless, skip the EndpointStory, remove the failed element so that a neutral element is played instead or inform his team that a problem exists, which must be solved.

Here is an example screenshot of a show rundown in a main TurboPlayer. You can see that the line view for regio elements is quite different than for other elements. You can also see the play info for the elements of the first EndpointStory and the load state for the elements in the second EndpointStory. Loading of the first regio element in this story obviously failed.



Hint: displaying the load state and the play info like this in the view for the show rundown does not only work if the elements are loaded and played by a remote TurboPlayer. It works, too, if the elements are loaded and played by a stack in stack mode Endpoints on the same TurboPlayer. Nevertheless, you need to activate sending messages in the engine with the SendMessagesForRundown parameter and sending load state messages with the parameter SendLoadStateMessages in exactly the same way as it is necessary for sending messages to a remote TurboPlayer.

6 FAQs, Problems and Solutions

6.1 Move activation between 2 TurboPlayers

In installations with multiple TurboPlayers you often have the problem that parts of the rundown should be played by one TurboPlayer and other parts by a second (or any additional) TurboPlayer. If there are presenters running the show in live-assist mode, this is possible with multiple active TurboPlayers. In this case it is important not to use time starts because they might be executed by more than one TurboPlayer. Here we want to talk about a different case: a full (or a partial) automatic show and only one TurboPlayer should be active and playing at each time. So, the question is: how can you achieve that the activation is moved between TurboPlayers at a certain point in the rundown whereas "moving" means, one TurboPlayer becomes passive, the other one becomes active and starts playing. We are going to discuss two solutions.

6.1.1 Using TransferExecution

TurboPlayer can send macro commands to another TurboPlayer. This is done with the special command "TransferExecution". It must appear as first line within a macro and the remaining part of the macro is executed on the computer which is specified as parameter. Therefore, the example code below is divided intow two parts. One part switches the active computer to passive (macro 1) and calls macro 2. Macro2 will be executed on the TurboPlayer which should become active and take over the playout. Here is the example code:

Macro 1:

```
TP_ToggleMode   (   Activation,
                    Passive )
Macro2()
```

Macro 2:

```
TransferExecution ( "TurboMod", "DIRECT+BCS" )
TP_ToggleMode ( Activation, LiveAssist )
%i = 0
while ( %i < 20 &&
      CurrentMode ( Activation ) $= "Passive" &&
      DataOfElement (Show, Next, MediaType) $= "Control")
{
    Sleep ( 100 )
    %i = %i + 1
}
TP_Start ( Show, NextSystem )
```

Macro1 can be called from a control element (or any other element with a control appended to it) within the rundown. Just as well macro1 can be called from any other trigger, like an external signal.

To work, macro2 must be transferred via network. This works either with broadcast-messages via the BCS or with a direct connection between both TurboPlayers. We recommend activating both ways, so you are on the safe side.

Macro2 uses a while-loop to wait for two things to happen: the first one is that the takeover of the activation mode succeeds. The second one is that the <next> pointer points to a non-control element. (Here we assume that the first element to be played on the new TurboPlayer is a system element, e. g. an audio element.) With this wait we achieve that the control element got at least the send state “sending” which causes the <next> pointer to be moved to the next element after the control element. Without this wait it can happen that the new TurboPlayer sets the control element to “Skipped” when it starts the first system element in the last line of the macro.

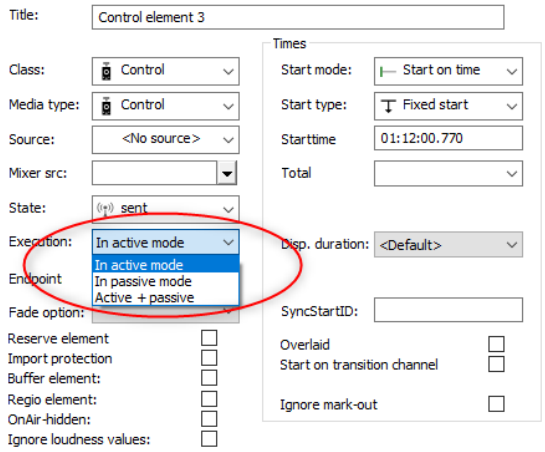
The start command in the last line of the macro uses the special keyword “NextSystem”. This is intended to prevent that the control element is started by the new TurboPlayer, too. Normally, this should not happen because we wait until the <next> element is no longer the control element. But the loop has an emergency-break after 2 seconds (20 * 100ms sleep) so it might happen that the loop is being left while the <next> pointer still points to the control element.

Another thing you should take care of is to allow the execution of start commands triggered by macros for a passive TurboPlayer. This can be achieved by adding the string “EventOut” to the parameter “TurboPlayer\StartStopPassive”. If you do not use this parameter “EventOut” will be an allowed start trigger by default.

The last important topic is good time synchronization between the two TurboPlayers. Otherwise, the real times being written will be in disorder or it is even possible that the authentication fails.

6.1.2 Using a special control element

A pure control element (media type = control) can be started by TurboPlayer even in passive mode. To achieve this, you have to set a special flag for the element, which specifies in which activation modes the element should be started. In DigAIRange this flag can be found in the general submask (see picture below). So the principle idea is to use a single control element which executes a macro which performs the switching on both computers. Find below an example macro which performs this task:




```
if ( ComputerName() ~= "TurboNews" )
{
    TP_ToggleMode ( Activation, Passive )
}
else if ( ComputerName() ~= "TurboMod" )
{
    TP_ToggleMode ( Activation, Active )
    Sleep ( 1000 )
    TP_Start ( Show, Next )
}
```

You might note that the macro checks on which computer it is running and on each computer, it is doing something different. You might wonder how this can work, because normally only a single TurboPlayer will execute an element. The other TurboPlayer sees that the element is already "playing" and will not start the element a second time. But for such a special element there is a different logic: the element can run on the passive TurboPlayer although it already runs on the active one. This works for elements with an absolute time start. For elements with a sequenced or a relative start it would work too, but the problem is that the previous element is not running normally (if it is an audio element only running on the active TurboPlayer). Therefore, we recommend using this method with a time start only.

You have to pay attention that the element is not finished on one TurboPlayer before the other TurboPlayer has started the element. This can be achieved by two means: first, all computers must be synchronized. This means the time difference must be below approx. one second ! And second, the element must have a duration which is longer that the maximum time difference between the computers. If this is fulfilled, both TurboPlayers will run the element simultaneously and on both computers the macro will perform a change of the activation mode. In addition, the example shows how to start the next element in the rundown on the TurboPlayer which becomes active.

Be aware that this workflow can create a small interruption of about a second in the playout. This can be avoided e.g. by slowly fading out the last running element in the TurboPlayer which is going to passive mode. Such an extension can be added to the macro easily.

6.2 Playout of text elements

No, TurboPlayer cannot read text elements – there is no text to speech routine. Playout of text elements means any type of treatment of text elements so that you either get time information and / or the elements get a send state.

Users expect different ways of handling text. Therefore, TurboPlayer supports some different ways. The main decision is: should text elements be played virtually or not ?

6.2.1 Playing text virtually

If text elements are played virtually, you can use one of the player windows to display information about the element, like: play time, remain time or a progress bar. All you have to do is to add “Text” to the registry parameter “TurboPlayer\LiveMediaTypes” to declare that text elements are live elements. Then TurboPlayer will treat them as playable. They are included in the preload range, the <next> pointer steps onto them, they can be started and stopped, and they get the normal metadata like the send state and real times.

In principle there are two ways how to start a text element. First, you can use a start button. This button must trigger a macro command which starts either the <next element> or one of the virtual channels, because a text element can only be played on a virtual channel. Therefore, a start command for an audio channel does not help. The same is true when the user opens a fader. This normally triggers the start of the corresponding audio channel only. There is the parameter “TurboPlayer\ ShiftToLiveChannel”. If you activate this parameter, the user can open any fader and TurboPlayer will automatically start the element on the next free virtual channel. This is the second way how to start a text element and might be useful sometimes.

Normally a text element has a planned duration and TurboPlayer will automatically stop the element after this time. If the user reads the text while the element runs virtually, it might not be desired that TurboPlayer ends the element himself but instead waits for an external stop command. This can be achieved with the parameter “TurboPlayer\IgnoreMarkOutMediaTypes”. If you add the text media type to this parameter all text elements will continue to play after their mark-out (=planned duration). To stop the text element, you can use a button. The macro command “TP_TreatSpecial” will be a help because it can be used to stop all running text elements easily. Here is the example of a small macro command which stops all running text elements and if the <next element> is a text element it is started. The usage is a typical “Start/Stop text” button:

```
TP_TreatSpecial ( Show, Text, Stop )
if ( DataOfElement ( Show, Next, MediaType ) $= "Text" )
{
    TP_Start ( Show, Next )
}
```

6.2.2 Tick off text elements

Often text elements should not be played virtually. These elements should be in the rundown to give the user a clue when to read the text. But the user does it by reading from a paper and he is not interested in any play information. Nevertheless, he should inform the system whether he did read the text or not.

First of all, text elements are not treated as playable if the text class/media type does not appear in the parameter mentioned above. The user can only set the send state of text elements to “sent” or “skipped”. This can be done with the macro command “TP_TreatSpecial” which allows to perform these operations. For example, a button could execute this command:

```
TP_TreatSpecial ( Show, Text, Sent, Next )
```

It sets the <next element> to send state “Sent” if this element is a text element. Of course, this requires a manual action of the user. It is also possible to do this tick-off automatically when the next audio element is started. Let’s look at this macro which should be installed for the internal event “ElementStarting”:

```
TP_TreatSpecial ( %TheRundown, Text, Sent, PreNewestPlaying )
```

Whenever a new element starts the macro sets text elements which are in front of the just starting element (=newest playing) to “Sent”. Therefore, the user does not need to do anything manually. If you

do not make use of this macro TurboPlayer will set text elements to “skipped” which is the standard behaviour.

6.2.3 Macro for the lazy user

In case you want to play text elements virtually we have to modify the previous macro. The user must press a button to start a text element when he starts reading (see above). Ideally, he presses the button again when he is finished which sets the element to “Sent”. But if he forgets to press the button, we can write a macro which performs this operation automatically. Again, we install this macro for the internal event “ElementStarting”:

```
if ( ! ( DataOfElement ( TheRundown, TheElement, MediaType ) $= "Text" ) )
{
    TP_TreatSpecial ( %TheRundown, Text, Stop )
    TP_TreatSpecial ( %TheRundown, Text, Skip, PreNewestPlaying )
}
```

Now it is important to check whether we are just starting a non-text element (because text elements can now be started too). This is done with the “if” in the first line. The stop command in the third line stops all running text elements, setting them to “Sent”. But if there are text elements in front of the just starting element which were not “played” yet, we want to set them to “Skipped”. This is done with the command in the fourth line.

Be aware that this macro is useful if you start audio elements by opening a fader, too. But you cannot combine it with the “ShiftToLiveChannel” parameter which was proposed above. The reason is: if you activate the parameter, opening an audio fader will start a pending text element above the next audio element and the macro does not find anything to skip. Instead, you must activate another parameter: “TurboPlayer\PreferSystemElementStart”. Then the behavior is like this: if the <next> pointer is on a text element and the user opens an audio fader TurboPlayer will start the next audio element though it is not the <next> element. This allows the macro to skip the text element.

6.3 Automatic assignment of overlaid flag

6.3.1 Why is the overlaid flag needed ?

The overlaid flag (“Time_Overlaid”) should be assigned to all elements which are being played in parallel to the sequence of regular (=non-overlaid) elements. A typical example is a short drop-in (e.g. a station jingle) which is being played at the beginning of a music element. For more information about this flag see BCSTechManual chapter 4 about “Start attributes” and especially the chapter “Usage of overlaid flag”.

Let’s assume the situation that you have 2 music elements and the second one should start automatically and therefore got the start mode “Sequenced”. Now the user wants to play a short drop-in near the beginning of music 1. This is no problem at all if the user starts the drop-in from the jingles or any other rundown. There is also no problem if CrossfadeMixer or OnAIR TrackMixer is used to schedule the drop-in. You will end up with a rundown which has 3 elements:

(1) Music 1
(2) Drop-in
(3) Music 2

The drop-in will automatically get the correct start attributes, including the overlaid flag, assigned by CFM/OTM. This is necessary because music 2 has the start mode “Sequenced” which means: start if the predecessor has finished. Without the overlaid flag the predecessor would be the drop-in – which is probably not what the user expects. But with the overlaid flag it is signaled to TurboPlayer that the drop-in should be ignored in the sequence of regular elements and that music 2 should start after music 1.

The problem starts if the user creates the sequence above by inserting the drop-in into the rundown via TurboPlayer or via DigAIRange. In this case he would have to assign the overlaid flag manually. This might not be desirable, because it creates an additional burden to the user, and it also requires a fundamental understanding of the overlaid flag. Therefore, TurboPlayer can assign the overlaid flag automatically since version 5.4.2085.0.

Be aware that TurboPlayer can only make a guess whether the overlaid flag is really needed or not. The implemented logic works quite well for a standard situation but in special cases the guess might be wrong. Therefore, a general mode “AssignOverlaid” exists which can be activated or deactivated by the user or by a macro command. By default, the mode is off. If you need it permanently, you can activate it during startup by writing a little macro: add a new registry key below “TurboPlayer\EventsInternal”, set:

Event = Startup

Command = TP_ToggleMode (AssignOverlaid, On)

6.3.2 Description of the assignment logic

TurboPlayer assigns the flag “Time_Overlaid=1” to an element if:

(1) The general mode “AssignOverlaid” is active.

(2) A rundown which executes sequences is affected. (A rundown with rundown mode “Random” like it is used for the jingles does not execute sequences.)

(3) An element is started manually. This can be done:

   a. by opening a fader, including the generation of a spontaneous element,

   b. via key shortcut by pressing a key on a keyboard or

   c. by clicking a button in the GUI.

(4) The element does not have the overlaid flag already set from scheduling.

(5) The element is completely overlapped by a previously running non-overlaid element.

Details:

- The computation is done with every start or stop of an element, when the mark-out of an element changes, when the ignore-mark-out state changes or when the AssignOverlaid mode changes. Therefore, it is possible that an element gets the overlaid flag assigned during its start, but later other things might happen, and the overlaid flag is revoked because according to the logic above the flag is no longer necessary. Or it can happen that an element gets the overlaid flag only assigned during the stop.
- It is necessary that the running non-overlaid element is directly in front of the potential overlaid element. It is only possible that other overlaid elements are in-between, but according to BCSTechManual overlaid elements must directly follow their assigned main element. There must not be a change of the hierarchy or an interruption by other elements, like by info elements.
- For computing whether the elements overlap the currently known real start and stop times, the link and mark values and the ignore-mark-out state are taken into account.
- If a single element is quasi-stretched by pausing it, this is not integrated in the computation. In this case you might see a discrepancy to what you might expect. But: typically, all playing elements are only paused at the same time. This prolongs all elements by the same amount of time and therefore does not influence the computation.
- An interesting case is a spontaneous element. It is typically created with the ignore-mark-out flag set. As such an element does not have a take duration, TurboPlayer can only assume that such an element is going to run endlessly. This affects the computation, of course, because such an element can never be overlapped by a regular audio/video element with a finite length. Therefore, a spontaneous element with the ignore-mark-out flag set will not get the overlaid flag. Only if the user stops the spontaneous element the stop time is known and TurboPlayer might assign the overlaid flag accordingly.
- In pre-5.4.2085.0 versions TurboPlayer used to assign the overlaid flag to spontaneous elements. This happened always, independent of any mode, times or lengths. With the mentioned version the behavior has changed. Now spontaneous elements are treated in the same way as any other element. If you really want to keep the old behavior, you can achieve this by adding “[Time_Overlaid]1” to the metadata which is assigned to the spontaneous element during its creation. (See the parameters “SetData” and “SetDataUnexpected” of the mixer sources.) As the element has the overlaid flag initially set, TurboPlayer excludes the element from the automatic overlaid assignment. By the way, you can also set “[Time_Overlaid]0” in this special case to suppress the automatic assignment.
- If an element which has the overlaid flag set is stopped manually, a following non-overlaid element is not set to start mode “Manual”. It would be the normal behavior of TurboPlayer to set the next element to “Manual” if an element is stopped by manual interaction, but for overlaid elements it is clear that they do not interfere with the sequence of regular elements and therefore this action is not necessary.

6.4 Multiple GUIs on a computer

Since build 900 of TurboPlayer it is possible to start multiple TurboPlayer GUIs on a single computer. This scenario has a limited usefulness, and you must make a proper setup which is described in this chapter.

Normally TurboPlayer does not start in multiple instances. This is a very good limitation because then the usage of the resources is clear. You must think about things like: MultiPlayer and audio devices, IO modules and IO cards, CrossfadeMixer and MultiRec, Keypads and hooked shortcuts, communication means like IP ports, Logfiles and other stuff. It is not very easy – and for some of the resources impossible – to do a separation for multiple TurboPlayer instances. If you can accept these shortcomings, it would even be possible to start multiple TurboPlayers (with engine) on a computer, but we do neither recommend nor support this. But running a main TurboPlayer (with engine) and additional TurboPlayerGUIs on a computer is possible. In principle there is one single use-case when you should do this: if you want to watch what another TurboPlayer is doing. There might either be a pure watch computer which shows the GUIs of multiple remote TurboPlayers, or you might want to see GUIs in addition to the own TurboPlayer with engine.

To deactivate the protection which prevents the multi-instance start specify the command line parameter "/mi". Now TurboPlayer would start multiple times but always with the same settings and therefore with the same resources. This must be avoided by using different configurations (see chapter Concepts/Configurations) for each instance. The configuration to use can be given on the command line too, but first you must prepare a new key in the registry with settings which specify different resources than the other instances. Pay especially attention to the following things:

- Communication: An extra GUI for a remote TurboPlayer needs some IP connections. TurboPlayers way of communication uses each IP connection only one-directional. Therefore, even the GUI must open listen ports for the engine it connects to. As standard value the GUI uses the port 9978 for the TreeManager connection and 9979 for the RundownKernel connection. Both ports can be adjusted with the values "TurboPlayer\Communication\GUI\PortForTreeManager" and "...\PortForRundownKernel". The communication types can be limited to: "TurboPlayer\Communication\ComTypes=IP".
- Each additional GUI should have a unique GUI number (-> "TurboPlayer\Communication\GUI\GuiNumber"). The problem here is: it should be unique for the computer on which multiple GUIs are running, but at the same time it must be unique for all GUIs connected to one engine. With unique numbers the advantage is – again – the avoidance of conflicts. For example, you can use multiple instances of StandaloneCFM.exe on one computer.
- You should not start an IO-Manager or use any IO modules on the additional GUIs. This is not needed for a watch GUI. It is best to set the value "TurboPlayer\Communication\IOManager\InitializeIOManager" to FALSE.
- Pay attention to the key-shortcuts. Do not use hooking for the additional GUIs, hooking must be limited to / and should be used with the main TurboPlayer with engine. With multiple GUIs on the screen it will happen that an additional GUI has the focus and normal shortcuts will not reach the main TurboPlayer. Especially with an extra keypad it seems to be much better that this keypad always controls the main TurboPlayer.
- Protocol: The problem with the protocol is that there is only the top-level key "Protocol" in the registry. Therefore, the different configurations do not apply and all instances use the same protocol destinations. Most of the destinations are intended for the engine, but as you should not start multiple engines, they are no problem. For the GUI protocol ("TurboPlayer" destination) you can define a destination for each configuration by adding values "TurboPlayer_<config name>". If you forget it, the GUI will create these values automatically by copying the "TurboPlayer" value. Other protocol destinations (like BCE-Messages) cannot be differentiated but this only makes debugging a little bit more complicated. It should be no problem in normal operations.

6.5 Play Info

Since version 6.0 TurboPlayer can report the current play status of elements to BCS. This includes the play status itself, which immediately reports the new states "play", "pause" and "stop", but also regular

reports of the current play/remain time of playing elements. BCS will then collect this information from multiple TurboPlayers and for multiple programs/services.

In TurboPlayer sending these messages need to be activated. You can do this with the registry parameter "TurboPlayer\SendPlayInfoForChannels". Here you can list all channels, for which TurboPlayer will send messages. The parameter can be the joker "*" (a star) in order to send messages for all channels. Otherwise, the value is a comma-separated list of channel numbers, but you can also use a range. Example: "1,2,4-7". The reason, why this parameter is not a flag within the settings of each channel, is the fact that many channels cannot be configured in the registry. This are all external or virtual channels. Nevertheless, TurboPlayer can generate play info messages for these channels, too, e.g. for a "running" text element. For prelistening a play info is never sent.

BCS can inform all interested clients about the currently playing elements. Therefor clients have to request play info notifications. DigAIRange will do that automatically whenever the column "Playtime / Remaintime" in the rundown view is visible. In TurboPlayer you can configure player windows for remote channels. These channels have the numbers 10000, 10001 and so on. They are assigned automatically by TurboPlayer, by always using the lowest free channel number. So far there is no possibility to restrict certain players to display only information from certain other TurboPlayers or certain rundowns or whatever. Such filter possibilities might be implemented in the future. Pay attention to restart TurboPlayer when you configure the first of these remote channels, because only then TurboPlayer will start to request play info notifications from BCS. Alternatively, you can change the program to activate getting play info. Afterwards you can add more remote players or remove them as you like. The old remote channel with number 9999 still exists in version 6.0, but as it has limited capabilities (only a single channel and only faked state and time values) it is recommended to only use the new remote channels.

If you have a setup of multiple BCS servers (e.g. a master/buddy pair or servers with remote links on show/track level) the servers will talk to each other to synchronize the information they have in memory. This means that play information is available for remote TurboPlayers, too. With two limitations: First, this works only as long as there are active remote links, which means that a user must have a show/track with a remote link open in a client. Second, the implementation transmits the play info only via one "hop" to other BCS servers but then not further on to more BCS servers. In other words: if you have a complex mesh of BCS servers, in which a remote BCS is also being linked by further BCS servers, this synchronization will not work.

Regarding the BCS and/or network load from the play info messages, you might want to have an eye on it. Nowadays, the network load should not be an issue. Play info messages have a typical size of 1 kByte. Assumed that you have 100 clients sending or receiving the messages, you get a load of 100 kBytes/s or 0.8 MBit/s. On a GBit port this is only a load of 0.1%. BCS itself handles the play info with as many threads as there are CPU cores on the system and independently from the regular tree/metadata operations. Therefore, no problem should be expected from this side, either. Nevertheless, if you start using play info in your system, you can monitor the CPU and the network load of BCS and see, whether there is any unexpected increase in the load.

Some sentences about the accuracy of the times. When play info is generated and when it is evaluated, all times are based on the computer clock with a function, which yields a precision below one millisecond. It is possible and recommended to synchronize the computer clocks quite well. With the standard Windows components, an NTP server and the correct settings an accuracy of 1 millisecond is achievable. For more information, please check this page from Microsoft:

https://docs.microsoft.com/en-us/windows-server/networking/windows-time-service/configuring-systems-for-high-accuracy

As the BCS clients display the playtime and the remaintime only with full seconds, a millisecond accuracy is not necessary, but something less than a half second should be the goal. If a client receives a play time info, he compares the timestamp in the message with the own computer clock. If the reported time is more than 2 seconds in the past or in the future, it is clear that the synchronization of the clocks does not work. In this case the client will use the reported play- and remaintime and will not try to correct the delay, which results from the transport of the message and from the internal handling in BCS. In this case the user will see play-/remain times, which might differ slightly from what is being displayed in the playing TurboPlayer and it will happen that the numbers are not counted consecutively.

The whole concept with play info messages will no longer work if there is too much delay for these messages, may it be in the network of by a too high load for BCS. In this case you must upgrade the performance of the system somehow or the play info will not be usable.

6.6 Monitoring TurboPlayer

Since version 5.0 TurboPlayer supports a function “GetHealthStatus”. It can be queried with TurboPlayerService, either via REST or via web sockets. The returned data can be either formatted in XML or in JSON. This helper function is intended to write something like custom “sensors” for a monitoring system like Nagios or PRTG.

Here is an example in format XML:

```
<?xml version="1.0" encoding="utf-16"?>
<TurboPlayerHealthStatus>
 <MessageType>TurboPlayerHealthStatus</MessageType>
 <KernelStartupTime>2016-04-12 17:25:10.016</KernelStartupTime>
 <ActivationMode>LiveAssist</ActivationMode>
 <ServerConnectionMode>Connected</ServerConnectionMode>
 <ServerConnectionState>Connected</ServerConnectionState>
 <NumberOfConnectedGUIs>2</NumberOfConnectedGUIs>
 <GUIs>
  <GUI>
   <Number>1</Number>
   <ConnectionState>Connected</ConnectionState>
  </GUI>
  ......
 </GUIs>
 <NumberOfPlayingElements>2</NumberOfPlayingElements>
 <NumberOfPausedElements>0</NumberOfPausedElements>
 <PlayingOrPausedElements>
  <Element>
   <ID>00000000/00000067/000007E1/0000000B/00000009/00000066/00000002/00000085</ID>
   <Title>Ready Teddy</Title>
   <Channel>1</Channel>
   <Line>1</Line>
   <Type>Internal/Audio</Type>
   <PlayState>Playing</PlayState>
   <PFL>0</PFL>
   <PlayTime>7456</PlayTime>
   <RemainTime>82640</RemainTime>
  </Element>
  ......
 </PlayingOrPausedElements >
 <LastPlayReportTotal>2016-04-12 17:33:21.160</LastPlayReportTotal>
 <LastPlayReports>
  <Report>
   <RundownListType>Show</RundownListType>
   <ReportTime>2016-04-12 17:33:21.160</ReportTime>
  </Report>
  <Report>
   <RundownListType>Jingles</RundownListType>
   <ReportTime>2016-04-12 17:33:21.040</ReportTime>
  </Report>
 </LastPlayReports>
</TurboPlayerHealthStatus>
```

Most of the fields should be self-explanatory if you know TurboPlayer well. Only two hints:

- Times fields which are pure numbers are in milliseconds.
- The “LastPlayReports” are either start, play- or remain-time reports from MultiPlayer to TurboPlayer. The time-reports are sent approx. twice per second. If the last report is too much in the past, something is wrong. The play reports are listed for each rundown, “total” is the newest one of all reports.

7 PARAMETERS

This chapter contains a description of registry parameters which are used by TurboPlayer. The content is identical to parts of the general document “DigParam.rtf”. Not all parameters are described, parameters which should be changed via GUI settings dialogs only, are mostly omitted.

Normally, you define most of the TurboPlayer parameters in the local registry. TurboPlayer first tries to read parameters from the local registry and only falls back to global parameters if nothing is found locally. (Hint: some few parameters are only read locally – e.g., the station name.) This allows to define all TurboPlayer parameters in the global registry. Since TurboPlayer 6.0 this is even supported better, because TurboPlayer v6 has some new features: extended definitions of available configurations with filters for specific computers, a new setting in the general settings/tab "Miscellaneous" to disable the saving of any changes (which is always done to the local registry) and extended command line parameters, which allows to select the TurboPlayer engine to connect to.

When you want to maintain the settings globally, it is recommended to have a computer on which you do all the administration - in the GUI and directly in the registry. On this computer you would have all the settings locally. This is necessary, because TurboPlayer will always store changed settings locally. If you are done, you can use Admin.exe to copy the parameters to the global registry. Trying to modify globally stores settings with the TurboPlayer GUI is not recommended and sometimes does not even work. Reason is that TurboPlayer always tries to save changes locally and this can give unexpected behaviour when global parameters are present, too. E.g., when you try to delete a globally defined GUI window, TurboPlayer removes its settings from the local registry. But as the global settings for this window still exist, the window will reappear when you restart TurboPlayer.

\TurboPlayer

This key contains all parameters for the TurboPlayer. Besides some values are read from DigAlRange, like the mixer sources.

AbsoluteStartOnHighChannels: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) When TurboPlayer selects a channel for playing an element, channels with lower numbers are preferred normally. This is also true for absolute/fixed-start elements. If you set this parameter to true, TurboPlayer will prefer channels with high numbers for absolute/fixed-start elements. This allows the operation that the presenter uses the lower channels for the elements he starts while he leaves the channel(s) with the highest number(s) free for TurboPlayer to start the fixed elements.

ActionOnVariableEnd: (Par_string, Possible values=Start, Skip, None, Default=None) This parameter is only evaluated for elements with start type "variable" / start mode "external" and for elements with "variable" / "manual" if TurboPlayer is in automatic mode. It defines the behaviour of TurboPlayer at the end of the allowed start interval, if no start-event/start-trigger was received. The following settings are possible:
- "None": the element is set to "floating" but it is not started.
- "Skip": the element is skipped.
- "Start": the element is started. (Valid since: 01.10.2005)

ActivationModes: (Par_string, Default=Passive, LiveAssist, Automatic) Defines which of the activation modes (Passive, LiveAssist, Automatic) are allowed. Activation modes which are not listed here cannot be set by the user or by a macro. (Valid since: 01.01.2004)

AllowMultipleActiveClients: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=TRUE) The standard behaviour is that all running TurboPlayers can be active (= mode "Live assist" or "Automatic") at any time. If you set this parameter to FALSE, only one TurboPlayer for each service can be active. The others automatically change to "passive". This parameter must be identical for all TurboPlayers of one service.

AlwaysSetSignalProcessing: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) By default an optimod/audio signal processing for a line is only set when the needed setting is different from the last known setting. If your mixer console has predefined settings, which are reset whenever a line is being closed, you must set this parameter to yes.

AutoDemute: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=TRUE) Defines the behaviour of TurboPlayer when an element with attribute "MuteStart" is still running mute and all previous elements have been played. If this parameter is TRUE, the mute element is demuted

automatically. That means, the fader is opened to the main level value of the element. If the parameter is FALSE, the fader remains untouched. (Valid since: 01.10.2005)
ButtonJingleMode: (Par_string, Possible values=Hot start, Pre select, Combination, Default=Hot start) This parameter specifies the behaviour of TurboPlayer when it receives start/stop commands from buttons (key shortcuts). Three modes are defined: (1) Hot start: Each key command will be executed as a start/stop. This is the default mode. (2) Pre select: Key commands will not be interpreted as start/stop but as a selection of the corresponding jingle instead. (3) Combination: The execution depends on the current state. If no jingle is running, a key command is executed as a jingle select. If at least one jingle is running, each command is interpreted as start/stop. (Valid since: 16.01.2006)
BWFLiveAutoStart: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) This parameter controls how elements with start attributes backward-floating and manual/sequenced are started in live-assist mode. By default they are _not_ started automatically - in contrast to automatic mode in which they are started automatically. If you want to have the same behavior in live-assist as in automatic mode, set this parameter to TRUE.
CartPlayInfo: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, <Filename>, Default=No) The TurboPlayerGUI can display clocks on top of cart elements to signal the user that a cart should not be played for some time. The information for these clocks is stored by the TP engine. With this parameter you can activated this storage which is the precondition that the GUIs can display a clock.

If you set this parameter to TRUE/Yes/1 the engine will assemble the play info for carts, but only in memory. After a restart the information will be lost. If you specify an XML filename instead, this file will be used to store the information and it will be available after a restart. In principle it is even possible to use a file shared by multiple TurboPlayer engines, but pay attention: the file access is done within one of the engine threads (TreeManager). If the access to the file is disturbed other operations might be disturbed, too. Therefore we recommend to use a local file only.
CheckSlots: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) Normally, TurboPlayer checks whether the sum of configured preloaded elements for all rundown lists does not exceed the number of available preload slots of MultiPlayer. This is useful as long as you are working only (or mostly) with audio or video elements. But if you have a lot of external or live elements which need not be preloaded by MultiPlayer this is sometimes an undesirable limitation. Especially for a reuse workflow with live elemnets on a metadata track it is desirable that all live elements are within the preload range of the corresponding rundown list. Therefore you can deactivate the slot check TurboPlayer performs at startup by setting this parameter to FALSE (or No or 0). Pay attention that you do this only if you can make sure by your setup/configuration that it cannot happen that TurboPlayer will try to preload more audio/video elements than MultiPlayer has slots. This would create permanent preload errors in the log and it is not defined, which of the elements get preloaded and which not. (Valid since: 12.07.2018)
ContentPollingInterval: (Par_int, Possible values=20, 21, 22, ..., Default=<empty>) With this parameter a regular polling for all loaded BCS content can be activated. An interval in seconds must be given which must be greater than 20. Normally this parameter should not be set ! TurboPlayer gets informed about metadata changes by BCS notifications. Only in a situation in which you encounter problems with the metadata updates within TurboPlayer you can try to activate the polling. Keep in mind that this creates a higher load for BCS. It does not harm if both, notifications and polling are active. If BCS build is 310 or higher, TurboPlayer will even try to reactivate all notifications whenever a data retrieval is done. (Valid since: 2016-12-05)
CPUWatchdogWarnTime: (Par_int, Default=0) This parameter controls the CPU watchdog. This watchdog is intended for debugging and measures, whether its thread can obtain CPU processing time. This parameter is the time in milliseconds, after which the watchdog writes a warning to the log when it did not get a CPU computing slot for the specified time. If you specify 0 (which is also the default) the watchdog is not active. Otherwise something like 10 ms is a good value. (Valid since: 16.8.2021)
EnableDoNotUseFlag: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) If an element cannot be preloaded, TurboPlayer can set the flag "DoNotUse" in the XML data of the element. This flag signals the BroadcastServer and other clients that the element cannot be played (probably). The BCS will ignore the element in its time calculation and possiblya gap will become visible. Sometimes you do not want to make use of this feature - typically in a multi-TurboPlayer enviroment, in which one Turbo can load the element and the other cannot. Then you should disable the feature by setting the parameter to No/FALSE/0. Otherwise you might get into the situation in which one TurboPlayer can load a file while another can't. This causes very fast load/unload operations and sets/resets of the flag which can heavily disturb the operations.

ExternalLookAheadTime: (Par_int, Default=0) This parameter is only evaluated for elements with a fixed start type and start mode "external". The value defines the interval (in seconds) before the planned start time, in which the element will be "hot". That means in this interval the element can be started by the corresponding in-event. Ouside of the interval the in-event will be ignored. (Valid since: 01.10.2005)
FileTimeout: (Par_string, Default=6000) Before accessing files or directories TurboPlayer makes use of a special function to check the availability. For this function a timeout can be specified. It is the time in milliseconds the function waits for the answer of the operating system (or a file server) before stopping the operation. Increase this value from the standard 6000 if your network or server is slow or if you encounter error messages that a file or directory could not be found, though this is not true. Hint for the experienced user: if you specify two comma separated timeouts, the first one is the time when the function stops the operation if at least a single file could be found (if multiple files are looked up simultaneously).
ForcedShowUnloading: (Par_string, Possible values=Yes, No, TRUE, FALSE, Auto, 1, 0, Default=No) Normally TurboPlayer does not unload an old show (even if the end time of the show has been passed) if there are still elements to be played or even playing in this show. Sometimes this behaviour is not desired. For a pure "slave" regio player it might be most important, to follow the shows in time - even if some elements have not been played. Therefore you can set this parameter to yes. Then TurboPlayer will skip all remaining elements of a show at the end time of the next show, followed by an unloading of the eldest loaded show. Pay attention that there is always a "next" and "previous" show being loaded ! That means, you must load at least one future show, one past show and all shows must exist. If no next show can be found on the BCS, TurboPlayer keeps the last known shows.
The parameter can also be set to "Auto" or "Automatic" which enables show unloading only for the automatic mode.
Since version 5.1 you can also write something like "Main+5", meaning: unload the main show 5 minutes after its end. If you use this sort of specification it is not needed to have past shows loaded, the ShowInterval can be given without a negative show number. You can also add "Auto" (e.g. "Auto,Main+5") to make it active in Automatic mode only.
IgnoreMarkOutClasses: (Par_string, Default=<empty>) With this parameter you can define a (comma-separated) list of classes. Elements which have one of the specified classes are played as if the "Ignore mark-out" flag was set. This is typically useful for the class "text" if you also "play" text elements (by adding "Text" to the parameter "LiveMediaTypes"). This assures that the virtual play counter continues until the reading person stops the element manually. Otherwise the element would be stopped automatically at the planned duration. Check also the parameter "IgnoreMarkOutMediaTypes".
IgnoreMarkOutMediaTypes: (Par_string, Default=<empty>) Like the parameter "IgnoreMarkOutClasses" but here you specify a list of media types. We recommend to use only one of both parameters because the mark-out is ignored if either the class of the media type says the mark-out should be ignored.
InitByEngine: (Par_string, Possible values=Connect,SetProgram,LoadCurrentShow,LoadCurrentOrNextShow, Default=<empty>) Normally (without this parameter being defined), some steps during startup are triggered by the first GUI, which connects to a TurboPlayer engine. Since version 6.0 it is also possible to let the engine itself (also when running standalone) do these steps. Therefore, list the steps in a comma-separated list within this parameter. Possible keywords are:
Connect, SetProgram, LoadCurrentShow or LoadCurrentOrNextShow
The first one performs the connect to the configured BCS (parameter ServerToConnect). The second one selects a program (parameter InitialProgram). The third one tries to load the current show. If no current show exists, nothing is loaded if you use the value LoadCurrentShow. In order to make TurboPlayer scan for the next show in the future, use the value LoadCurrentOrNextShow.
Please have in mind that higher steps require all lower steps, e.g loading the current show is only possible if a BCS connect was done first. (Valid since: 31.05.2021)
InitialActivationMode: (Par_string, Possible values=Passive, LiveAssist, Automatic, Default=LiveAssist) Defines the activation mode after startup and after changing the service.
InitialFreeShowlistMode: (Par_string, Possible values=Yes, No, TRUE, FALSE, 1, 0, Default=0) Sets the initial value of the free-showlist-mode. Due to timing reasons it is difficult to set this mode in an internal startup event. Therefore this parameter exists.
InitialProgram: (Par_string) This program will be selected at startup. If empty no program is selected. Parameter is easy to set in the GUI (->system menu ->settings | BCServer).

KeepTextFilesOpen: (Par_int, Possible values=0, 1, Default=1) Value = 1: If you are opening a text file to edit it inside TurboPlayer this file will be kept open exclusively for the one user; means it cannot be edited by others. It will be shown as read only for them.
KernelHasFullData: (Par_string, Possible values=FALSE, TRUE, No, Yes, 0, 1, Default=FALSE) If this parameter is false (which is the standard) the kernel receives only a small part of the element data from the TreeManager. It receives only the necessary fields for the preloaded elements. In case you want to use macros which set or query data, this might be a problem, because the macros are executed by the kernel. Therefore you can set this parameter to true and the TreeManager will send all XML data to the kernel. This includes all XML fields, not-prelaoded elements and groups.
KickOutWaitTime: (Par_int, Possible values=10-1000, Default=250) Sometimes when TurboPlayer wants to start a new element, there is no free channel. If it is allowed to stop one of the running channels (so called kick-out), TurboPlayer stops a channel, waits for some time and then tries again to start the element. The wait time can be defined with this value in milliseconds and must be bigger than the time needed for the stop process (must be evaluated from the logfiles). This parameter/time also applies when a channel is playing in prelisten/PFL mode and has to be stopped before an element can be played on-air.
LiveClasses: (Par_string, Default=<empty>) This is a deprecated parameter which is only supported up to TurboPlayer 4.x versions. In newer versions you must use the parameter "LiveMediaTypes". Nevertheless this is a recommendation for older versions, too.
This parameter defines whether elements of the specified classes are played as "live elements".
LiveMediaTypes: (Par_string, Default=Live,Control) Here you can specify a (comma-separated) list of media types. Elements which have one of these media types are played as "live element". This means: on a virtual channel with playtime info only. For more information see the chapter about "Playout categories" in TurboPlayer-TechManual.
Since version 5.x of TurboPlayer the default for this parameter is "Live,Control". We strongly recommend to always specify these two media types, even if you add other types. Reason is that it is the expected behaviour of TurboPlayer to play these media types as live elements.
This parameter is typically used to enable a simluated playout of text elements. But pay attention: For text elements this parameter allows to "play" them and see time information in the player window. But for elements with a media file it means they will be "played" virtually only, the audio/video is not really started. (Hint: if you do not want a text element to stop automatically at the planned duration see the parameter "IgnoreMarkOutClasses" too.)
LocalClipDir: (Par_string) This is the directory which is used by TurboPlayer for files of clipboard elements (typically elements in a stack which is configured in clipboard mode). It is also used for new recordings of OnAIR TrackMixer. If the specified directory does not exist TurboPlayer will create it. If nothing is configured, TurboPlayer will call the GetTempPath function of Windows (which typically returns something like "C:\Users\...\AppData\Local\Temp") and create a "TurboPlayer" subdirectory. On program termination TurboPlayer will delete all files in the directory.
LoudnessSetForNoBCS: (Par_string) Used when a Loudness Set is desired to be used, if the option /noBCS is given in command line. It will only be considered if this option is used, else the AssignedLoudnessSet will be used as usual.
MakeSnapshotOnResyncToBCS: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=TRUE) When TurboPlayer reconnects to BCS, a resync between TP and BCS takes place. In case of a lost connection or when switching from connection mode "Standalone" to "Connected" TurboPlayer overwrites the data in BCS. Before doing that, TP normally creates a snapshot of an overwritten track.
This can be a problem if you have a very unstable connection. In this case a big amount of snapshots will be created. You can set this parameter to FALSE in order to prevent TP from creating these snapshots. (Valid since: 16.02.2021)
MaxSkipFailed: (Par_int, Default=9999) In automatic mode TurboPlayer skips elements if he fails to open active lines. (If necessary, more than one line is tried for each element, see parameter ShiftDefectLine.) With this parameter you can limit the number of elements which are skipped. 0 means: no element is skipped at all, 1 means: one element is skipped and so on. An internal counter counts the failed starts and if the limit is reached, the current element is not skipped but instead set to manual/floating and no further starts are tried. Of course this means TurboPlayer will stop playout but this case typically occurs only when the mixer console or the communication to the console fails.
MinRunTime: (Par_string, Default=0) Normally, an element is considered as "sent", if it has been played for a millisecond only. You can change this, by specifying a minimum runtime. Elements, which have been played for a time, shorter than the minimum runtime, are considered as not being sent. After

the stop, the send state will be reset to its original value (planned or cleared) and the real times will be erased. If the element has been inserted automatically (due to an unexpected opened external line) the element will be erased. If you enter a pure number it is the minimum runtime in milliseconds. Since build 1100 you can also enter a percentage (e.g. "10%") which makes the minimum runtime relative to the planned duration of each stopped element. (Valid since: 01.05.2004)
MoveFaderAfterFaderStart: (Par_string, Possible values=true, false, Yes, No, 1, 0, Default=false) If you are using TurboPlayer with a mixer console with active motor faders there might be a problem. If a user starts an element with a fader-start (which means: by grabbing the fader and moving it up) then it can happen that TurboPlayer wants to handle fades via controlling the motor of the fader, whileas the user has his finger at the fader and works against the motor. Therefore, TurboPlayer does not try to move the fader in such a situation by default.
As TurboPlayer does not know when the user has removed his finger from the fader, a simple (and not always fitting logic) has been implemented. Turboplayer will not execute any fades within the first half of an element. The fades in the second part and at the end of the element will be executed. This is the default behaviour, which can also be achieved by setting this parameter to false.
If you set this parameter to true, TurboPlayer will always try to execute all fades. This might mean that the user has to work against the fader movements of TurboPlayer, if necessary, but this setting is especially useful if you have a mixer console, which has extra line open/close or on/off buttons and the users normally have all faders open and start elements by pressing the open/on button on the console. (Valid since: 14.02.2023)
MuteStartOnClosed: (Par_string, Possible values=Yes, No, TRUE, FALSE, 1, 0, Default=No) Normally TurboPlayer prefers open lines to play elements, or he opens the lines before an element starts - in case of mute-start elements only to a minimal level. If you set this parameter to Yes/True/1 TurboPlayer can use a closed line for starting a mute-start element. Nevertheless TurboPlayer prefers open lines. The parameter does apply only for "passive" lines, other lines can either be opened or their state is ignored (none).
MuteStartSkipsPrev: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) When an element starts muted previous planned elements are not skipped as it happens when an element starts audible. If you want the same behaviour for a mute/audible start (that the previous elements are skipped) set this parameter to TRUE/Yes/1.
OnAirPFLElement: (Par_string, Possible values=Next, Cursor1, Cursor2, ..., Default=Next) When PFL is activated for an empty on-air channel TurboPlayer picks out the <next> element for prelistening normally (or one of its successors if the <next> element is already running). This is the standard behavior which can be achieved with the setting "Next" or without defining this parameter. TurboPlayer can also prelisten one of the cursor elements. Each GUI has its own cursor but the mixer console exists only once. Therefore you must define which GUI/cursor is used for the selection of the prelisten element. Set this parameter to "CursorX" where X is the desired GUI number.
Password: (Par_string) Encrypted. The password to connect to BCServer.
Password can only be set in the GUI (->system menu ->settings | BCServer).
PreferSameLineForStartModes: (Par_string, Default=<empty>) This parameter is only relevant if you have a configuration with multiple channels per line/fader. The parameter controls how TurboPlayer picks a channel for playing an element. It can either prefer a free channel for the same line as the previous element is being played, or it can prefer a different line while ignoring the other channels of the playing line. The first behaviour might be useful if only a single fader should/must be used for playing a whole transition with all main elements and drop-ins on the same line. The second behaviour is useful to be able to control each element individual with its own fader. In this parameter you must list all start modes for which the first behaviour should apply in a comma-separated list. Possible start modes are: Manual, Sequenced, StartOnTime, EndOnTime, External, Relative.

Be aware that the parameter says "Prefer...". TurboPlayer will try to select a play channel according to this parameter but there might be a lot of other restrictions or side-effects which make a different channel more appropriate. (Also check the parameter "SequencedOnSameLine" which is the same/old setting but only for start mode "Sequenced".)
PreferSystemElementStart: (Par_string, Possible values=TRUE/Yes/1, FALSE/No,0, Default=No) By default TurboPlayer starts only the element in a rundown which is marked as <next element> when the user opens a fader or presses a start button. Sometimes an external or live element is the next one (possibly with some more external/live elements below) but the user wants to start the next internal element somewhere down in the rundown. Normally he has to skip the unwanted elements before the internal element can be started. If you set this parameter to true/yes/1 the user can simply open an

internal fader or press a start button for an internal channel and the next internal(=system) element will start. The unplayed elements above will then be skipped as usual.
PrelistenStop: (Par_string, Possible values=End, MarkOut, Default=End) By default TurboPlayer stops the internal prelisten at take-end. If you want to stop prelisten at mark-out, you have to set this parameter to "MarkOut". Be aware that the parameter is only valid for internal prelistening via MultiPlayer. CrossfadeMixer and EasyPlayer have their own logic.
PrepareTimeAbsolute: (Par_int, Default=3000) If there is a free channel and if there is enough time TurboPlayer prepares elements to a channel before they are really started. The standard time is 3000 ms before the start point. You can change this interval but do not make the value too big because this will result in disadvantages like a higher need for free channels or astonishing reactions on user input if an element is already prepared. This parameter applies to all elements with an absolute start (start-on-time, start at the end of a variable interval or other combinations in automatic mode).
PrepareTimeRelative: (Par_int, Default=3000) If there is a free channel and if there is enough time TurboPlayer prepares elements to a channel before they are really started. The standard time is 3000 ms before the start point. You can change this interval but do not make the value too big because this will result in disadvantages like a higher need for free channels or astonishing reactions on user input if an element is already prepared. This parameter applies to all elements with a relative start (sequenced, relative or other combinations in automatic mode).
PrepareTimeTransition: (Par_int, Default=500) This parameter is a specialization of the PrepareTimeRelative. It is applied to elements within a transition beginning with the second one. This parameter has no effect if you do not have transition lines defined or if you do not have elements which are flagged with Time_StartOnTransitionChannel=1. The idea of this parameter is to be able to reduce the regular prepare time to be able to play and prepare as much elements as possible on the transition channels. Therefore this time should be very small but big enough that MultiPlayer can really prepare the element within the specified time. The unit is milliseconds. (Valid since: 2017-10-11)
ProtocolTrack: (Par_int, Default=1000) The number of the show track where TurboPlayer inserts all played elements. (Valid since: 01.01.2004)
RemotePlayMode: (Par_string, Possible values=No, Yes, Delink silent, Delink ask, Default=No) Specifies the behaviour when the user tries to start an element within a remote show or track. These values are possible: "No" means: a start is not possible, an error message is displayed. "Yes" means: the remote link is ignored, the element is played like a local element. "Delink silent" means: when the first element of a remote show/track is started, TurboPlayer automatically removes the remote link. The show/track will be played from the local server and changes are no longer reflected from/to the remote system. "Delink ask" means: a message box appears, telling the user about the remote link and asking, whether he wants to remove the link. The user can either remove the link, or starting elements is not possible.
ResetOnShowChange: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=TRUE) By default TurboPlayer performs a reset whenever the main show is changed explicitly by the user. If this is unwanted, you can deactivate this behavior by setting the parameter to FALSE. This can be used e.g. to have a continuously running jingle during the show change. It is also useful setting this parameter to FALSE when you are working with a free showlist and want to add shows to the list during the runtime of TurboPlayer.
ResetPauseOnClose: (Par_string, Possible values=0, 1, TRUE, FALSE, Yes, No, Default=0) Normally, closing a line while pause mode is active will pause the element playing on the closed line, but the pause mode will stay on, so you can pause more elements by closing other lines. If you want to remove the pause mode whenver a line is being closed and an element is paused, set this parameter to 1 (TRUE, Yes).
ResetPauseOnStart: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) Normally, the start of an element will not change the pause mode. If you set the parameter to TRUE, any start of an element will reset the pause mode - if this mode is active. (Valid since: 01.05.2004)
RetryCloseLineCommands: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) The watchdog of TurboPlayer can check pending line commands (move fader commands). Sometimes a mixer console does not respond to a close line / fader command. In case the user can keep faders open, this is not an error and it is not helpful if TurboPlayer retries to close the line. This is the standard behaviour or the behaviour if the value is "No". On the other side, if the user cannot hold the line open, there might be a communication problem with the mixer console. In this case it could be useful to re-send the close line command. (Valid since: 10.02.2006)
SendPlayInfoForChannels: (Par_string, Default=<Empty>) Since version 6.0 TurboPlayer can generate play info messages for playing elements. In this parameter you can configure for which

channels these messages are generated. The parameter can be the joker "*" (a star) in order to send messages for all channels. Otherwise, the value is a comma-separated list of channel numbers, but you can also use a range. Example: "1,2,4-7". For more information please see TurboPlayerTechManual chapter 6.5 "Play Info". (Valid since: 1.7.2021)
SequencedAcrossPlayed: (Par_string, Possible values=Passive,LiveAssist,Automatic) Attention: This setting is related to a deprecated feature! In newer versions, beginning with 5.4.2085.0, this feature should no longer be used! Instead the automatic assignment of the overlaid flag (->mode "AssignOverlaid") should be used!

By default a sequenced element is only started directly behind the previous normal element. This might be a problem if you have a chain of a long element playing, a short element played and a sequenced element waiting to be started. The third element might not start because the element in the middle is not running. But if this parameter is activated, TurboPlayer will skip across the played element when examining a sequenced start.
You can activate this behaviour independend for every activation mode. Write a comma separated list with the modes for which the sequence evaluation should look across played elements (the first letter of the mode name is sufficient, e.g. "L,A").
Default value was "Automatic" up to version 4.7.1606.0. Since version 4.7.1607.0 the default is an empty string => the feature is no longer active by default.
SequencedAtIgnoredMarkOut: (Par_string, Default=Continue) Suppose there is a running element (1) which has the flag "IgnoreMarkOut" set (or the general IgnoreMarkOut mode of TurboPlayer is being set) and the next element (2) has start mode "sequenced". When the mark-out of element (1) is reached, there are now four possible behaviours which can be set with this parameter:
- Continue: Element (1) continues playing, nothing else happens. Element (2) will be started when the take-end of element (1) has been reached. This is the default, because this behaviour is what "ignore mark out" says.
- Continue+Start: Element (1) continues playing, element (2) is started.
- Stop+Start: Element (1) is stopped, element (2) is started. This setting means, the "IgnoreMarkOut" flag is not obeyed if a sequenced element follows.
- Fade+Stop+Start: Element (1) is faded out and stopped, element (2) is started. This is like before, but a fade out is performed even if the general "PlayFadings" mode is off.
Be aware that a looped element is treated like an ignored mark-out and this parameter applies too. This parameter has no function in automatic mode because it is designed to work for live assist.
SequencedOnSameChannelClasses: (Par_string, Default=<empty>) It is possible to play a sequenced chain of elements of a specific class on a single channel. Typically this is desired for commercials. Enter a comma-separated list of the classes which should be played on the same channel as the first started element. Example: SequencedOnSameChannelClasses=Commercial. If you specify multiple classes here, this does not mean that these classes are mixed. Only elements with the same class as the previous element are played on a single channel.
SequencedOnSameLine: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) This parameter is only relevant if you have a configuration with multiple channels per line/fader. The parameter controls how TurboPlayer picks a channel for playing a sequenced element. It can either prefer a free channel for the same line as the previous element is being played (=TRUE) or it can prefer a different line while ignoring the other channels of the playing line (=FALSE). The first behaviour might be useful if only a single fader should/must be used for playing a sequenced chain. The second behaviour is useful to be able to control each element individual with its own fader.
Hint: this parameter is deprecated and was replaced by parameter "PreferSameLineForStartModes" in build 1605.0. In old versions the default for this parameter was TRUE, since 5.2.2007.0 it is FALSE.
ServerToConnect: (Par_string) The name of the BCS server.
Parameter is easy to set in the GUI (->system menu ->settings | BCServer).
SetNextCursorToOverlaid: (Par_string, Possible values=Yes, No, TRUE, FALSE, 1, 0, Default=Yes) Overlaid elements are elements which are intended to be played during other, longer elements - e.g. a long music bed with overlaid jingle drop-ins. If it should be possible to start these overlaid elements manually, it is necessary that the <next> cursor of TurboPlayer positions itself on these elements (=yes=default value). If you set the parameter to "no" the <next> cursor will be positioned on the first normal (=non-overlaid) element behind the chain of overlaid elements. This allows the user to start the next normal element without the need to skip the overlaid elements first.

Details: This parameter applies mostly to the "Sequenced" rundown mode, because in all other modes the <next> cursor can be set by the user as he likes. The parameter applies only to overlaid elements with start modes "Manual" or "Sequenced", but not for "Relative" starts. If you do not want to start overlaid elements manually, there is no need to position the <next> cursor on overlaid elements.
ShiftDefectLine: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) If you have active lines, TurboPlayer will try to open a line before an element can be started on this line. Now it is possible that a line cannot be opened, e.g. because it is defect or because the corresponding fader is blocked. It is a feature of TurboPlayer to try up to 3 times to open a line. If it a line cannot be opened, TurboPlayer can change to another possible line and will try to open this line. Again this line is tried up to 3 times. If the second line does not work too, TurboPlayer gives up and the element will not be played. The change to the second line wil be done only, if this parameter is set to TRUE. (Valid since: 01.10.2004)
ShiftInvalidChannelStart: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) If you have configured a class-channel assignment or an endpoint restriction for channels (->parameter NeutralEndpointOnly), the user cannot select each channel to play an element of such a class/endpoint. Elements of the specified class can be played on the assigned channels only and on a channel with NeutralEndpointOnly=true only neutral elements within an EndpointStory can be played. If you set this parameter to TRUE, it does not matter for which channel the user generates a start command (e.g. by pressing a start button), TurboPlayer will automatically change to an allowed channel. If the parameter is FALSE, an error message will be displayed instead and the user cannot start the element unless he selects the correct channel. (Valid since: 01.05.2004)
ShiftToLiveChannel: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) If this parameter is set to TRUE, the user can start a live element by opening a wrong fader - e.g. an audio fader. TurboPlayer will automatically shift the start of the live element to the first free live channel. Be warned that this is only true for the start. Stopping a live element is not possible by closing an audio fader.
This parameter is automatically set (=true) if the audio playout / MultiPlayer has been deactivated.
ShowFilter: (Par_string) Normally TurboPlayer sees all shows defined on a BCS. You can limit the visible shows by defining a show filter in this value. This is done by specifying an expression according to TurboPlayerTechManual chapter 4.5.
If the expression yields true or 1 for a show, the show is visible. Alternatively you can define functions. In this case the last defined function must be named "filter" and it must return true or false. When true is returned, the show is visible.
You can use the special variables %fieldname (e.g. %Name) to retrieve a field of each evaluated show. The same is achieved with the function Data(fieldname) or DataOfShow(fieldname). In addition DataOfProgram(fieldname) allows to get data of the current program/service. There are also 4 global variables pre-defined, which can be used for comparisons. These are: $ComputerName, $StationName, $StudioName, $UserName.
If you specify the value "*" as first parameter of one of the functions: strcmp, strsearch, regex_match or regex_search, then these functions look into all existing fields of the show.
Hint: the show filter is implemented independent of the general macro processor sitting in the kernel. Therefore, you cannot make use of any of the many functions, procedures or variables, which are available in the regular/kernel macros. (Valid since: 1.8.2022)
ShowFilterFile: (Par_string) If the expression or the functions for the parameter "ShowFilter" are too long to be reasonably be stored directly in the registry, you can alternatively use this value and specify a text file, which contains the expression or the functions for filtering shows. (Valid since: 1.8.2022)
ShowInterval: (Par_string, Default=-1,1) This parameter defines the number of shows which are loaded by TurboPlayer. You must specify two values: a negative one, which says how many shows of the past are being loaded and a positive one, which says how many shows of the future are being loaded. With the standard value "-1,1" three shows are loaded: the current, the past and the next show. (Valid since: 01.01.2004)
SkipUnusedReserve: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=TRUE) TurboPlayer will automatically skip reserve elements if they were originally scheduled but are not needed anymore during playout. In rare cases this behaviour might lead to an unwanted skipping of elements, typically in a setup with multiple active TurboPlayers. Then you encounter this situation you can disable skipping of unneeded reserve elements in all TurboPlayers except the main one with this parameter. (Valid since: 8.5.2018)
SlavedShowLoading: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) If set to true, TurboPlayer will automatically load the same shows as a remote TurboPlayer. That means, you

can setup a bunch of connected TurboPlayers. If in one of them a new show is being loaded by a user, all others with this parameter set to yes will follow. This is useful for pure regio-TurboPlayer which have to follow a main TurboPlayer.Hints: (1) This feature requires that the TurboPlayers either have a direct connection or they can communicate via generic BCS broadcast messages. (2) The Turbos must have the same program/service loaded and the start time of the loaded show must be identical on all systems. (3) This system even works, if you set this parameter for all TurboPlayer to yes.
StandardDemuteTime: (Par_int, Default=500) Defines the standard fade-in time used by TurboPlayer when demuting elements which were started muted.
StartOfExternal: (Par_string, Default=EventIn) There are different sources which can generate a start or a stop command. With this parameter you can specify, which of them will be executed for elements with start mode "external". The possible sources are: GUI, Line, Button, EventIn, EventOut. (Line means: fader start, Button means: key shortcut, EventOut means: generic events, macros etc.) If a source is not listed in this comma-separated list, it cannot generate start/stop commands. Commands of additional sources, like the timer or the internal logic, cannot be configured. If you specify a star '*' it means: all sources can generate start/stop commands.
StartOfSyncStart: (Par_string, Default=*) See the description for "StartOfExternal". This parameter is valid for all elements with start mode SyncStart. (Valid since: 15.11.2022)
StartOfTimestart: (Par_string, Default=*) See the description for "StartOfExternal". This parameter is valid for all elements with start mode StartOnTime or EndOnTime.
StartStopAutomatic: (Par_string, Default=*) There are different sources which can generate a start or a stop command. With this parameter you can specify, which of them will be executed in automatic activation mode. The possible sources are: GUI, Line, Button, EventIn, EventOut. (Line means: fader start, Button means: key shortcut, EventOut means: generic events, macros etc.) If a source is not listed in this comma-separated list, it cannot generate start/stop commands. Commands of additional sources, like the timer or the internal logic, cannot be configured. If you specify a star '*' it means: all sources can generate start/stop commands. (Valid since: 01.10.2005)
StartStopLiveAssist: (Par_string, Default=*) See the description for "StartStopAutomatic". (Valid since: 01.10.2005)
StartStopPassive: (Par_string, Default=*) See the description for "StartStopAutomatic". In addition to the other modes you can specify "SyncStart" if you want to allow synchronized and/or relative starts in passive activation. (Valid since: 01.10.2005)
StudioName: (Par_string) This name will be used by TurboPlayer as the studio name in protocols. For example it is used to fill the field "OnAirPlayedByStu" of played elements, inserted into the protocol track (Valid since: 01.10.2005)
SubclipLayersToStart: (Par_string, Possible values=0,1,2,3,4,---, Default=<empty>) Within a reuse workflow with live elements on a metadata track which are started while playing a recording, TurboPlayer will convert subclip information of the recording element into the embedded starts. With this parameter you can configure which subclip layer (one of the fields of each subclip) should be treated as an embedded start. By default this parameter is empty which means: treat all subclips as possible embedded start. ("Possible", because they must have an EmbeddedStartID, too.) Otherwise, you can define a comma-separated list of layer numbers for the layers, which should be treated as embedded start. (Valid since: 12.07.2018)
SyncPlayOptions: (Par_string, Possible values=Strict,Pause,Stop, Default=<empty>) This parameter allows to specify some details for the SyncStart of elements. You can give a list of comma-separated keywords. 3 keywords are available:
- Strict: If "Strict" is set, TurboPlayer will only start elements synchronized, which have the start mode "SyncStart". Without this option TurboPlayer can start any element synchronized when the TurboPlayer mode SyncStart is active and elements have the same content in the specified SyncStartField. For more details see TurboPlayerTechManual chapter 5.
- Pause: If "Pause" is set, elements, which were started with a SyncStart, are also paused synchronized. The same is true for a restart after a pause.
- Stop: If "Stop" is set, elements, which were started with a SyncStart, are also stopped synchronized, when the stop was triggered explicitly/manually by the user. If a master element is stopped regularly (by running till the intended end) the stop is not synchronized. In this case the slave elements will run till their intended end, too. (Valid since: 31.10.2022)
SyncStartField: (Par_string, Default=<empty>) Old value, replaced by -> BroadcastSystem\SyncStartField
TimeOut: (Par_int, Possible values=0...60000, Default=10000) Timeout value in msec for tree operations on BCS.

Parameter is easy to set in the GUI (->system menu ->settings | BCServer).
TimeOutRequestInfo: (Par_int, Possible values=0...10000, Default=2000) Time in msec the TP waits at startup to get full info about all states (e.g. selected program, show, etc.) before it makes its own selection of initial program and show due to daytime.
Parameter is easy to set in the GUI (->system menu ->settings | BCServer).
TimeoutSetActivationMode: (Par_int, Possible values=0...100000, Default=30000) Time in milliseconds the asking TurboPlayer waits for its request to change the activation mode from Passive to LiveAssist/Automatic. The new activation mode is pending as long as the request is not answered. The asked TurboPlayer shows a dialog to let the user react on the request (allow/deny). If the timeout is reached, the dialog is closed and the asking TurboPlayer is allowed to become active. If the parameter is 0, no dialog box is shown at all - the asked TurboPlayer changes immediately to passive. The Parameter is easy to set in the GUI (->system menu ->settings | BCServer).
User: (Par_string) User name for the login to BCServer; in combination with the password.
Easy to set by main settings dialog in the GUI (system menu -> settings).

\TurboPlayer\ChannelsOut

TurboPlayer is able to send out serial information to differ between music and text content. The subsections of this section contain the assigned classes for the different behaviour.

\TurboPlayer\ChannelsOut\...

Any name can be used for these sections like TurboMusic or TurboSpeech. This name is also used inside TIODiamond.dll configuration dialog to set the correct bit via the serial communication.
Class: (Par_string, Possible values=Audio,Music,Cart,Promotion,Magazine,Commercial,News) Assigned classes to a specific channel behaviour, separated by commas. (Valid since: 08.04.2008)
Name: (Par_string) Alias name for the channel behaviour. (Valid since: 08.04.2008)

\TurboPlayer\Communication

TurboPlayer is built from multiple internal parts. These parts use a messaging system for the communication. In this key you can configure parameters of the communication.
ComTypes: (Par_string, Default=InProc) This is a comma separated list of the supported communication types. Three values are defined so far: InProc (used inside of a single executable), MMF (=memory mapped filed, used between two executables on the same computer) and IP (=TCP/IP, used between two computers). You should activate only the communication types you are going to use, because there is a small overhead (startup time, memory, threads) if you activate types which are never used. (Valid since: 01.01.2004)
FullAccessForGUIs: (Par_string, Default=1) This is a comma separated list of GUI numbers which have full access to the engine. Full access means: every action which changes something. If a GUI is not listed, it has read-only access. (Valid since: 01.01.2004)
GenericMessageAuthentication: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=Yes) A part of the communication between TurboPlayers can be done with generic broadcast messages of the BCS. As this messages can be generated by any client, an authentication is possible. This is done via an encrpyted timestamp, which will be checked by the receiving TurboPlayer if this parameter is yes. To work, the system clock of all computers must be synchronized and must not differ more than 1 or 2 seconds.
GlobalNamespaceForMMF: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0) If you want to use memory-mapped files (MMF) for communication and if one part of the communication (e.g. TurboPlayerService) should run as Windows service, the MMF must be created in the global Windows namespace. This is necessary because the name of a MM file will only be visible within the same ring, meaning: a desktop file (in ring 3) will not be visible for the service (in ring 0) and vice versa. If you activate this parameter, a desktop program will need a special right to create a file in the global namespace. According to Microsoft the local group policy right "Create global objects" needs to be granted to the account running the desktop progam. We have noticed that this right has no effect. It is necessary to start the program with "run as administrator". Therefore we recommend to use IP for the communication if you have such a setup. (Valid since: 2016-06-29)

\TurboPlayer\Communication\GUI

-

GUINumber: (Par_int, Default=1) The number used by the GUI. GUI numbers start with 1 and there must not be two GUIs with the same number.

Attention: the specified GUI number must fit to the message box number of the RundownKernel/TreeManager. These two modules deduce the GUI number from the box used by the GUI. E.g. if you open ports 9981-9985 for GUIs then 9981=GUI1, 9982=GUI2, ... If you do not follow this rule connection problems might occur and the engine log will report an invalid GUI number. (Valid since: 01.01.2004)
PortForRundownKernel: (Par_int, Default=9979) This parameter is evaluated by the GUI and specifies the IP port number, the GUI opens for the connection to the RundownKernel. Hint: an extra port in addition to the address value is necessary because the IP connection for one port is used unidirectional only. This value need only be created if you run multiple GUIs on a single computer, otherwise the default value 9979 is fine.
PortForTreeManager: (Par_int, Default=9978) This parameter is evaluated by the GUI and specifies the IP port number, the GUI opens for the connection to the TreeManager. Hint: an extra port in addition to the address value is necessary because the IP connection for one port is used unidirectional only. This value need only be created if you run multiple GUIs on a single computer, otherwise the default value 9978 is fine.
RundownKernelAddress: (Par_string) This parameter is evaluated by the GUI and specifies the address of the message box, the rundown kernel has opened for the GUI. For a description of the address syntax see the chapter about comunnication settings in the TurboPlayer technical manual. (Valid since: 01.01.2004)
TreeManagerKernelAddress: (Par_string) This parameter is evaluated by the GUI and specifies the address of the message box, the tree manager has opened for the GUI. For a description of the address syntax see the chapter about comunnication settings in the TurboPlayer technical manual. (Valid since: 01.01.2004)

\TurboPlayer\Communication\IOManager

-
InitializeIOManager: (Par_binary, Possible values=TRUE, FALSE) This parameter specifies whether the TurboPlayer or TurboPlayerGUI should initialize the IO manager during startup. This is true, if you have IO modules which should be running. Sometimes this is not necessary, e.g. if you start a GUI-Only which connects to a remote engine and the GUI does not handle any IO modules.
ATTENTION: If you run a TurboPlayerEngine and a TurboPlayerGUI on the same computer this parameter must be set to FALSE! The engine initializes the IO manager independent of this parameter and a second initialization within the GUI (if the parameter would be TRUE) can produce strange results. (Valid since: 01.05.2004)
Port: (Par_int, Default=9990) This parameter is only needed for a standalone IOManager. If you use a full engine, the IOManager is included and only the "RundownKernelAddress" must be specified. This parameter is the local IP port number which is opened from the IOManager for the communication with the RundownKernel. Specifying a port number activates the server part of the communication (IOManager will open an IP port and will wait for incoming connect requests.) The client side can be activated at the same time with the parameter "RundownKernelAddress" which allows to start both sides in arbitrary order. Note that each port number can be used only once for exactly one connection ! (Valid since: 16.02.2006)
RunAsServer: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) With this parameter you can force the IOManager to behave as an IP server only. (It will open an IP port and will wait for an incoming connect request.) Normally this is not useful and the parameter is only present for backward compatibility. Standard behaviour is that both sides behave as server and client at the same time which allows to start both sides in arbitrary order. (See the parameter IOManager-Address in the communication settings of the kernel too.) (Valid since: 16.02.2006)
RundownKernelAddress: (Par_string) This parameter is evaluated by the IO manager and specifies the address of the message box, the rundown kernel has opened for the IO manager. For a description of the address syntax see the chapter about comunnication settings in the TurboPlayer technical manual. This parameter must always be specified, for a standalone IOManager as well as for a full engine including the IOManager. For the standalone IOManager it activates the client side of the IP connection (see also the parameter "Port"). (Valid since: 01.05.2004)

\TurboPlayer\Communication\RemoteTurboPlayers

Here you can define parameters for other TurboPlayers at remote locations. Each subkey represents one remote TurboPlayer. The local TurboPlayer can open a communication channel for each other player, which will be used for a multi-purpose communication between them.
SendBroadcastMessages: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) TurboPlayers can communicate to each other via direct connection, or via generic broadcast messages of the BCS. The direct communication can be configured with the subkeys of "RemoteTurboPlayers". The use of broadcast messages can be activated with this value. Normally one of both ways is sufficient, whereas the direct communication is faster. It is possible to activate both: the direct communication and broadcast messages, depending on the message TurboPlayer will decide which way to use better, in other cases it will ignore messages which are recieved twice on both ways.
SendLoadStateMessages: (Par_string, Possible values=true/yes/1/false/no/0, Default=false) By default, messages about the load state (preloaded, unloaded, load failed) of elements are not sent to remote TurboPlayers. This can be activated by setting this parameter to true/yes/1. In addition a pre-requirement is that sending messages for a specific rundown was activated with the parameter SendMessagesForRundown (in this key, too).
This information is not needed by remote TP engines, but it is needed by a remote TurboPlayerGUI if it should visualize the load state of remote elements in the column "Play info" of a rundown view.
Hint: the play info itself, like play and remain time, is being distributed via BCS and it does not require sending load state messages. The load state messages only give extra information to the receiving TurboPlayer. For activating sending play time info see the parameter: TurboPlayer\SendPlayInfoForChannels. (Valid since: 31.10.2022)
SendMessagesForRundown: (Par_string) TurboPlayer can send messages to remote TurboPlayers which inform them about started elements, next item changes or loaded shows. This is only necessary if the remote TurboPlayers depend on this information. For example if they have to start elements with a relative start linked to elements on the local system, they must be informed about starts. With this parameter you can configure for which rundowns these messages are being sent. Enter a comma-separated list of rundown names, e.g. "Show,Jingles". If you leave the parameter empty, no messages are sent. If the parameter does not exist, messages for all rundowns are sent.

\TurboPlayer\Communication\RemoteTurboPlayers\...

The name of this section is arbitrary but should of course reference a relevant label that may make sense to the user as to where the remote instance is to be found.... = Name of the new connection, e.g. "TP_City_2"
Active: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=Yes) With this value the communication to the remote TurboPlayer can be deactivated.
Address: (Par_string) Specifies the address of the other side. It must have the format "IP-Address=[dnsname or IP number]:[portnumber]". (Only IP-Addresses are allowed, a communication via InProc or MMF is not possible.) The TurboPlayer on the other side must have the corresponding port opened (see the port parameter). If an address is specified, TurboPlayer behaves as client. It tries to connect to the port of the other side. This is done permanently. Therefore it does not matter, which side is started first. The same is true if the connection is disturbed, a background thread tries to restore it. If only a port number (and no address parameter) is specified, TurboPlayer behaves as server and opens only a listening port. It even works to enter an address parameter on both sides, but normally you should make the central TurboPlayer (or the one with connections to more than one TurboPlayer) the server.
Name: (Par_string) This is the name which will be displayed to the user. If empty or not present, the key name will be used instead.
PingInterval: (Par_int, Possible values=1, 2, 3, ..., Default=5) Ping messages (a sort of live beat) are sent to each connected TurboPlayer to check the connection. Here you can specify the interval between two pings in seconds.
Port: (Par_int, Default=9940 + <key enumeration number>) The port number of the local (=own) port, which is opened for the communication. This port number must be specified in the Address-parameter of the other side. Pay attention not to use a port number which is used for anything else on the computer, especially do not use a single port number for multiple remote TurboPlayers !

\TurboPlayer\Communication\RundownKernel

-
CloseMultiPlayerOnExit: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) Controls whether MultiPlayer is closed during shutdown.

IOManagerAddress<n>: (Par_string) This parameter is only needed, if you want to run a standalone IOManager. This parameter is evaluated by the RundownKernel and specifies the address of the message box, the IOManager has opened for the RundownKernel. <n> is the number of the IOManager, started with 1. For a description of the address syntax see the chapter about communication settings in the TurboPlayer technical manual. (Valid since: 16.02.2006)
MultiPlayerComputer: (Par_string, Default=.) Here you can configure the name of the computer which hosts the MultiPlayer. If TurboPlayer is installed on the same computer you can enter "." (Valid since: 30.04.2010)
MultiPlayerNumber: (Par_int, Default=0) This parameter specifies the number of the MultiPlayer to use. Hint: you can configure multiple MultiPlayer settings and start them at the same time. Each setting has an assigned number. (Valid since: 01.05.2004)
PortsForGUIs: (Par_string, Default=9970-9970) This parameter is evaluated by the rundown kernel and specifies the number of messages boxes and/or the IP port numbers, the rundown kernel opens for the GUIs. For each GUI which wants to connect, you must have a message box. Attention: though the parameter name contains a "ports", you must specify this value, even if you do not use TCP/IP at all. This is necessary, because the rundown kernel deduces the number of needed message boxes from the specified range. The format is: "[minimum port number]-[maximum port number]". The example: "9970-9971" opens two message boxes. Maximum number of GUIs is 25 (since build 5.4.2084.0, prior to that it was only 10). (Valid since: 01.01.2004)
PortsForIOManager: (Par_string, Default=9980-9980) This parameter is evaluated by the rundown kernel and specifies the number of messages boxes and/or the IP port numbers, the rundown kernel opens for the IO manager. For each IO manager which wants to connect, you must have a message box. Attention: though the parameter name contains a "ports", you must specify this value, even if you do not use TCP/IP at all. This is necessary, because the rundown kernel deduces the number of needed message boxes from the specified range. The format is: "[minimum port number]-[maximum port number]". The example: "9980-9981" opens two message boxes. Maximum number of IOManagers is 25 (since build 5.4.2084.0, prior to that it was only 10). (Valid since: 01.01.2004)

\TurboPlayer\Communication\TreeManager

-
CloseEngine: (Par_int, Possible values=0,1, Default=0) This parameter is only evaluated by a pure GUI. If set to 1 the TurboPlayerEngine is closed together with the GUI.
PortsForGUIs: (Par_string, Default=9960-9960) This parameter is evaluated by the tree manager and specifies the number of messages boxes and/or the IP port numbers, the tree manager opens for the GUIs. For each GUI which wants to connect, you must have a message box. Attention: though the parameter name contains a "ports", you must specify this value, even if you do not use TCP/IP at all. This is necessary, because the tree manager deduces the number of needed message boxes from the specified range. The format is: "[minimum port number]-[maximum port number]". The example: "9960-9961" opens two message boxes. Maximum number of GUIs is 25 (since build 5.4.2084.0, prior to that it was only 10). (Valid since: 01.01.2004)
StartEngine: (Par_int, Possible values=0,1, Default=0) This parameter is only evaluated by a pure GUI. If set to 1 the TurboPlayerEngine is started together with the GUI.

\TurboPlayer\Configurations

Here you can list values with the TurboPlayer configurations. (A configuration is a top-level registry key with a valid parameter set. Standard configuration is "TurboPlayer".) For each subvalue the value name is the top-level keyname of the configuration and the data (a string) is the name which is displayed to the user in a selection dialog. (Hint: all parameters can be moved to a different top-level key but the list of configurations must be below "TurboPlayer" because this is the fixed start point for TurboPlayer to look for possible configuration names.) Example for a subvalue: "Turbo_Youth=Configuration for youth service"Since TurboPlayer v6 subkeys can be used instead of the usage of values as described here.
ConfigKey: (Par_string) Not used anymore.
ConfigValue: (Par_string) Each value represents a TurboPlayer configuration. The value name (here: ConfigValue) is a top-level key which contains a full set of TurboPlayer parameters. The data is a string which is displayed to the user in the selection dialog during startup. You can use two special characters in the value name: if the value name starts with an underscore "_" the value is ignored. If the value name ends with a star "*" it will be made default configuration in the selection dialog.
DialogTimeout: (Par_int, Default=6) With this parameter you can configure how long the dialog for selecting one of the configurations is displayed at startup. The time is in seconds and the default value

is 6 seconds. You can use the value 0 to achieve that the dialog is automatically closed without any delay. But make sure that you have defined exactly one configuration to be the default one. Otherwise, the dialog will stay open. (Valid since: 14.07.2021)

\TurboPlayer\Configurations\...

Since version 6.0 TurboPlayer allows to define the list of available configurations in subkeys of key "TurboPlayer\Configurations" instead of values. This allows to define more parameters as it is possible with the simple values. Though TurboPlayer adds up all defined configurations from subkeys and values, it is recommended to use either subkeys or values, but not both.
AvailableForHosts: (Par_string) A comma-separated list with computer names, for which the configuration should be available. You can use a star and a question mark as placeholders, like for a file search. A single star matches all possible computer names. Using this parameter can help when you maintain the list of configurations globally. (Valid since: 19.04.2021)
AvailableForStations: (Par_string) A comma-separated list with DigaSystem station names, for which the configuration should be available. You can use a star and a question mark as placeholders, like for a file search. A single star matches all possible station names. Using this parameter can help when you maintain the list of configurations globally. (Valid since: 19.04.2021)
DefaultForHosts: (Par_string) A comma-separated list with computer names, for which the configuration should be the default configuration. Using this parameter can help when you maintain the list of configurations globally. Pay attention to make only a single configuration the default one for each host. If you define multiple default configurations, the behaviour is undefined, TurboPlayer will pick one of them. (Valid since: 19.04.2021)
DefaultForStations: (Par_string) A comma-separated list with DigaSystem station names, for which the configuration should be the default configuration. Using this parameter can help when you maintain the list of configurations globally. Pay attention to make only a single configuration the default one for each host. If you define multiple default configurations, the behaviour is undefined, TurboPlayer will pick one of them. (Valid since: 19.04.2021)
Label: (Par_string) The name of the configuration as it should be displayed to the user. If nothing is specified, the name of the configuration key is used. (Valid since: 19.04.2021)

\TurboPlayer\EventsIn

Each subkey represents an in-event.

\TurboPlayer\EventsIn\...

-
Name: (Par_string) A user friendly name for the in-event. If nothing is defined, the key name is used instead. (Valid since: 01.10.2005)
Program: (Par_string) This is a comma-separated list of service names for which the in-event should be available. The specification of a single star '*' means: it is available in all services. (Valid since: 01.10.2005)

\TurboPlayer\EventsInternal

Each subkey represents an internal event.

\TurboPlayer\EventsInternal\...

-
Command: (Par_string) A command or condition string which is executed if the corresponding internal event is being triggered. (Valid since: 01.10.2005)
CommandFile: (Par_string) Like the parameter "Command", but here you can specify a filename containing the macros. It can be any textfile with encoding: ANSI, Unicode or UTF-8, but for Unicode and UTF-8 the correct byte order mark (BOM) must be present in the file. If both parameters are given, "Command" is preferred.
Event: (Par_string) Internal events are triggered when something happens in TurboPlayer internally. Here you can specify for which event the corresponding command should be executed. The following events have been defined so far: Init, Startup, Reset, GUIConnected, ModeChanged, NextElementChanged, ControlChanged, ElementStarting, ElementStarted, ElementStopped, PrelistenStateChanged, PFLStateChanged. See TurboPlayerTechManual for more information. (Valid since: 01.10.2005)

\TurboPlayer\EventsOut

All possible out-events are listed in the keys below.

\TurboPlayer\EventsOut\...

The name of the subkey is the operation as it is stored in the BCS-XML data in tagname "Control_Operation[n]".Please note: The name of this subkey must not start with the suffix "TP_" and it must not contain any spaces or arithmetic operators like a minus! Limit the key name to letters A-Z, a-z, numbers 0-9 and underscore "_"! Reason is that you define a new macro command with the key name. If you use, e.g. a space, you will not be able to call the defined EventOut from a macro.
Bitmap: (Par_string) If an event is "fully specified" (that means the reference point, the offset and a bitmap are being specified) the event can be displayed in the list view of TurboPlayer and DigAIRange. You can specify a bitmap file which is used as symbol in the events column. Alternatively you can specify a single letter only. TurboPlayer and DigAIRange will display this letter instead of a bitmap. For more information about supported formats, prerequisites and general information about how to create images for events-out, please have a look into chapter 1.2.5 "Customizable image files" of BCSTechManual. (Valid since: 01.01.2005)
BitmapModified: (Par_string) If you allow the users to modify a fully-specified event-out (by settings the parameter Changeable=true), you can specify an alterative image with this parameter. This image will be displayed when an element has a control, which is not identical to its corresponding event-out in the registry. (One of the values: reference point, offset or parameter can be modified.) If no such alternative image is defined, the programs will draw a red frame around the main image (from the "Bitmap" parameter). (Valid since: 18.12.2023)
BitmapPosition: (Par_string, Possible values=1,2,3,4,5...) Here the position of the bitmap symbol within the events column can be specified. This is a number, starting with 1 for the leftmost position. The same position can be used for multiple events (typically for exclusive events). For events without position TurboPlayer/DigAIRange will use continuous positions, starting one right of the highest specified position. (Valid since: 18.04.2005)
Changeable: (Par_string, Possible values=true, false, TRUE, FALSE, yes, no, 1, 0, Default=false) Normally, controls, which correspond to a fully-specified event, cannot be modified by the user. This means: the values for: reference point, offset and parameter are always identical to the values for the event-out in the registry. You can allow the users to modify one the mentioned values by setting this parameter to "true". In this case it might be helpful to provide an alterantive image with the parameter "BitmapModified" to signal the users that a control was modified and is no longer identical to its corresponding event. (Valid since: 18.12.2023)
Command: (Par_string) Here the command string of the generic- or out-event is defined. It will be executed by the macro processor when the event is triggered.
See the chapter "Macro programming" in TurboPlayer technical manual for more information. (Valid since: 01.10.2005)
CommandFile: (Par_string) Like the parameter "Command", but here you can specify a filename containing the macros. It can be any textfile with encoding: ANSI, Unicode or UTF-8, but for Unicode and UTF-8 the correct byte order mark (BOM) must be present in the file. If both parameters are given, "Command" is preferred.
Creator: (Par_string, Default=<none>) Allows automatic creation of the control if a specific code string appears in one of the metadata fields. For a detailed description see TurboPlayer-TechManual chapter "Control creation". Short description: "field(position)!/:code" field=fieldname, position=[B(egin),E(nd),*(anywhere),1..n], !=case sensitive, /=remove code, :=code separator, code=string to look for.
Exclusive: (Par_string) Here you can list the keys of other events which must not be set simultaneously with this event. When this event is being set, all events specified in the exclusive list are removed from the element. The list is comma separated. (Valid since: 18.04.2005)
Name: (Par_string, Default=is equal to the key name) The name of the event as it is being displayed to the user (e.g. in controls/events for elements or on stack buttons). (Valid since: 01.01.2004)
Offset: (Par_int) The offset together with the reference point defines the time position during playout when the event is fired. The value is in milliseconds, positive values are behind the reference point. Hint: the absolute position must not be outside of the played interval. (Valid since: 01.01.2004)
Parameter: (Par_string) Some events might need a parameter (most do not). This can be a string with any possible data. This registry value is only used as default if a new event is created, afterwards the parameter is stored in the BCS-XML data. (Valid since: 18.04.2005)

Program: (Par_string) The programs (=services) for which the event is valid can be limited with this key. You can enumerate all programs in a comma separated list. If the value is a star '*', the event is valid for all programs.
Since version 6.3 you can add the three reserved keywords "GlobalJingles", "GeneralPools" and "PersonalPools" to make an event available in these node types (which are all outside of any service). If you use the star "*", there is no need to add one of the three reserved keywords, because they are included automatically. Attention: if you make an event available in one of these three node types, always make the event available in all programs, in which elements with these events could be used (e.g. by copying an element into a rundown within a program). If an event becomes unavailable within a program, this would be confusing for the users. And important: for TurboPlayer the currently selected program is decisive! That means: if the current program of TurboPlayer is e.g. "P1" and the user loads a global jingle group into a stack, TurboPlayer will only execute events, which are available within "P1". Events, which are available for global jingles, but which have been disabled for "P1" will not be executed. (Valid since: 01.01.2004)
ReferencePoint: (Par_string, Possible values=MarkIn, MarkOut, LinkIn, LinkOut, Intro, Outro) The reference point together with the offset specifies the time position during playout when the event is fired. (Valid since: 01.01.2004)
RundownListType: (Par_string, Possible values=One of the rundown list types) If you want to use an event for a stack button in mode "change parameter set" you must specify with this parameter for which rundown the event is valid. You can list one of the rundown list types (typically Stack1...Stack15) or '*' meaning: all rundowns.
VIS: (Par_string, Possible values=Yes, No, TRUE, FALSE, 1, 0, Default=Yes) This parameter decides whether the event is Visible In the Scheduler. By default it is visible, but if you want to have internal events of the on-air application, you can set this parameter to no.

\TurboPlayer\EventsTimed

Here events can be defined which are executed regularly.

\TurboPlayer\EventsTimed\...

-
ActiveAtStartup: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=Yes) Defines whether the event is already active when TurboPlayer starts. (Valid since: 10.02.2006)
Command: (Par_string) A command or condition string which is executed if the corresponding event is being triggered. (Valid since: 10.02.2006)
CommandFile: (Par_string) Like the parameter "Command", but here you can specify a filename containing the macros. It can be any textfile with encoding: ANSI, Unicode or UTF-8, but for Unicode and UTF-8 the correct byte order mark (BOM) must be present in the file. If both parameters are given, "Command" is preferred.
HourPoints: (Par_string) This value must only be defined if you want to use a timer, which runs at specific time points each hour. Specify a comma-separated list of the points. The format of each point can be: <seconds>, <minutes:seconds> or <minutes:seconds.milliseconds> Example: "30:00,59:57.500" (Valid since: 10.02.2006)
Log: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) Defines whether a line is written to the protocol "TechLogEx" whenever the event is being executed. (Valid since: 10.02.2006)
Period: (Par_int) This value must only be defined if you want to use a timer, which runs periodically. Specify the period in milliseconds. The minimum value is 10 ms. (Valid since: 10.02.2006)

\TurboPlayer\FileCopy

Here you can define the behaviour of TurboPlayer when it has to copy files to the media directories. For a detailed description of the values see -> DigAIRange\FileCopy. All values and comments apply for TurboPlayer too.

\TurboPlayer\GUI

This section and all subsections stores GUI relevant data which are easy to set by settings dialogs in the GUI. Each frame window has got a menu item "settings" that opens a settings dialog.Only those values differing from their defaults are written to the registry.

AutoDisableDlgs: (Par_int) If settings dialogs are password protected and they are enabled by entering the right password the value of 1 disables automatically the settings dialogs after a certain timespan (->AutoTimeDisableDlgs).
Parameters are eays to set in the GUI (-> system menu -> settings | Dialogs&Menus)
AutoTimeDisableDlgs: (Par_int) Time in seconds the password protection for the settings dialogs will be automatically deactivated again.
Parameter is easy to set in the GUI (-> system menu -> settings | Dialogs&Menus).
See also AutoDisableDlgs.
CloseStartupIfError: (Par_int, Possible values=0... 10000, Default=0) In this parameter TP sets an internal timer to trigger the closing of the startup dialog. It is NOT supposed to be changed via Admin by the user.
Being 0, the Startup dialog remains there even if errors occur. If different from 0 (set internally from TP), the dialog is closed automatically after some time the initialization is finished.
DisableContextMenus: (Par_int, Possible values=0, 1, Default=0) Value of 1 disables all context menu items of each frame window in the GUI.
Parameter is easy to set in the GUI (->system menu ->settings | Dialogs&Menus).
GapCalcWithFixedDurationOfGroups: (Par_int, Possible values=0,1, Default=1) By default the gap calculation in TurboPlayer refers to the planned duration of groups. Setting the value to 0 will no longer take the fixed length of the groups but the real duration of the elements inside the group. This parameter can be changed inside the settings dialog of a TimeInfo window for gap/overlap display and works globally for all time windows. (Valid since: 11.11.2010)
LogMemoryStatusInterval: (Par_int, Default=0) In case you encounter memory leaks or an out-of-memory situation in the TurboPlayerGUI, you can activate a regular logging of the current memory situation with this parameter. It is an integer, the interval between two log outputs in seconds. The output is written to the regular log of the GUI. A value of 0 switches the regular logging off. (Valid since: 14.12.2022)
PasswdSettingsDlgs: (Par_string) Encrypted password to protect settings dialogs from user access. Password can only be set in the GUI (->system menu ->settings | Dialogs&Menus.
SwitchStartModesIfAutomatic: (Par_int, Possible values=0,1, Default=0) Set this parameter to 1 to display the 'sequenced mode' icon in TurboPlayer during the automatic mode. (Valid since: 01.01.2007)

\TurboPlayer\GUI\UiSchemes

Contains additional UISchemes in the respective subkeys, as well as defines the currently used UIScheme. If nothing is defined here, the internal "UIScheme_light" will be used, which can also be changed here creating a subkey with this same name.
UIScheme_current: (Par_string, Default=UIScheme_light) References a UIScheme, which can be defined in the local or global TurboPlayer|GUI|UISchemes key, as well as under Common/UISchemes. (Valid since: 10.06.2020)

\TurboPlayer\GUI\Windows

This section and all subsections stores data for the several frame windows.The data are easy to set by settings dialogs in the GUI. Each frame window has got a menu item "settings" that opens a settings dialog.Only those values differing from their defaults are written to the registry.
GridSpacing: (Par_int, Possible values=1...100, Default=5) Grid space for move and resize the frame windows in the GUI.
GridSpacing: (Par_int, Possible values=1...100, Default=5) Grid space for move and resize the frame windows in the GUI.
Parameter adjustable in the GUI (->system menu ->Add Windows).

\TurboPlayer\GUI\Windows\Button*

* = 1...n the button number.This section stores properties for the button Trashcan, AdjustTimes, etc.Parameters are eays to set in the GUI (->menu ->settings).

\TurboPlayer\GUI\Windows\CFM

Settings for CrossfadeMixer

\TurboPlayer\GUI\Windows\CFM\[n]\Extra

TurboPlayer has a settings dialog for CrossfadeMixer/OnAIR TrackMixer. Unfortunately there are some parameters which cannot be changed with the dialog. This key allows to set these parameters. Every value you define in this key will be relayed to CFM/OTM. The value type can be: String and Integer. It is even allowed to define subkeys to build up hierarchical XML parameters. It is not possible to overwrite parameters which are already available in the dialog. For a list of possible parameters you have to check the technical documentation of CFM/OTM. (Since version 4.8.1708.0)
CollapsePanes: (Par_string, Possible values=TRUE, FALSE, 1, 0, Default=FALSE) Collapse Detail Panes:
TRUE:
The panes "Marker Details" and "Show Overview" will collapse/expand, when the tracks are collapsed/expanded.
FALSE:
The panes "Marker Details" and "Show Overview" will not change, when the tracks are collapsed/expanded.
DragTailElements: (Par_string, Possible values=Never,WithShift,WithoutShift,WithAlt,WithoutAlt,ButtonOff,ButtonOn, Default=Never) Drag tail elements feature in OnAIR TrackMixer:
• Never: Feature is disabled. This is the default if the parameter is not set. Button is not shown in main toolbar.
• WithShift: Feature is active, if the shift key is pressed before dragging the audio element. Button is shown in main toolbar and is deactivated.
• WithoutShift: Feature is active, if the shift key is not pressed and not active otherwise. Button is shown in main toolbar and is activated.
• WithAlt: Feature is active, if the alt key is pressed before dragging the audio element. Button is shown in main toolbar and is deactivated.
• WithoutAlt: Feature is active, if the alt key is not pressed and not active otherwise. Button is shown in main toolbar and is activated.
DrawZeroAtFirstElementMarkOut: (Par_string, Possible values=True,False, Default=True) The parameter DrawZeroAtFirstElementMarkOut shifts the 0:00 point to the Mark In of the second element instead of the Mark Out of the first element if set to False. Only Visible if Relative Times is selected. If between 1st and 2nd loaded element there is a gap, then 0 point (transition) starts at Mark Out of 1st element.
DuckingDuringVoiceTracking: (Par_string, Possible values=True,False, Default=False) Use ducking feature in VoiceTracking mode in OnAIR TrackMixer:
• False - No ducking is used in VoiceTracking Mode
• True - Ducking is used in VoiceTracking Mode.
EditBeginAndEndOfShow: (Par_int, Possible values=0, 1, Default=0) Allow modification of start markers on first element of show and end markers of last element in show.
• 0: Modification of start/end markers is not possible
• 1: Modification of start/end markers is allowed
FadeInCurveType: (Par_string, Possible values=Linear,Log1,Log2,LogMinus2, Default=Linear) Curve type for "FadeIn" Button:
• Linear - Creates a linear curve.
• Log1 - Creates a Log 1 curve.
• Log2 - Creates a Log 2 curve.
• LogMinus2 - Creates a Log -1 curve.
FadeOutCurveType: (Par_string, Possible values=Linear,Log1,Log2,LogMinus2, Default=Linear) Curve type for "FadeOut" Button:
• Linear - Creates a linear curve.
• Log1 - Creates a Log 1 curve.
• Log2 - Creates a Log 2 curve.
• LogMinus2 - Creates a Log -1 curve.
FadePointLowerScale: (Par_string, Possible values=-98.0..-12.0, Default=-98.0) This value defines the minimum amplitude a fade point can have. This value is defined in dB.
FadePointUpperScale: (Par_int, Possible values=0..., Default=0) This value defines the maximum amplitude a fade point can have. This value is defined in dB.
HideIntroOutroOutsidePlayableArea: (Par_string, Possible values=True,False, Default=False) Hide the Inro1, Intro2, Outro1 and Outro2 symbols if they are before MarkIn or behind MarkOut.

LoadNextSequencedElement: (Par_string, Possible values=True, False, Default=False) False [Default]: Only two elements are loaded
True: When next element is set sequenced, then 3 elements are loaded
LogLevel: (Par_string, Possible values=None, Minimal, Normal, Extended, Debugging, OcxDebugging, Default=Normal) Set the detail grade for the information in the OTM log file
* None
* Minimal
* Normal
* Extended
* Debugging
* OcxDebugging
MarkerDetailsExpanded: (Par_string, Possible values=TRUE, FALSE, 1, 0, Default=TRUE) Expand/collapse "Marker Details" pane at startup.
TRUE:
The pane "Marker Details" is expanded at startup.
FALSE:
The pane "Marker Details" is collapsed at startup.
MaximumLoudnessRange: (Par_string, Default=10) This parameter is optional part of a loudness set.

Maximum acceptable loudness range in LU.
MaximumMomentaryLoudnessLevel: (Par_string, Default=-23.0) This parameter is mandatory part of a loudness set.

Target value for Maximum Momentary Loudness in LUFS. Value is a negative floating-point number, e.g. "-23.0". The decimal separator should always be a dot (".") and not a comma (",").
For very short audio material (less than 30 seconds), EBU R128 recommends to rely on this value (and MaximumShorttermLoudnessLevel) and not the Integrated Loudness.
MaximumShorttermLoudnessLevel: (Par_string, Default=-20.0) This parameter is mandatory part of a loudness set.

Target value for Maximum Shortterm Loudness in LUFS. Value is a negative floating-point number, e.g. "-23.0". The decimal separator should always be a dot (".") and not a comma (",").
For very short audio material (less than 30 seconds), EBU R128 recommends to rely on this value (and MaximumMomentaryLoudnessLevel) and not the Integrated Loudness.
MaximumTruePeakLevel: (Par_string, Default=-1.0) This parameter is mandatory part of a loudness set.

Maximum true peak level in dBTP. Value is a negative floating-point number, e.g. "-1.0". The decimal separator should always be a dot (".") and not a comma (",").
MinimumFadepointDistance : (Par_int, Possible values=100..., Default=200) Minimum distance between automatically generated fade points in milliseconds.
MouseButtonRight: (Par_string, Possible values=PausePlayback,None, Default=PausePlayback) Right mouse button behaviour in OnAIR TrackMixer:
• PausePlayback - Right mouse button pauses the playback. This is the behavior of OTM 1.0.
• None - Right mouse button does not pause playback.
MouseWheelUp: (Par_string, Possible values=MoveWaveformToLeft,MoveWaveformToRight, Default=MoveWaveformToLeft) Mouse wheel scrolling behaviour in OnAIR TrackMixer:
• MoveWaveformToLeft - Waveform is moved to the left for Mouse-Wheel-Up. This is the behavior of OTM 1.0.
• MoveWaveformToRight - Waveform is moved to the right for Mouse-Wheel-Up. This is the behavior of most DAVID Applications
NextElementIndicatorColor: (Par_string, Possible values=#000000) Color of the next element position indicator line.
NextElementIndicatorVisibility: (Par_string, Possible values=True, False, Default=False) Show the vertical indicator line at the next element position.
NextMarkInIndicatorColor: (Par_string, Possible values=#000000) Color of the vertical indicator line at MarkIn position of the next element.
NextMarkInIndicatorVisibility: (Par_string, Possible values=True, False, Default=False) Show the vertical indicator line at the position of the MarkIn of the next element.

OpenSelectedElement: (Par_string, Possible values=0, 1, Default=0) Controls which element is loaded after switching from multi-track into 1-track mode:
0 - the first element (i.e. the element from the top track of multi-track mode)
1 - the element that was originally loaded into OTM
When you drag one audio element into OTM, OTM (since version 1.3 or so) attempts to load a complete transition. With OpenSelectedElement=0, the first element of this transition is loaded into 1-track mode; with OpenSelectedElement=1, the element is loaded into 1-track mode that was dragged into OTM.

This parameter is only used for OTM versions 1.3.346.19 and higher patch levels, or 2.0.497.0 and above.
OutputFaderVisible: (Par_string, Possible values=True,False, Default=True) Output Fader Visiblity in OnAIR TrackMixer:
• True - Output Fader is visible. This is the behavior of OTM 1.0.
• False - Output Fader is hidden.
RecordingMetadata: (Par_string, Possible values=<Field1>Value</Field1>...<FieldN>Value</FieldN>, Default=<Class>Audio</Class>) This Parameter contains configurable Metadata that OTM will write for every recording (regardless of the recording mode). The format of the value is a list of XML segments. Example:<Class>Audio</Class><Field1>String1</Field1>
* The admin is responsbile for setting meaningful XML values and to spell them correctly according to the BCSTechManual. E.g. setting values that are critical to the BCS System (Time_Start, etc.) doesn't make any sense.
* The parameter values are set on element creation and can be overwritten later on by the OTM.
* As of version 2.2.610.0 it is possible to overwrite the "Title" field.
RememberZoomRange: (Par_int, Possible values=0, 1, Default=0) This parameter defines the behavior of the zoom level.
0: As before, the zoom level is reset to the configured default value for NextTransition/PreviousTransition command.
1: The zoom level is retained for NextTransition/PreviousTransition command.
RestartRecording: (Par_string, Possible values=TRUE, FALSE, 1, 0, Default=FALSE) Enable the restart recording feature.
ScaleType: (Par_string, Possible values=LogScale, LinearScale, Default=LogScale) Scale type of amplitude scale:
LogScale: Logarithmic scale.
LinearScale: Linear scale
ShowMainSubButton: (Par_string, Possible values=NoTransitionChannels, Always, Never, Default=NoTransitionChannels) This parameter controls if the Main-Element/SUb-Element Button is shown in the Track Header:
* NoTransitionChannels
The Button is shown if no Transition channels are configured in the BCS.
* Always
The Button is always visible.
* Never
The Button is never visible.
ShowMoveToButtons: (Par_string, Possible values=True, False, Default=False) Show or hide the toolbar buttons 'Move to Start' and 'Move to End'.
True: Show the buttons.
False: Hide the buttons.
ShowOverviewExpanded: (Par_string, Possible values=TRUE, FALSE, 1, 0, Default=TRUE) Expand/collapse "Show Overview" pane at startup.
TRUE:
The pane "Show Overview" is expanded at startup.
FALSE:
The pane "Show Overview" is collapsed at startup.
SoundheadPositionOnLoad: (Par_string, Possible values=MarkOut, MarkIn, Default=MarkOut) • MarkOut: Set the view to the MarkOut of the first element when a new transition is loaded. Set the soundhead to the left of the view if SoundheadPreroll is not used or set the soundhead according to the SoundheadPreroll if the parameter is used.

• MarkIn: Set the view when a new transition is loaded to the Mark In of the second element. Set the soundhead to the MarkIn if SoundheadPreroll is not used or set the soundhead according to the SoundheadPreroll if the parameter is used.
SoundheadPreroll: (Par_int, Possible values=-1...86400, Default=-1) Position of Soundhead after loading of elements:
* -1
Soundhead is positioned on MarkIn of first element. If MarkIn is not visible the soundhead is set to the left window border.
* 0
Soundhead is positioned on MarkOut of first element.
* 5
Soundhead is positioned 5 seconds left of the MarkOut of the first element. If the position is not in the visible area, the soundhead is set to the left window border.
* 60
Soundhead is positioned 60 seconds left of the MarkOut of the first element. If the position is not in the visible area, the soundhead is set to the left window border. If the element is shorter than 60 seconds, the soundhead is set to the MarkIn if visible.
SourceEnvelopeType: (Par_string, Possible values=Visible,Hidden,VisibleOutsideOfMarkInOut, Default=Visible) Behavior of source waveform (without fading).
Volume changes such as fades/ducking are reflected at the waveform.
• Hidden - The area before mark in and after mark out is completely hidden and fades/duckings reduce the height of the waveform. No gray waveform in the background.
• Visible - New behavior: Gray waveform in the background of the entire audio. This is the default value.
• VisibleOutsideOfMarkInOut - Behavior of CFM: Gray waveform before Mark-In and after Mark-Out. No gray waveform between Mark-In and Mark-Out.
StartModeBlock: (Par_string, Possible values=True, False, Default=False) Experimental feature: Block movement of audio if value of StartMode is EndOfTime or StartOfTime.
StopPlaybackAtMarkOut: (Par_string, Possible values=True, False, Default=False) True: Stop the playout in single track mode at the mark out.
False: Stop the playout in single track mode at the end of the audio file.
StopPlaybackWhenSave: (Par_string, Possible values=TRUE, FALSE, Default=FALSE) Stop the playback of the audio when the changes are saved.
• TRUE: Stop the playback.
• FALSE: Continue the playback.
TransitionFlaggingClasses: (Par_string, Possible values=[class];[class];...) The main audio elements will be converted to sub audio elements, if the class is in this list.
TransitionFlaggingLength: (Par_int, Possible values=0...100, Default=30) The main audio elements will be converted to sub audio elements, if the length is less than <TransitionFlaggingLength> percent of the longest element in the timeline.
TransitionFlaggingMode: (Par_int, Possible values=1, 2, 3, 4, Default=1) Controls when audio elements are converted between main and sub elements. If transition channels are configured, this is the same as routing to main or transition channels.
1: Conversion runs after each change. The button in the control bar is hidden.
2: Conversion must be triggered manually by a click on the button in the control bar.
3: The button in the control bar toggles automatic conversion (like mode 1) on or off; initially automatic conversion is ON.
4: The button in the control bar toggles automatic conversion (like mode 1) on or off; initially automatic conversion is OFF.

Notes:
- In modes 3 and 4, the button indicates the ON or OFF state.
- Depending on whether or not transition channels are configured, the button has different appearance in modes 2, 3, and 4.
TransitionFlaggingRecording: (Par_string, Possible values=Default, ForceOn, ForceOff, Default=Default) * Default: The transition flagging logic respects class and length (as in release 2017.2.0)
* ForceOn: OTM forcibly flags all recordings (identified by: Generator field equals "OTM" with Time_StartOnTransitionChannel=1. It also means recordings are always sub-elements and never main-elements.

* ForceOff: OTM forcibly flags all recordings (identified by: Generator field equals "OTM" with Time_StartOnTransitionChannel=0
TransitionView: (Par_string, Possible values=TRUE, FALSE, 1, 0, Default=FALSE) Only important elements in a transition are focused. Therefore a different visualization is applied for non-rundown elements that are not located in the middle of a transition.

Assuming a rundown that consists of several regular rundown elements and many transitions, it is key that OTM tries to load only a single transition. In order to not hide elements that are relative to the loaded rundown elements, but are not part of the transition currently being edited, these elements should also be loaded, but visualized differently.

* TRUE
Activate feature
* FALSE
Deactivate feature
UseTimeOfDayView: (Par_string, Possible values=TRUE, FALSE, 1, 0, Default=False) Switch between time/duration and real clock times:
* TRUE
Show Start and Stop fields instead of MarkIn and MarkOut fields.
Switch ruler to "Time Of Day" view.
* FALSE
Show MarkIn and MarkOut fields instead of Start and Stop fields.
Switch ruler to "Relative Times" view.
WaveformOffset: (Par_string, Possible values=0.0...100.0, Default=0.0) The vertical zoom level in dB, which is applied to the waveform if activated.
If the value is "0.0" then the toggle button is not shown.
WaveformOffsetInitiallyActive: (Par_string, Possible values=0, 1, FALSE, TRUE, YES, NO, Default=FALSE) Determines the initial state of the "waveform offset" toggle button:
FALSE - waveform offset is off
TRUE - waveform offset is on
ZeroPositionIndicatorColor: (Par_string, Possible values=#000000) Color of the zero position indicator line.
ZeroPositionIndicatorVisibility: (Par_string, Possible values=True, False, Default=False) Show the vertical indicator line at the zero position.
ZoomDefaultCenterPosition: (Par_string, Possible values=Soundhead, ViewArea, Default=Soundhead) Default center position for zoom action:
Soundhead: Zoom to Soundhead (Shift+Scrollwheel, ZoomButton); Zoom to view area (CTRL+Zoom Button/Ctrl+Shift+Scrollwheel)
ViewArea: Zoom to view area (Shift+Scrollwheel, ZoomButton); Zoom to Soundhead (CTRL+ZoomButton/Ctrl+Shift+Scrollwheeel)

\TurboPlayer\GUI\Windows\CFM\[n]\Extra\AdvancedVoiceTracking\Ducking

Ducking parameter for OTM Advanced Voice Tracking
Amplification: (Par_string, Possible values=-200...0, Default=-15.0) Amplification in dB for ducking settings in AdvancedVoiceTracking Mode.
When an audio is ducked it will be lowered by %Amplification% dB in the ducking area.
FadeDown: (Par_int, Possible values=0...100000, Default=1000) FadeDown in milliseconds for ducking settings in AdvancedVoiceTracking Mode.
"FadeDown" is used together with "FadeDownCurveType" to define the form of the fade down part of the ducking curve.
The distance between the start and the end of the fade down is %FadeDown% ms.
FadeDownBefore: (Par_int, Possible values=0...100000, Default=500) FadeDownBefore in milliseconds for ducking settings in
AdvancedVoiceTracking Mode.
"FadeDownBefore" defines the alignment between the fade down curve and the duck position.
If %FadeDownBefore% is 0 ms, the fade down starts at the duck position.
If %FadeDownBefore" is 500 ms, the fade down starts 500 ms before the duck position.
FadeDownCurveType: (Par_string, Possible values=Linear,Log1,Log2,LogMinus2, Default=Linear) FadeDownCurveType fir ducking settings in AdvancedVoiceTracking Mode.

"FadeDownCurveType" is used together with "FadeDown" to define the form and size of the fade down part of the ducking curve.
"FadeDownCurveType" defines the form of the fade down curve.
The supported values are:
• Linear - Creates a linear curve.
• Log1 - Creates a Log 1 curve.
• Log2 - Creates a Log 2 curve.
• LogMinus2 - Creates a Log -1 curve.
FadeUp: (Par_int, Possible values=0...100000, Default=1000) FadeUp in milliseconds for ducking settings in AdvancedVoiceTracking Mode.
"FadeUp " is used together with "FadeUpCurveType" to define the form of the fade up part of the ducking curve.
The distance between the start and the end of the fade up is %FadeUp% ms.
FadeUpAfter: (Par_int, Possible values=0...100000, Default=500) FadeUpAfter in milliseconds for ducking settings in AdvancedVoiceTracking Mode.
Removed in version 2.0.534.0. and replaced by parameter FadeUpBefore.
FadeUpBefore: (Par_int, Possible values=0...100000, Default=500) FadeUpBefore in milliseconds for ducking settings in AdvancedVoiceTracking Mode.
"FadeUpBefore" defines the alignment between the fade up curve and the unduck position.
If %FadeUpBefore % is 0 ms, the fade up starts at the unduck position.
If %FadeUpBefore " is 500 ms, the fade up starts 500 ms before the unduck position.
FadeUpCurveType: (Par_string, Possible values=Linear,Log1,Log2,LogMinus2, Default=Linear) FadeUpCurveType for ducking settings in AdvancedVoiceTracking Mode.
"FadeUpCurveType" is used together with "FadeUp" to define the form and size of the fade up part of the ducking curve.
"FadeUpCurveType" defines the form of the fadeup curve.
The supported values are:
• Linear - Creates a linear curve.
• Log1 - Creates a Log 1 curve.
• Log2 - Creates a Log 2 curve.
• LogMinus2 - Creates a Log -1 curve.

\TurboPlayer\GUI\Windows\CFM\[n]\Extra\AdvancedVoiceTracking\FadeOut

FadeOut parameter for OTM Advanced Voice Tracking
FadeCurveType: (Par_string, Possible values=Linear,Log1,Log2,LogMinus2, Default=Linear) FadeCurveType for fadeout settings in AdvancedVoiceTracking Mode.
"FadeCurveType" is used together with "FadeTime" to define the form and size of the fade out curve.
"FadeCurveType" defines the form of the fade out curve.
The supported values are:
• Linear - Creates a linear curve.
• Log1 - Creates a Log 1 curve.
• Log2 - Creates a Log 2 curve.
• LogMinus2 - Creates a Log -1 curve.
FadeTime: (Par_int, Possible values=0...100000, Default=2500) FadeTime in milliseconds for fadout settings in AdvancedVoiceTracking Mode.
"FadeTime" is used together with "FadeCurveType" to define the form and size of the fade out curve.
The distance between the start and the end of the fade out is %FadeTime% ms.

\TurboPlayer\GUI\Windows\CFM\[n]\Extra\AdvancedVoiceTracking\Leveling

-
DistanceOfFadePoints: (Par_int, Possible values=0...5000, Default=100) DistanceOfFadePoints in milliseconds for leveling settings in AdvancedVoiceTracking Mode.
"DistanceOfFadePoints" is used together with "LevelModification" to define the size of the fade for level up/down.
The distance between the start and the end of the fade is %DistanceOfFadePoints% ms.
LevelModification: (Par_string, Possible values=0.5...5.0, Default=1.0) LevelModification in dB for leveling settings in AdvancedVoiceTracking Mode.
"LevelModification" is used together with "DistanceOfFadePoints" to define the size of the fade for level up/down.

The height in dB between the start and the end of the fade is %LevelModification% dB.
MinimumDuckLevel: (Par_string, Possible values=-10.0...-3.0, Default=-6.0) MinimumDuckLevel in dB for Leveling settings in AdvancedVoiceTracking Mode.
The "MinimumDuckLevel" controls the ducked state of the AdvancedVoiceTracking mode.
If the current duck level is below or at the same level of %MinimumDuckLevel% dB, the state is ducked.
In the ducked state the fade line and the duck symbol in the track header are shown in blue.
The ducked state can be reached with the level down shortcut(s) or with the duck shortcut(s).

\TurboPlayer\GUI\Windows\CFM\[n]\Extra\AutoDuck

Ducking parameters for "OnAIR TrackMixer"
FadeDownCurveType: (Par_string, Possible values=Linear,Log1,Log2,LogMinus2, Default=Linear) Curve type for fade down of the ducking feature:
• Linear - Creates a linear curve.
• Log1 - Creates a Log 1 curve.
• Log2 - Creates a Log 2 curve.
• LogMinus2 - Creates a Log -1 curve.
FadeUpCurveType: (Par_string, Possible values=Linear,Log1,Log2,LogMinus2, Default=Linear) Curve type for fade up of the ducking feature:
• Linear - Creates a linear curve.
• Log1 - Creates a Log 1 curve.
• Log2 - Creates a Log 2 curve.
• LogMinus2 - Creates a Log -1 curve.

\TurboPlayer\GUI\Windows\CFM\[n]\Extra\CrossfadeFadeInParameters

Crossfade fadein parameters for "OnAIR TrackMixer"
CrossfadeTrack: (Par_string, Possible values=True,False, Default=True) Crossfade track for the fade in track of the crossfade:
• True - Delete old fade in curve and create new fade in curve according to setting FadecurveType
• False - Leave the current fade in curve.
DefaultMode: (Par_string, Possible values=True,False, Default=True) Default mode for the fade in track of the crossfade:
• True - Use the current play position to calculate the fade time
• False - Use the FadeTime setting as the fade time
FadecurveType: (Par_string, Possible values=Linear,Log1,Log2,LogMinus2, Default=Linear) Curve type used for the fade in track of the crossfade:
• Linear - Creates a linear curve.
• Log1 - Creates a Log 1 curve.
• Log2 - Creates a Log 2 curve.
• LogMinus2 - Creates a Log -1 curve.
FadeTime: (Par_int, Possible values=100..., Default=2500) Fade time for the fade in track of the crossfade in milliseconds. Used only if DefaultMode = False for the fade in track.

\TurboPlayer\GUI\Windows\CFM\[n]\Extra\CrossfadeFadeOutParameters

Crossfade fadeout parameters for "OnAIR TrackMixer"
CrossfadeTrack: (Par_string, Possible values=True,False, Default=True) Crossfade track for the fade out track of the crossfade:
• True - Delete old fade out curve and create new fade out curve according to setting FadecurveType
• False - Leave the current fade out curve.
DefaultMode: (Par_string, Possible values=True,False, Default=True) Default mode for the fade out track of the crossfade:
• True - Use the current play position to calculate the fade time
• False - Use the FadeTime setting as the fade time
FadecurveType: (Par_string, Possible values=Linear,Log1,Log2,LogMinus2, Default=Linear) Curve type used for the fade out track of the crossfade:
• Linear - Creates a linear curve.
• Log1 - Creates a Log 1 curve.
• Log2 - Creates a Log 2 curve.
• LogMinus2 - Creates a Log -1 curve.

FadeTime: (Par_int, Possible values=100..., Default=2500) Fade time for the fade out track of the crossfade in milliseconds. Used only if DefaultMode = False for the fade out track.

\TurboPlayer\GUI\Windows\CFM\[n]\Extra\StartNextAutoDucking

-

Amplification: (Par_string, Possible values=-200.0...0.0, Default=-10.0) Amplification in dB for ducking settings in StartNext-Autoducking.
When an audio is ducked it will be lowered by %Amplification% dB in the ducking area.
"Amplification" is used for duck targets ducked by duck source audios and by audio segments from the recording.
AttackOffset: (Par_int, Possible values=0...100000, Default=150) AttackOffset in milliseconds for StartNext-Autoducking.
The "AttackOffset" will be used together with the "AttackTime", the "AttackThreshold" and the "ThresholdTolerance" to detect the start of an audio segment in the recording.
1. The %AttackOffset% ms offset will compensate for internal delays by the detection logic with a default value of 150 ms.
2. The %AttackOffset% ms offset is not needed to compensate for the %AttackTime% ms offset.
3. The %AttackOffset% ms offset can also be used to further correct the detected start position of an audio segment in the recording to the left or the right in the timeline.
AttackThreshold: (Par_string, Possible values=-100.0...0.0, Default=-30.0) AttackThreshold in dB for StartNext-Autoducking.
The "AttackThreshold" will be used together with the "AttackTime", the "AttackOffset" and the "ThresholdTolerance" to detect the start position of an audio segment in the recording.
The start of an audio segment in the recording is detected if the signal is above %AttackThreshold% dB for at least %AttackTime% ms and in at least %ThresholdTolerance% percent of the values.
AttackTime: (Par_int, Possible values=0...100000, Default=100) AttackTime in milliseconds for StartNext-Autoducking.
The "AttackTime " will be used together with the "AttackThreshold", the "ThresholdTolerance" and the "AttackOffset" to detect the start position of an audio segment in the recording.
The start of an audio segment in the recording is detected if the signal is above %AttackThreshold% dB for at least %AttackTime% ms and in at least %ThresholdTolerance% percent of the values.
FadeDown: (Par_int, Possible values=0...100000, Default=1000) FadeDown in milliseconds for ducking settings in StartNext-Autoducking.
"FadeDown" is used together with "FadeDownCurveType" to define the form of the fade down part of the ducking curve.
The distance between the start and the end of the fade down is %FadeDown% ms.
"FadeDown" is used for duck targets ducked by duck source audios and by audio segments from the recording.
FadeDownBefore: (Par_int, Possible values=0...100000, Default=1000) FadeDownBefore in milliseconds of the ducking settings in
StartNext-Autoducking.
"FadeDownBefore" defines the alignment between the fade down curve and the duck position.
"FadeDownBefore" is used for duck targets ducked by duck source audios and by audio segments from the recording.
If %FadeDownBefore% is 0 ms, the fade down starts at the duck position.
If %FadeDownBefore" is 500 ms, the fade down starts 500 ms before the duck position.
FadeDownCurveType: (Par_string, Possible values=Linear,Log1,Log2,LogMinus2, Default=Linear) FadeDownCurveType of the ducking settings in StartNext-Autoducking.
"FadeDownCurveType" is used together with "FadeDown" to define the form and size of the fade down part of the ducking curve.
"FadeDownCurveType" defines the form of the fade down curve.
The supported values are:
• Linear - Creates a linear curve.
• Log1 - Creates a Log 1 curve.
• Log2 - Creates a Log 2 curve.
• LogMinus2 - Creates a Log -1 curve.
FadeUp: (Par_int, Possible values=0...100000, Default=1000) FadeUp in milliseconds for ducking settings in StartNext-Autoducking.

"FadeUp " is used together with "FadeUpCurveType" to define the form of the fade up part of the ducking curve.
The distance between the start and the end of the fade up is %FadeUp% ms.
"FadeUp" is used for duck targets ducked by duck source audios and by audio segments from the recording.
FadeUpBefore: (Par_int, Possible values=0...100000, Default=0) FadeUpBefore in milliseconds of the ducking settings in
StartNext-Autoducking.
"FadeUpBefore" defines the alignment between the fade up curve and the unduck position.
"FadeUpBefore" is used for duck targets ducked by duck source audios and by audio segments from the recording.
If %FadeUpBefore % is 0 ms, the fade up starts at the unduck position.
If %FadeUpBefore " is 500 ms, the fade up starts 500 ms before the unduck position.
FadeUpCurveType: (Par_string, Possible values=Linear,Log1,Log2,LogMinus2, Default=Linear) FadeUpCurveType of the ducking settings in StartNext-Autoducking.
"FadeUpCurveType" is used together with "FadeUp" to define the form and size of the fade up part of the ducking curve.
"FadeUpCurveType" defines the form of the fade upcurve.
The supported values are:
• Linear - Creates a linear curve.
• Log1 - Creates a Log 1 curve.
• Log2 - Creates a Log 2 curve.
• LogMinus2 - Creates a Log -1 curve.
HoldOffset: (Par_int, Possible values=0...100000, Default=150) HoldOffset in milliseconds for StartNext-Autoducking.
The "HoldOffset" will be used together with the "HoldTime" the "HoldThreshold" and the "ThresholdTolerance" to detect the end of an audio segment in the recording.
1. The %HoldOffset% ms offset will compensate for internal delays by the detection logic with a default value of 150 ms.
2. The %HoldOffset% ms offset is not needed to compensate for the %HoldTime% ms offset.
3. The %HoldOffset% ms offset can also be used to further correct the detected end position of an audio segment in the recording to the left or the right in the timeline.
HoldThreshold: (Par_string, Possible values=-100.0...0.0, Default=-30.0) HoldThreshold in dB for StartNext-Autoducking.
The "HoldThreshold" will be used together with the "HoldTime", the "ThresholdTolerance" and the "HoldOffset" to detect the end position of an audio segment in the recording.
The end of an audio segment in the recording is detected if the signal is below %HoldThreshold% dB for at least %HoldTime% ms and in at least %ThresholdTolerance% percent of the values.
HoldTime: (Par_int, Possible values=0...100000, Default=1000) HoldTime in milliseconds for StartNext-Autoducking.
The "HoldTime " will be used together with the "HoldThreshold" the "ThresholdTolerance" and the "HoldOffset" to detect the end position of an audio segment in the recording.
The end of an audio segment in the recording is detected if the signal is below %HoldThreshold% dB for at least %HoldTime% ms and in at least %ThresholdTolerance% percent of the values.
ThresholdTolerance: (Par_int, Possible values=0..100, Default=75) ThresholdTolerance in percent for StartNext-Autoducking.
1. The "ThresholdTolerance" will be used together with the "AttackTime", the "AttackThreshold" and the "AttackOffset" to detect the start position of an audio segment in the recording.
The start of an audio segment in the recording is detected if the signal is above %AttackThreshold% dB for at least %AttackTime% ms and in at least %ThresholdTolerance% percent of the values.
2. The "ThresholdTolerance" will be used together with the "HoldTime", the "HoldThreshold" and the "HoldOffset" to detect the end position of an audio segment in the recording.
The end of an audio segment in the recording is detected if the signal is below %HoldThreshold% dB for at least %HoldTime% ms and in at least %ThresholdTolerance% percent of the values.

\TurboPlayer\GUI\Windows\CFM\1

EnableMultiTrack: (Par_int, Possible values=0,1, Default=0) 0: CrossfadeMixer will apear with 1 track.
1: CrossfadeMixer can be used with 2 tracks for creating crossfades. (Valid since: 11.01.2007)

HideButton: (Par_int, Possible values=0,1, Default=1) 0 = HideButton is disabled
1 = HideButton is enabled (Valid since: 09.11.2006)
MoveFadepointsWithMarkpoints: (Par_string, Possible values=Yes, No, 1, 0, Default=No) Move the fade points together with the mark points.
Note: The behavior can be toggled holding SHIFT.
Rect: (Par_string) Position and size of the window (Valid since: 09.11.2006)
ShowAmplification: (Par_int, Possible values=0,1, Default=0) - (Valid since: 09.11.2006)
ShowAmplificationIn: (Par_int, Possible values=0,1, Default=0) - (Valid since: 09.11.2006)
ShowAmplificationOut: (Par_int, Possible values=0,1, Default=0) - (Valid since: 09.11.2006)
ShowEditPositions: (Par_int, Possible values=0,1, Default=1) - (Valid since: 09.11.2006)
ShowFadeIn: (Par_int, Possible values=0,1, Default=1) - (Valid since: 09.11.2006)
ShowFadeOut: (Par_int, Possible values=0,1, Default=1) - (Valid since: 09.11.2006)
ShowIntro: (Par_int, Possible values=0,1, Default=1) - (Valid since: 09.11.2006)
ShowLinkIn: (Par_int, Possible values=0,1, Default=1) - (Valid since: 09.11.2006)
ShowLinkOut: (Par_int, Possible values=0,1, Default=1) - (Valid since: 09.11.2006)
ShowMarkIn: (Par_int, Possible values=0,1, Default=1) - (Valid since: 09.11.2006)
ShowMarkOut: (Par_int, Possible values=0,1, Default=1) - (Valid since: 09.11.2006)
ShowOutro: (Par_int, Possible values=0,1, Default=1) - (Valid since: 09.11.2006)
ShowOverviewTrack: (Par_int, Possible values=0,1, Default=0) - (Valid since: 09.11.2006)
SkipbackAfterLinkoutSet: (Par_int, Default=0) A time in second which is only used by OnAir TrackMixer. It is the time, the soundhead is positioned back if the user sets the LinkOut during playout. (This parameter is not available in the settings dialog of TurboPlayer and can only be configured directly in the registry.)
StartPlayOnDrop: (Par_int, Possible values=0,1, Default=1) - (Valid since: 09.11.2006)
ZoomRange: (Par_int, Default=15) - (Valid since: 09.11.2006)

\TurboPlayer\GUI\Windows\DateTimeInfo*
* = 1...n the datetime window number.This section stores properties for the date and time frames in the GUI.Parameters are eays to set in the GUI (->menu ->settings).

\TurboPlayer\GUI\Windows\Jingles*
* = 1...n the jingle window number.Section stores properties for the jingle window.Parameters are eays to set in the GUI (->menu ->settings).

\TurboPlayer\GUI\Windows\List*
* = 1...n the showlist window number.Section stores properties for the showlist.Parameters are eays to set in the GUI (->menu ->settings).

\TurboPlayer\GUI\Windows\ModeButton*
* = 1...n the modebutton number.Section stores properties for modebuttons (=PlayFadings, Pause, Activation, Ghost, etc).Parameters are eays to set in the GUI (->menu ->settings).

\TurboPlayer\GUI\Windows\Player
Section stores properties for each player used in TurboPlayer, as well as generic properties that are applied to all players.

\TurboPlayer\GUI\Windows\Player*
* = 1...n the player number.Section stores properties for the player.Parameters are eays to set in the GUI (->menu ->settings).

\TurboPlayer\GUI\Windows\Selection*
* = 1...n the selection window number.Section stores properties for the selection window (program, show, date, jingle selection).Parameters are eays to set in the GUI (->menu ->settings).

\TurboPlayer\GUI\Windows\StatusLine*
* = 1... n the statusline number.Section stores properties for the statusline.Parameters are easy to set in the GUI (->menu ->settings).

\TurboPlayer\GUI\Windows\TimeInfo*

* = 1... n the timeinfo window number.Section stores properties for the timeinfo window (NextFixedStart, RemainNextFixedStart, RemainGroup, GapOverlap, etc).Parameters are easy to set in the GUI (->menu ->settings).

\TurboPlayer\GUI\Windows\UserButton*

* = 1... n the userbutton number.Section stores properties for the userbutton (=fire eventout).Parameters are easy to set in the GUI (->menu ->settings).

\TurboPlayer\GUI\Windows\WndSelection*

* = 1... n the WndSelection window number.Section stores properties for the WndSelection window (=contains Prelisten, CrossfadeMixer, Text, Infotext frames and pushbuttons).Parameters are easy to set in the GUI (->menu ->settings).

\TurboPlayer\GUIxxxx

You can create multiple GUI keys which can serve as template. E.g. to provide templates for different screen resolutions create keys like: "GUI_1024x768", "GUI_1600x1200". If TurboPlayer starts and no key "GUI" can be found, it will present the user the list of all found keys which start with "GUI". If the user selects one of them, TurboPlayer will copy this key to a local "GUI" parameter and continue with its startup.

\TurboPlayer\IO

Each subkey represents an IO module.

\TurboPlayer\IO\...

-

Active: (Par_binary, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=TRUE) Enables or disables the IO module. (Valid since: 01.05.2004)
IOModule: (Par_string) Specifies the location of the IO module. That is the filename of the corresponding DLL. (Valid since: 01.05.2004)

\TurboPlayer\IO\...\Settings

This key and all its subkeys contain the module-specific settings. You should not change them in the registry but only in the configuration dialog of the module, which is accessible from the general settings dialog of TurboPlayer.

\TurboPlayer\IO\TIODiamond\Settings

Settings for the TIODiamond.dll which uses a serial protocol for communication with LAWO mixer consoles or mixer consoles which are based on the LAWO Diamond protocol like Mandozzi.
CloseFaders: (Par_string, Possible values=0,1, Default=1) 1: Close faders when closing the application (Valid since: 27.09.2010)
LogPath: (Par_string) Specifies a logging folder for the Diamond DLL (Valid since: 27.09.2010)
MinimumOpenLevel: (Par_string, Possible values=-1...-90) The mixer consoles are designed to start reacting on a fader move when the opening action exceeds a trigger level. Typical values are located around -60 and -70 dB. The correct settings must be evaluated from the mixer console.
This parameter converts all values which are delivered from TurboPlayer and which are smaller than the configured value to the value. (Valid since: 27.09.2010)
MixerMode: (Par_string, Possible values=Diamond,Zirkon, Default=Diamond) Protocol version of TIODiamond.dll (Valid since: 27.09.2010)
SetNextText: (Par_string, Possible values=0,1, Default=0) - (Valid since: 27.09.2010)
SetProtocol: (Par_string, Possible values=0,1, Default=1) - (Valid since: 27.09.2010)
SourceOffset: (Par_string, Default=7) - (Valid since: 27.09.2010)
Timeout: (Par_string, Default=5000) Mixer timeout in ms (Valid since: 27.09.2010)

\TurboPlayer\IO\TIODiamond\Settings\SignalProcessing

Used by TIODiamond.dll

Music: (Par_string, Possible values=TurboMusic) This value equals to the technical name of the subsection of TurboPlayer for the corresponding ChannelsOut configuration. (Valid since: 08.04.2008)
Speech: (Par_string, Default=TurboSpeech) This value equals to the technical name of the subsection of TurboPlayer for the corresponding ChannelsOut configuration. (Valid since: 08.04.2008)

\TurboPlayer\IO\TIOGpio\Settings

Settings for the IO module used for communication with PclComServer or PCL Service.
UsePclDrvDll: (Par_string, Possible values=0,1, Default=0) The TIOGpio.dll is able to run with both PclComServer and PCL Service.
0 = Using PclComServer
1 = Using PCL Service (TurboPlayer.exe must have access to PclDrv.dll) (Valid since: 04.08.2009)

\TurboPlayer\Keystrokes

In this section TurboPlayer stores all settings for key shortcuts. Please do not edit these values manually. Use the general settings dialog / key shortcuts tab of TurboPlayer.

\TurboPlayer\Keystrokes

All keystrokes are stored here.The key is the keystring in the format [Ext+][Shift+][Ctrl+][Alt+]keycode, e.g. Ctrl+112 for Ctrl and F1 pressed;The value is the command - either an internal command beginning with prefix TP_ and parameters in brackets, e.g. TP_StartStop(Jingles, 1),or an eventout (defined in TurboPlayer|EventsOut) that will be executed.Keystrokes only can be assigned in the GUI (->system menu ->settings | Keystrokes).

\TurboPlayer\Keystrokes\Hook

All keystrokes which are hooked are stored here.For a description of the key format see parent section.The value is ignored - the presence of the keystring makes the key to be hooked.Hooked keys only can be assigned in the GUI (->system menu ->settings | Keystrokes).

\TurboPlayer\Lines

This key contains the line and channel definitions. See the chapters about lines and channels in the TurboPlayer technical manual for more information.

\TurboPlayer\Lines\...

This key contains the settings of one line. The line number can appear anywhere in the key name, the first number found is interpreted as line number (example: Line-1). The line number must be greater or equal than 1, but need not be numbered consecutively.If there are one or more "channel" subkeys below a line it means that the line is an internal line because it has internal audio channels assigned. External lines do not have channel subkeys.
AutoMode: (Par_string) This value is identically to "Mode" but it does apply in automatic-mode only. If nothing is specified, the same value applies for live-assist and automatic mode.
CloseDelayActive: (Par_string, Default=250) This parameter gives the time in milliseconds, TurboPlayer waits when an element was stopped until the corresponding line/fader is closed by TurboPlayer. A little wait time is desirable to avoid that the end of an element might be cut off too early by the console. This parameter is also applied as delay time before TurboPlayer decides, whether a line must be closed if a new element might possibly start on the same line (e.g. because another channel is mixed to the same line or because another element is queued/joined on the same channel). Therefore, don't make this value too small or you risk to disturb the start of sequenced elements in some cases. As a general rule of thumb: do not make this value smaller than the value for OpenDelayActive. (Valid since: 23.10.2023)
CloseDelayPassive: (Par_int, Default=see comment) If this parameter is 0 TurboPlayer will stop a running element immediately when a line/fader is being closed. In some cases mixer consoles with motor faders send a "line is closed" followed by a "line is open" only a few milliseconds after the first report. Typically this is the case if TurboPlayer tries to move a fader and the user detains the fader. To avoid that Turbo stops the running element you can set this parameter to something like 100ms. TurboPlayer will wait this delay time when a line is closed before it stops the running element. If the line is re-opened before the stop is executed, no stop/start will occur. By default this parameter is identical to the value of the parameter "OpenDelayActive" for lines with mode "variable" (=motor faders) and is 0 by default for all other modes.

Gain: (Par_string) This value specifies which module should handle the gain. The gain is a level correction which can be specified with every element in the "Fade_Amplification" field and which is handled independent of other fade values. If your mixer console supports setting an input gain for each line you set the parameter to "Mixer" or "Console". If it doesn't, you can set the parameter to "MultiPlayer" or "AudioPlayer" or "MP". Then the MultiPlayer will be told to add the gain value to each fade value. Or you can let the parameter be empty or set it to "None", which will cause TurboPlayer to ignore the gain.
Attention: at the moment you must set this value if you want fadings to be executed correctly !
KickOut: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) If TurboPlayer has to start an element, but there are no more free channels, it will queue the element behind the running element which has the shortest remaining time. This is the standard behaviour, but you can set the KickOut parameter of a line to TRUE. Then TurboPlayer will stop one of the elements running on the line and will use the free channel to start the new element.
MinimumOpenLevel: (Par_int, Default=-98) With this parameter you can configure for each line/fader individually the minimum level which can be sent to the mixer console and which will yield an open line. The standard value is -98 dB, but such a low level is not supported by most mixers. (Instead the fader is at the lowest position and the line is closed, if such a low level is being sent to the mixer.)
MixerSource: (Par_string) If your mixer console does not support the selection of mixer sources via its interface (e.g. GPIO) you can use external lines nevertheless. It is necessary to do the assignment of mixer sources to lines either with this parameter or via the macro command TP_SelectMixerSource. The assignment with this parameter is done with every startup, reset and activation of the "SelectSources" mode. Of course the mixer source used must be known to TurboPlayer, meaning: there must be a key with the same name below TurboPlayer\MixerSources\... This parameter makes the line "external", it is not possible to have a subkey "Channel..." at the same time, because this would make the line "internal".
Mode: (Par_string) With this value you can define whether a line can be opened and whether there is a motor fader. The following values are possible:
- None: The line does not report its state and it cannot be opened/closed by TurboPlayer.
- Passive: The line can report its state (open/close) but it cannot be opened/closed by TurboPlayer.
- Active: The line can report its state and it can be be opened/closed by TurboPlayer, but it does not have motor faders. This mode should be used when the user really moves a fader to open/close the line.
- Active/OnOff: The mixer console can report the line state and the line can be opened or closed by TurboPlayer. This mode should be used if there is no fader at all or if the fader is irrelevant to TurboPlayer, meaning: the user will press On/Off buttons to open/close the line.
- Active/Variable: The line can report its state (open/close), it can be opened/closed by TurboPlayer and the faders can be moved by TurboPlayer.
See also in the TurboPlayer technical manual the chapter about faders.
Name: (Par_string) Here a user friendly name can be specified, which is used in error messages or in logging.
OpenDelayActive: (Par_string, Default=0) If TurboPlayer requests the mixer console to open a line, it will take some time until the console reports the line as open. With this parameter TurboPlayer can be told to wait some time before it starts playing. The time you specify should be quite sufficient for the mixer console to open a line in most times. If it is not always fast enough, TurboPlayer will continue to wait up to [n] times, every time waiting the specified delay. Nevertheless a red warning line will be written to the error log, so you can adjust the delay if it is too small. The repeat value [n] can be appended optionally to the delay time after a star. Example: to wait 6 times 100 ms enter: "100*6". The standard value for [n] is 3 and should only be adjusted for very slow mixer consoles.
Hint: see also the parameter "PreStartOpDelay".
OpenDelayPassive: (Par_int, Default=0) If a presenter opens a fader manually, it takes some time to move the fader to a position in which sound is audible. With this parameter TurboPlayer can be told to wait some time before it starts playing.
PFLNumberForGUIPrelisten: (Par_string) If you want to activate PFL when a GUI starts its own prelistening (EasyPlayer or CrossfadeMixer) you can assign lines and corresponding PFL bus numbers in this parameter. You can enter assignments for multiple GUIs which must be separated by commas. Each assignment has the form: <GUI number>:<PFL number>. For example to assign both GUI 2 and 3 to PFL bus 2 enter: "2:2,3:2". If no assignment is done for a GUI, PFL will not be activated when this GUI starts prelistening. If you want to assign a GUI to a line but do not want to use PFL, set the PFL

number to 0 (e.g. 2:0). Such an assignment might be necessary to query the prelisten state with "DataOfLine()".
PreferredStartType: (Par_string) This value is only needed if you have multiple internal channels assigned, which belong to different rundown lists, e.g. jingles and the main show list. With this value you specify for which list a start should be executed, when the line is opened spontaneously.
PrelistenFades: (Par_string, Possible values=MultiPlayer, AudioPlayer, MP, Mixer, Console, <empty>, Default=<empty>) By default no fades are played when prelistening. You can activate fades for prelistening via MultiPlayer (not CFM/EP). Set the parameter to "MultiPlayer", "AudioPlayer" or "MP" and MultiPlayer will do the fades. (The values "Mixer" or "Console" are possible too, but as a fader normally stays closed during prelisten this is rarely useful.) Be aware of two things: first, the fades might not be identical to fades during on-air playout when your console handles fades differently than MultiPlayer. And second, a fade-curve (or fade-trapezoid) might have ranges with a very low level (typically -98 dB before mark-in and behind mark-out). Playout in these ranges will not be audible.
PreStartOpDelay: (Par_int, Default=-) Before starting an element TurboPlayer sometimes needs to perform one or multiple operations. This can be: set the input gain of the mixer console, set the signal processing or any generic operation as requested by an IO module. In this case TurboPlayer will delay the start by the time specified with this value in [ms].
For the most common pre-start operation to open a line/fader there is the extra parameter "OpenDelayActive".
If multiple operations are necessary (opening a fader and other pre-start operations) then the maximum of both delay values is used. (Valid since: 16.12.2020)
SetSoundProcessing: (Par_string, Possible values=Yes, No, TRUE, FALSE, 1, 0) If activated TurboPlayer sets the sound processing (optimod) for this line. Default is "no" for all dedicated prelisten lines and "yes" for all other lines.
UseForTransitions: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) With this flag you can declare a line to be a transition line. All elements belonging to a single transition can be played on this line. In other words: these elements are treated as a unit. The transition elements must be marked with the flag "Time_StartOnTransitionChannel". Enough channels must have been assigned to this line so that all elements of a transition which are intended to be played simultaneously can be played. For more information see TurboPlayerTechManual chapter Concepts/Transition lines. (Valid since: 23.01.2017)

\TurboPlayer\Lines\...\...

This key contains the settings of one internal channel. The channel number can appear anywhere in the key name, the first number found is interpreted as channel number (example: Channel-1). The channel number must be greater or equal than 1 and must be identical to the corresponding MultiPlayer audio device.
Class: (Par_string) Here you can define a class-channel assignment. See the chapter about channels in TurboPlayer technical manual for more information.
GUI: (Par_string) This parameter is only used for prelisten channels. You can specify the GUI numbers (as a comma-separated list) of GUIs which uses this channel for prelistening. See the chapter about prelistening and PFL in the TurboPlayer technical manual. Warning: if you do not specify a GUI parameter at all, all known GUIs are automatically assigned to the first prelisten channel. If you have at least one GUI parameter, every GUI which wants to use the engine prelisten must be listed, otherwise the GUI cannot start an engine prelisten.
ListPosition: (Par_string) Here you can define a list position-channel assignment. See the chapter about channels in TurboPlayer technical manual for more information.
Name: (Par_string) Here a user friendly name can be specified, which is used in error messages or in logging.
NeutralEndpointOnly: (Par_string, Possible values=true/yes/1/false/no/0, Default=false) This parameter refers to so-called “neutral” elements within an EndpointStory. An EndpointStory can contain all elements for a regionalized leg, including one or more neutral elements, which are not played for a specific region but for the regular output for all listeners. Depending on the audio routing during the regionalized leg it might be necessary that such a neutral element must be played on a specific channel, which is reserved for that purpose. Playing a regular element on such a restricted channel or vice versa a neutral element on a regular channel would mean that the wrong listeners would hear the wrong content or they would not hear anything at all. Therefore, this is a hard restriction, which is always obeyed and also cannot be avoided by the user. A user has to start elements on the right channel type.

Alternatively, you can use the parameter name "NeutralRegionOnly". (Valid since: 31.10.2022)
NeutralRegionOnly: (Par_string, Possible values=true/yes/1/false/no/0, Default=false) This is an alternative parameter name for "NeutralEndpointOnly". (Valid since: 31.10.2022)
PFLNumber: (Par_int) This parameter is required for prelisten via on-air channels and is optional for dedicated prelisten channels. You can specify one of the PFL bus numbers your mixer console provides. Hint: this parameter is used for prelistening via engine (MultiPlayer), but not for prelistening via GUI (EasyPlayer, CrossfadeMixer). For more information see the chapter about prelistening and PFL in the TurboPlayer technical manual.
RundownListType: (Par_string) Each logical channel must be assigned to exactly one rundown list. The following rundown lists are known: Show, Jingles, Prelisten, Deleted (cannot be used here), Stack1, Stack2,...,Stack15
Type: (Par_string, Possible values=Audio, Video, Default=Audio) Defines the channel type. Possible values are Audio and Video. Other types like external or live are assigned automatically and need not be specified.
UseForAutoPrepare: (Par_string, Possible values=Yes, No, TRUE, FALSE, 1, 0, Default=Yes) If you activate the AutoPrepare feature for a rundown TurboPlayer will use all channels which are assigned to the rundown for auto-preparing elements to them. If you want to exclude a channel you must set this parameter to No/false/0. This parameter might be useful if you want to keep a channel free for timed or special starts.

\TurboPlayer\Logging

-
DebugLog: (Par_string) With this value you can switch on extended logging for debugging purposes. Enter a comma-separated list of keywords. At the moment the following keywords exist:
"Filler" enables extended information from the filler routine.
"BroadcastMessages" enables extended information about BCS broadcast messages.
"FileAccess" enables extended information about every file access - but so far this keyword is only evaluated by the TurboPlayer GUI.
MessageIDs: (Par_string) This parameter is a comma-separated list of all message IDs you want to log into the protocol of type "BCE-Messages". Hint: the internal parts of TurboPlayer use a messaging system for the communication. In TurboPlayers technical manual you can find a chapter about message logging. It contains a list of all defined messages with a short description. (Valid since: 01.05.2004)

\TurboPlayer\MediaDirectoriesToIgnore

In this key you can list all media directories which should be ignored by TurboPlayer. The names of the parameters do not matter. The value of each parameter must be the start string of a media path. If at least one value is given, each filename of an element is string-compared with all listed values. If a filename contains a path listed here, it is ignored. If the key is empty, all filenames are used. The key "MediaDirectoriesToUse" is preferred.

\TurboPlayer\MediaDirectoriesToPrefer

In this key you can list directories which are preferred. In addition you can specify local paths as replacement for UNC paths. The names of the parameters, which list directories must be: "Dir1", "Dir2",...
ApplyInEngine: (Par_string, Possible values=Yes, No, true, false, 1, 0, Default=No) With this parameter you can activate that the TP engine takes the list of directories to be preferred into account, too. The TP-GUI will do this always. This parameter exists only for backward compatibility: an evaluation by the TP engine is only available since version 6.0 and older versions did not evaluate these directories. In order to not change the behaviour of an older setup, you need to activate the utilization by the engine explicitly. (Valid since: 04.05.2022)
Dir<n>: (Par_string, Default=-) You can either specify a single path or two paths separated with a pipe '|'. If two paths are given, the second one should be the local replacement of a given UNC path. For example, a path specification of "\\TPComputer\Share\Audio" could be replaced by the local path "D:\Audio". This avoids that the local access to a file in this path is routed through the network stack of Windows (which might be troublesome in case of a network failure).
For the TP-GUI the given directories are always valid, for the TP engine they must explicitly be activated with the parameter ApplyInEngine=true.

The GUI takes the given directories into account when selecting a filename for GUI prelisten. The GUI will first look for a file in the preferred directories. If a file cannot be found there, all specified filenames are tried out.
For the TP engine the implementation is slightly different. Reason is that the engine does not look for existing files itself. It simply assembles a list of file names and sends this list to MultiPlayer, which then picks one of the given files. Therefore, the implementation works like this: If at least one of the file names in the metadata of an element matches a listed directory, all other file names, which do not match one of the preferred directories, are not sent to MultiPlayer. Only if MultiPlayer reports that none of the files of the filtered list can be found, TurboPlayer will make a second load request to MultiPlayer with the full/original list of file names. This behaviour has two disadvantages: First, it might result in delays of some seconds when an element cannot be loaded with the filtered list of file names. Second, if the initial loading of files from the local hard disk succeeds and then later the local files become inaccessible, MultiPlayer cannot fall back to files in the network, because MultiPlayer does not have knowledge of these files.
PAY ATTENTION to not make one of the preferred directories unusable by the settings of MediaDirectoriesToUse/MediaDirectoriesToIgnore! The behaviour for this case is undefined. (Valid since: 04.05.2022)

\TurboPlayer\MediaDirectoriesToSupplement

In this key you can list additional media directories for elements. This is used with database directories which are mirrored (e.g. with DigAlign). Though the metadata of a database element specifies only a single filename, additional filenames can be added on the fly. This can increase playout safety. The names of the parameters should be "Dir1", "Dir2",... Each value has the format: "Specified DB path|supplemental path". Example: "\\Fileserver\DB\Sound1|\\Backup\DB\Sound1". TurboPlayer performs a pure text search-and-replace when generating supplemental filenames. The first part of the value is searched, the second one replaces the found string. (You can test the correctness of a value by entering some DB filenames in a text editor and call the search-and-replace function to see if all generated filenames are valid.) Attention: only specify directories which are accessible normally and which contain all the files. It is not a good idea, e.g., to specify a directory which is rarely online.

\TurboPlayer\MediaDirectoriesToUse

In this key you can list all media directories which should be used by TurboPlayer. The names of the parameters do not matter. The value of each parameter must be the start string of a media path. If at least one value is given, each filename of an element is string-compared with all listed values. If a filename contains a path which is not listed here, it will be ignored. If the key is empty, all filenames are used.

\TurboPlayer\MixerSources

This key contains the definition of mixer sources and groups. See the chapter about mixer sources in the TurboPlayer technical manual for more information.
AutoSelect: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=TRUE) By default TurboPlayer automatically calls the mixer console to select mixer sources to lines whenever it seems necessary - that is when there is an external element in the preload range which specifies a mixer source. This behaviour is useful if your mixer console supports mixer sources. If it does not (e.g. with a GPIO interface) you have to do the assignment of mixer sources to lines via configuration (see parameters of the lines) or via macro command (TP_SelectMixerSource). In this case you should deactivate automatic mixer source selection by setting this parameter to false.
BitmapLive: (Par_string) Here you can specify a Windows bitmap file which is being displayed for the special mixer source <Live>. Alternatively you can specify a single letter only. TurboPlayer and DigAIRange will display this letter instead of a bitmap. For more information about supported formats, prerequisites and general information about how to create images for mixer sources, please have a look into chapter 1.2.5 "Customizable image files" of BCSTechManual.
BitmapNoSource: (Par_string) Here you can specify a Windows bitmap file which is being displayed for the special mixer source <NoSource>. Alternatively you can specify a single letter only. TurboPlayer and DigAIRange will display this letter instead of a bitmap. For more information about supported formats, prerequisites and general information about how to create images for mixer sources, please have a look into chapter 1.2.5 "Customizable image files" of BCSTechManual.
BitmapSystem: (Par_string) Here you can specify a Windows bitmap file which is being displayed for the special mixer source <System>. Alternatively you can specify a single letter only. TurboPlayer and

DigAIRange will display this letter instead of a bitmap. For more information about supported formats, prerequisites and general information about how to create images for mixer sources, please have a look into chapter 1.2.5 "Customizable image files" of BCSTechManual.
DeselectUnknown: (Par_binary, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) When the mixer console reports a mixer source as selected, which is not known to TurboPlayer (that means it is not listed in this key), this information is either ignored (parameter is FALSE or not exitisting) or the mixer source will immediately be deselected (parameter is TRUE).
GUIForSetNext: (Par_int, Default=1) Possibly you can configure your mixer console in a way that there is something like a "Set next" button for each fader. The IO module must be configured to send a "Set next" message when one of these buttons is pressed. When TurboPlayer receives the message, it will assign the mixer source, which is known for the corresponding fader, to the cursor element. As there can be multiple lists and multiple GUIs, each with an own cursor, the message is ambigous. The rundown list cannot be specified, the show list is taken always. But you can configure which GUI is taken into account, when the cursor position is evaluated.
OrderLive: (Par_int, Possible values=0..n, Default=0) It defines the sort order of the mixer source <Live> shown in the context menu;
If this value is not defined it stands above all other mixer source groups in the menu.

If multiple mixer sources have the same order value they are sorted in alphabetical order. (Valid since: 23.2.2011)
OrderNoMixerSource: (Par_int, Possible values=0..n, Default=0) It defines the sort order of the mixer source <No mixer source> shown in the context menu;
If this value is not defined it stands above in the menu.

If multiple mixer sources have the same order value they are sorted in alphabetical order. (Valid since: 23.2.2011)
OrderSystem: (Par_int, Possible values=0..n, Default=0) It defines the sort order of the mixer source <System> shown in the context menu;
If this value is not defined it stands above all other mixer source groups in the menu.

If multiple mixer sources have the same order value they are sorted in alphabetical order. (Valid since: 23.2.2011)
SetDataLive: (Par_string) In this value you can define a data string which is applied when the special mixer source <Live> is being selected. As format you can use either any XML string with the data tags on the third level (below a dummy root and a dummy object node) or a section string. If you use a section string, note that the section names are NOT the same as in DigaSystem section strings but instead like the XML tagnames of the BCS system ! Besides they are case-sensitive. (For example it is [Title] and [News_Ressort] instead of [TITLE] and [RESSORT])
SetDataSystem: (Par_string) In this value you can define a data string which is applied when the special mixer source <System> is being selected. As format you can use either any XML string with the data tags on the third level (below a dummy root and a dummy object node) or a section string. If you use a section string, note that the section names are NOT the same as in DigaSystem section strings but instead like the XML tagnames of the BCS system ! Besides they are case-sensitive. (For example it is [Title] and [News_Ressort] instead of [TITLE] and [RESSORT])
SetEventsLive: (Par_string) This can be a comma-separated list of fully specified generic events (below key EventsOut) which are applied whenever the special mixer source <Live> is selected. (Valid since: 16.01.2006)
SetEventsSystem: (Par_string) This can be a comma-separated list of fully specified generic events (below key EventsOut) which are applied whenever the special mixer source <System> is selected. (Valid since: 16.01.2006)

\TurboPlayer\MixerSources\...

The definition of a mixer source group.
Bitmap: (Par_string) Here you can specify a Windows bitmap file which is being displayed for the mixer source group. Alternatively you can specify a single letter only. TurboPlayer and DigAIRange will display this letter instead of a bitmap. For more information about supported formats, prerequisites and general information about how to create images for mixer sources, please have a look into chapter 1.2.5 "Customizable image files" of BCSTechManual.

MediaType: (Par_string) The default media type for new elements, inserted as a result of a spontaneous opened external fader.
Name: (Par_string) The user friendly name displayed in the GUI of TurboPlayer or DigAIRange. If no name is defined, the key name of the mixer source group is used.
Order: (Par_int, Possible values=0..n, Default=0) It defines the sort order of the mixer source group shown in the context menu;
If this value is not defined it stands above all other mixer source groups (with order > 0) in the menu.

If multiple mixer sources have the same order value they are sorted in alphabetical order. (Valid since: 23.2.2011)
Program: (Par_string) This is a comma-separated list of service names for which the mixer source group should be available. The specification of a single star * means: it is available in all services.
SetData: (Par_string) In this value you can define a data string which is applied when the corresponding mixer source group is being selected. As format you can use either any XML string with the data tags on the third level (below a dummy root and a dummy object node) or a section string. If you use a section string, note that the section names are NOT the same as in DigaSystem section strings but instead like the XML tagnames of the BCS system ! Besides they are case-sensitive. (For example it is [Title] and [News_Ressort] instead of [TITLE] and [RESSORT])
SetEvents: (Par_string) This can be a comma-separated list of fully specified generic events (below key EventsOut) which are applied whenever the mixer source group is selected. (Valid since: 16.01.2006)
Source: (Par_string) The two data fields "Source" and "MixerSource" are not independent. With this parameter you can specify a source which is set, if the user changes the mixer source of an element to one of the mixer sources below this key. For example, for all mixer sources of a group called "CD", it would be useful to set the source to CD, too. This parameter is also used when TurboPlayer creates a new "spontaneous" element.

\TurboPlayer\MixerSources\...\...

The definition of a mixer source.
Bitmap: (Par_string) Here you can specify a Windows bitmap file which is being displayed for the mixer source. Alternatively you can specify a single letter only. TurboPlayer and DigAIRange will display this letter instead of a bitmap. If the value is not defined, the bitmap of the parent group will be used instead (you can override the use of the group bitmap by specifying an empty bitmap value for a source). For more information about supported formats, prerequisites and general information about how to create images for mixer sources, please have a look into chapter 1.2.5 "Customizable image files" of BCSTechManual.
Class: (Par_string) The default class for new elements, inserted as a result of a spontaneous opened external fader.
DefaultDuration: (Par_string) The default duration (format: HH:MM:SS.MMM) for new elements, inserted as a result of a spontaneous opened external fader.
DefaultTitle: (Par_string) The default title for new elements, inserted as a result of a spontaneous opened external fader.
MediaType: (Par_string) The default media type for new elements, inserted as a result of a spontaneous opened external fader.
MixerSource: (Par_string) The identifier used in the communication with the mixer console. This parameter is needed, if the identifier has spaces at the end, because spaces at the end of key names are not possible in the registry. If this parameter is not defined, the key name of the mixer source is used.
Name: (Par_string) The user friendly name displayed in the GUI of TurboPlayer or DigAIRange. If no name is defined, the key name of the mixer source is used.
Order: (Par_int, Possible values=0..n, Default=0) It defines the sort order of the mixer sources within a mixer source group - shown in the context menu;
If this value is not defined it stands above all other mixer sources of the mixer source group in the menu.

If multiple mixer sources have the same order value they are sorted in alphabetical order. (Valid since: 23.2.2011)
Permanent: (Par_binary, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) Defines whether the mixer source is selected permanently or sporadic: permanently means during startup, sporadic

means when needed by an element. See TurboPlayer technical manual, chapter "Mixer sources" for more information.
Program: (Par_string) This is a comma-separated list of service names for which the mixer source should be available. The specification of a single star * means: it is available in all services.
SetData: (Par_string) In this value you can define a data string which is applied when the corresponding mixer source is being selected. As format you can use either any XML string with the data tags on the third level (below a dummy root and a dummy object node) or a section string. If you use a section string, note that the section names are NOT the same as in DigaSystem section strings but instead like the XML tagnames of the BCS system ! Besides they are case-sensitive. (For example it is [Title] and [News_Ressort] instead of [TITLE] and [RESSORT])
SetDataUnexpected: (Par_string) Like parameter "SetData" but this parameter is applied when TurboPlayer inserts a new spontaneous element as reaction to an unexpectedly opened external fader.
SetEvents: (Par_string) This can be a comma-separated list of fully specified generic events (below key EventsOut) which are applied whenever the mixer source is selected. If nothing is specified here, the corresponding setting of the mixer source group will be applied. (Valid since: 16.01.2006)
Source: (Par_string) The two data fields "Source" and "MixerSource" are not independent. With this parameter you can specify a source which is set, if the user changes the mixer source of an element to one of the mixer sources. For example, for a mixer source called "CD", it would be useful to set the source to CD, too. This parameter is also used when TurboPlayer creates a new "spontaneous" element.
TreatUnexpected: (Par_binary, Possible values=TRUE, FALSE, Yes, No, 1, 0) The standard behaviour of TurboPlayer is to ignore movements of external faders. If a mixer source is being selected to such a fader and the "TreatUnexpected" parameter of the mixer source is TRUE, TurboPlayer will treat a fader opened/closed event. That means if the "next" pointer of any rundown list is on an external element with the right mixer source, this element will be "started". If there is no such element, a new element will be inserted before it is started. Some metadata of the new element can be defined, see the other parameters of this key.

\TurboPlayer\MixerSources\...\...\...

In this level sub mixer sources can be defined. See TurboPlayer tech manual, chapter Concepts / Mixer sources for more information.
MixerSource: (Par_string) Like the same parameter for a mixer source. This string is used in the communication with the console. If not present, the key name is used.

\TurboPlayer\MultiRec

This section contains settings for the MultiRecorder.ocx within TurboPlayer. (Hint: the OCX is loaded twice, the first time directly for the "EasyPlayer" used for GUI prelistening and the second time it is loaded within CrossfadeMixer or OnAirTrackMixer. The settings apply for both controls.) Most of the possible values are identical to the section "DBM\MultiRec". Below only additional values or differences are described.
AsioMonitoring: (Par_string, Possible values=0, 1, false, true, Default=true) Enables/Disables monitoring while recording when using ASIO cards as recording source. If enabled the input audio is routed to the output while recording is active. This does not enable true hardware monitoring but a redirection of the input with low latency.

This parameter is similar to the one with same name in DBM|MultiRec, Multitrack|Settings and DigAIRange|MultiRec. (Valid since: 26.10.2017)
AsioPlayBufferCount: (Par_int, Default=8) Number of playback buffers for ASIO output device. (Valid since: 05.12.2016)
AsioPlayBufferSize: (Par_int, Default=8192) Size of the audio buffer in stereo samples for the playback function with ASIO devices. (Valid since: 05.12.2016)
AudioBoardDescriptionIn: (Par_string) This the Windows audio device which is being used by MultiRec.ocx within the GUI for audio recording. Please use always the general settings dialog, tab Prelisten for changing this value.
AudioBoardDescriptionOut: (Par_string) This the Windows audio device which is being used by MultiRec.ocx within the GUI (EaysPlayer, CrossfadeMixer and OnAIR TrackMixer) for audio playout. Please use always the general settings dialog, tab Prelisten for changing this value.
AudioBoardOut: (Par_int, Default=0) This is a deprecated parameter. Please use "AudioBoardDescriptionOut". (Valid since: 09.11.2006)

AudioFormat: (Par_string, Possible values=Linear, MUSICAM, Default=MUSICAM) The format of the audio data used for recordings. The same value in the "PlayRec" section is read too.
Bitrate: (Par_int, Possible values=32,48,56,64,80,96,112,128,160,192,224,256,320,384, Default=128) This entry selects the presetting of the total bitrate in kBit/s for recordings. For AudioFormat=Linear this entry is not evaluated. The same value in the "PlayRec" section is read too.
CommonPrelistenType: (Par_int, Possible values=0,1, Default=0) 0 = PFL is handled via MultiPlayer
1 = Prelisten kann be configured using MultiPlayer or MultiRec (Valid since: 20.12.2007)
FileFormat: (Par_string, Possible values=Musifile, Raw, Wavefile, BWF, Default=BWF) The file format of the audio file used for recordings. The same value in the "PlayRec" section is read too.
Logging: (Par_string, Possible values=[DIRECTORY]C:\DigaSystem\Logging\[FILENAME]TP_MulRec_EasyPlayer[TYPE]GLOBAL|EVENTS|METHODS|FILE|ACCESS|DEBUG[DELETEOLD]TRUE[DELETEDAYS]31) Please use settings dialog for configuring this parameter. (Valid since: 20.12.2007)
Logging_CFM_MultiRec: (Par_string, Possible values=[DIRECTORY]C:\DigaSystem\Logging\[FILENAME]TP_MulRec_CFM[TYPE]GLOBAL|EVENTS|METHODS|FILE|ACCESS|DEBUG[DELETEOLD]TRUE[DELETEDAYS]31) Please use settings dialog for configuring this parameter. (Valid since: 20.12.2007)
Mode: (Par_string, Possible values=Mono, Stereo, Joint-Stereo, Default=Mono) This entry selects the presetting of the mode used for recordings. The same value in the "PlayRec" section is read too.
PositionAtElement: (Par_int, Possible values=0,1, Default=1) Position of EasyPlayer GUI
0 = fixed position in TurboPlayer GUI
1 = position at played item (Valid since: 20.12.2007)
PrelistenType: (Par_int, Possible values=0,1,2, Default=0) 0 = MultiPlayer
1 = EasyPlayer
2 = CrossfadeMixer
Parameter only works if CommonPrelistenType is set to 1 (Valid since: 20.12.2007)
RecordFilename: (Par_string, Default=<None>) The filename of the file which is used for recordings within TurboPlayer / OnAirTrackMixer. The file is temporary and will be copied to one of the media directories after the record is finished. If multiple files are recorded OnAirTrackMixer will automatically append continuous numbers.
Sampling: (Par_int, Possible values=16000, 32000, 44100, 48000, Default=44100) This entry selects the presetting of the sampling-frequency in Hz used for recordings. The same value in the "PlayRec" section is read too.
Skin: (Par_int, Possible values=0,1,2,3,4, Default=0) Skin ID for EasyPlayer.

0: Classic,
1: Light gray
2: Black
3: Dpi aware with small buttons without VU meter
4: Dpi aware with big buttons and vertical VU meter (Valid since: 20.12.2007)
StopAtMarkOut: (Par_int, Possible values=yes, no, true, false, 1, 0, Default=no) Determines whether prelisten performed with EasyPlayer will stop at the markout of the prelistened element or will go till the end of the audio. (Valid since: 29.05.2019)
WaveInBufferNb: (Par_int, Possible values=2..64, Default=4) Number of recording buffers for input device.
WaveInBufferSize: (Par_int, Possible values=1024 ... 524288, Default=65536) Size of the audio buffer for the recording function with Multimedia sound cards.
WaveOutBufferNb: (Par_int, Possible values=2 ... 64, Default=8) Number of playback buffers for output device.
WaveOutBufferSize: (Par_int, Possible values=1024 ... 524288, Default=65536) Size of the audio buffer for the playback function with Multimedia sound cards.
WriteBlockSize: (Par_int, Default=0) The number of bytes that is written to the file. If you don't use this parameter (=0) the data is saved in blocks according to the number of bytes delivered from the audioboard. In networks it can be useful to set this tag to the size of your network packets to increase speed and decrease network traffic.
WriteCacheSize: (Par_int, Default=262144) The maximum number of bytes that is stored by MultiRec before data is lost and MultiRec indicates an error event. If your network sometimes has short interrupts MultiRec can cache record data until the size of this cache is reached.

\TurboPlayer\MusicMaster

In this local subkey TurboPlayer stores all settings from the MusicMaster tab of the settings menu. To enable you need to have DIMusicMaster.dll inside the program folder. The DIMusicMaster.xml need to contain a valid configuration of a NEXUS server.

\TurboPlayer\MusicMaster\Filters

This subkey contains all standard filters from the MusicMaster tab of the settings menu.Example:Filter1=KOMPONIST,COMPOSER,Music_Composer,0,,0,1

\TurboPlayer\PreloadedElements

The number of preloaded elements per rundown list. See the chapter about rundown lists in the TurboPlayer technical manual for more information.

\TurboPlayer\RundownLists

Here you can define the settings of the rundown lists. Possible subkeys are: "Show", "Jingles", "Stack1", ... "Stack15"
TracksFollowShow: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) This parameter is only evaluated, if you have additional schedule (regio) tracks loaded onto the stack lists. By default (if this parameter is "No") each rundown list is advanced to the next show individually. That means, if you have sent all elements in one of the lists, it will advance to the next show, independent of any other elements in the other lists which must still be sent. This behaviour is changed by setting the parameter to "Yes". Then TurboPlayer will automatically advance all additional lists together with the main show list, which again is independent of the state of elements in the additional lists.

\TurboPlayer\RundownLists\...

-
AllowRearrangement: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=TRUE) This property can be deactivated in order to disallow user actions, which change anything in the sequence of the corresponding rundown list. This means: insert, delete, copy or move operations are no longer possible. Nevertheless, it is still possible to change the metadata of individual nodes in the rundown. (Valid since: 09.10.2018)
AutoDelete: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, [+ Limit], [+Reschedule], Default=No) If this parameter is "Yes", elements which have been sent are automatically deleted from the rundown list. This can be useful to achieve a "real stack behaviour", meaning that sent elements drop out of the list. This parameter is only available for stack lists. Since build 802 you can add a limitation like this: "Yes,Limit=1". The limit means, the auto-delete function does not delete played elements if there are less or up to the limit elements in the rundown. Default for the limit is 0 which means: even if there is only a single element in the rundown it is deleted after it was played. Since build 1917.0 you can add a new keyword like this: "Yes,Reschedule". If set TurboPlayer will not delete elements which were played, but instead move them to the end of the rundown list and make them playable again.
AutoPrepare: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Yes/Next, Yes/Single, Default=No) If this parameter is "Yes", the next n elements are automatically prepared in the next n free channels which were assigned to the corresponding rundown list. This can be useful in connection with a stack if you want all stack players to be prepared with the next elements.
It is also possible to prepare only the "next element" to a free channel. If this is desired set the parameter to "Yes/Next" or "Yes/Single".
ClearOnReset: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=Yes) By default a rundown list is cleared during a reset, meaning that all elements are removed from the list. If you want to keep the content of a list during reset, this parameter must be "No". This parameter is only available for stack lists if the stack does not follow a track.
CurrentElements: (Par_int, Default=4, since build 506: max (4, PreloadedElements / 2)) With the parameter "PreloadedElements" you can define the number of elements which are preloaded. For a normal sequenced rundown the list starts with the "next element" and contains simply the next following elements until the preload number is reached. Elements below are not preloaded, normally. Sometimes it is necessary to preload some of those elements, which are far in the future, too. This is either the case for elements with an absolute (time-) start or for elements with a relative start if the referenced element will be played soon. Therefore these elements will be preloaded, but they do replace the

normal consecutive elements from the list. TurboPlayer starts to replace the elements at the end of the list first, of course. With this parameter you can limit the number of normal consecutive elements which always stay in the list and cannot be replaced by a time-start or relative-start element.
DistributionEndpoints: (Par_string, Default=<empty>) This parameter is only evaluated if a stack is operating in stack mode "Endpoints". Then it defines the DistributionEndpoints, which should be played by this stack. The names you can use must be one of the keys below the registry key: Common\DistributionEndPoints.
Though probably being extraordinary, you can give a comma-separated list of endpoint names if a stack should play multiple endpoints. In any case pay attention to specify each endpoint in only one stack, not in multiple ones!
If you are using the feature "Regionalization with DistributionEndpoints" it might seem more natural to use the value name "Regions" or even "Region" instead of "DistributionEndpoints". This is possible, but use only one of these values! (Valid since: 31.10.2022)
FillNeutralElement: (Par_string, Possible values=true, yes, 1, false, no, 0, Default=false) This parameter is intended for the use case "Using EndpointStories for regionalization". See chapter 5.3. of TurboPlayerTechManual for more information.
When TurboPlayer distributes the elements for the regions onto the stacks it can happen that the result is an empty EndpointStory in one of the stacks. This means that no element was scheduled for this region. If such an EndpointStory is played there will probably be silence for the affected region. In order to prevent this silence, you can set this parameter to true. Then TurboPlayer will pick the neutral element(s) of the EndpointStory and make a copy in the affected stack. As a result, the neutral element will not only be played multiple times simultaneously – once for the show rundown and one or more times for the affected stack(s). (Valid since: 31.10.2022)
PreloadAroundCursor: (Par_string, Default=No) Normally the user cannot control which elements are preloaded. User control might be desired, especially with rundowns which contain more elements than preload slots are available. If this parameter is activated TurboPlayer will preload the current cursor element. In addition n elements in front of and m elements behind the cursor element can be loaded together with the cursor element. Some settings:
No: function is deactivated
Yes: function is activated, cursor element is preloaded
0: function is activated, cursor element is preloaded
-1,2: function is activated, cursor element is preloaded together with 1 element in front of and 2 elements behind cursor
Very important: when this function is activated the normal preloading mechanism is deactivated! Then it is up to the user to pay attention that needed elements are preloaded, e.g. the next element for a sequenced start, relative elements, absolute starts, ... Therefore it is recommended to use PreloadAroundCursor only for lists in random mode. In sequenced modes it is very dangerous to use this feature! If you want to use it in sequenced modes, pay attention to include some past and future elements (e.g. -4,4) and make the number of "PreloadedElements" significantly bigger than the elements around the cursor. In addition the user must be extremely careful where to place the cursor to make automatic starts or other features like the minimum runtime working. Hint: this feature overrides "PreloadPositionsFixed".
PreloadedElements: (Par_string, Default=20 (24 for rundown Jingles)) The number of preloaded elements in the rundown list. For all rundown lists except the main show list you can specify a negative value but not 0: the absolute number of specified elements is loaded into TurboPlayer (so you can see them in the list and make drag&drop for example) but they are not loaded into MultiPlayer (so they cannot be played and do not occupy MultiPlayer slots). Pay attention to reserve a slightly higher number (~10%, at least 3) more slots in MultiPlayer for the rundown. These additional slots are needed because preloading is done faster than unloading when elements are replaced or the file list of an element changes. The value 0 deactivates the corresponding list as if it was not defined at all.
Amendment: since version 6.1 it is also possible to set this value for the show rundown to -1. In this case TurboPlayer will not preload any element for the show rundown. This special setting is intended for a pure regio TurboPlayer, which should run as a slave for a main TurboPlayer in the background.
PreloadPositionsFixed: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=FALSE) If this parameter is activated, TurboPlayer will always put the first element in a rundown onto the first preload position, the second element to the second position and so on. Unloadable elements occupy positions too. The preload range is never changed/advanced.

This behaviour is needed if you want to address elements by fixed positions in macros. It is also useful for MultiSequenced/Once-rundowns with less elements than available positions because it minimizes the preloads/unloads.
This parameter contradicts the parameter "PreloadAroundCursor", therefore you can use only one of them.
ProtocolTrack: (Par_int, Default=1000 (show, jingles), -1 otherwise) This parameter applies for the show list, for the stacks in mode "Track" and for stacks in other modes only if at least one show is being loaded to the show rundown. This parameter specifies the number of the protocol track in which the sent elements of the corresponding rundown list are written. If the parameter is -1 nothing is written to any protocol track.
ProtocolTrackOffAir: (Par_int, Default=identical to) This parameter applies for the show list, for the stacks in mode "Track" and for stacks in other modes only if at least one show is being loaded to the show rundown. In addition to the parameter "ProtocolTrack" a second track number can be specified which applies when the On-Air mode is off. If you specify -1 no protocol is written in off-air mode. If you do not specify a value the same track as specified in "ProtocolTrack" for the on-air mode is used.
Region: (Par_string, Default=<empty>) This is an alternative for parameter "DistributionEndpoints". (Valid since: 31.10.2022)
Regions: (Par_string, Default=<empty>) This is an alternative for parameter "DistributionEndpoints". (Valid since: 31.10.2022)
RundownMode: (Par_string, Possible values=Sequenced, Once, Random, MultiSequenced, Default=Sequenced (Show), Random otherwise) The rundown mode of the list. See the technical manual of TurboPlayer, chapter "Concepts" for an explanation of the rundown modes.
- Sequenced: The rundown is played from the beginning to the end exactly in the order the elements are scheduled. Of course there are exceptions, like time- or mute starts, but essentially the next pointer is given by the system and the user cannot start a different element than the next one. When an element has been played, its send state is set to "sent" and the element cannot be played again. In case an element which is not the next one has to be started (e.g. a timed start) all unplayed elements above the started element are skipped. This mode is used for shows which must be played in the given order.
- Once: As the name says, each element can be played once. After it has been played, the send state of each element is set to "sent" and the element cannot be played again. The difference to the previous mode is that the user need not play the elements in the given order. He can set the next pointer to any element which is (still) playable. This mode can be used for shows in which the user should have the freedom to play each element when he likes to. Nevertheless the system obeys the start attributes and something like "sequenced" or "relative" starts are executed. Be warned that you can get multiple running sequenced sub-chains in this mode. (E.g. element 1 and 4 are started by the user and elements 2 and 5 are sequenced. Then you have two sub-chains running independently.) In automatic activation this rundown mode is the same as "sequenced".
- MultiSequenced (since build 802): In this mode the user can select any element he wants to start. After starting it sequenced or relative elements below are started automatically. In contrast to mode “once” played elements do not get the send state “sent”. Therefore each element can be played as many times as the user likes. Time starts are executed. Like in mode “once” you can get multiple running sequenced sub-chains. This mode should not be used with automatic activation.
- Random: Each element can be played whenever the user wants to play it. Start attributes of the elements are ignored, no sequence is played. When an element has been played its send state is reset to its original state, typically "planned", and the element can be played again. The user can drag an element to a player assigned to the rundown to preload the element in the corresponding channel. This mode is used for jingles typically.
RundownModeLocked: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=Yes (Show, Jingles), No otherwise) This parameter defines if the rundown mode can be changed during runtime by the user in the GUI. Changing per macro is always possible.
SingleElement: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0) If you set this parameter to "yes" there cannot be more than a single element in the rundown. Surplus elements will automatically be deleted. This is useful for "pure players": players which are assigned to an invisible stack rundown. As the user sees only one element on the player, the assigned list should be limited to a single element too. To avoid losing BCS elements, this parameter is only available for stacks in clipboard mode.
StackMode: (Par_string, Possible values=Clipboard, Free, Track, Endpoints, Default=Clipboard) This parameter defines the stack mode for the rundown list. See the technical manual of TurboPlayer, chapter "Concepts" for an explanation of the stack modes.

StackModeLocked: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=Yes (Show, Jingles), No otherwise) This parameter defines if the stack mode can be changed during runtime by the user in the GUI. Changing per macro is always possible.
Track: (Par_string, Default=0 for the show list, none otherwise) This parameter applies only for the show list and for the stacks in mode "Track". The parameter specifies the number of the show track which is loaded into the corresponding rundown list.

\TurboPlayer\Stinger

Lists different stinger types. (A stinger is an element which fades other running elements when it is started.)
AbsoluteStartIsStinger: (Par_string, Possible values=Yes, No, TRUE, FALSE, 1, 0, Default=No) "Absolute start" means: an element starts at a fixed scheduled time, which is independent of previous elements. Typically this is the case for "start on time", "end on time" or the first element of a backward floating chain if it starts automatically. Normally the start of such an element does not fade out any other running elements. To achieve such a fade-out, you must either make the element a stinger, or you can set this parameter to Yes/TRUE/1 and every absolute start will fade-out running elements.
BeatExactDuckingLevel: (Par_int) You have to specify this value (in dB) if you want to duck the first element in a beat-exact transition when the second element starts. This is the end level of the fade. The first element will continue to run with this level until it is faded out. You must also specify the value "BeatExactDuckingTime", otherwise no ducking will be done.
BeatExactDuckingTime: (Par_int) If you specify this time TurboPlayer will duck the first element in a beat-exact transition when the second element starts. The ducking time is given in milliseconds it is the time until the ducking level is reached.
BeatExactFadeOut: (Par_string) Here you can specify how fast the first element of a beat-exact transition is faded out. If you specify two numbers (in milliseconds, comma separated) the first one is the time before the synchronized beat at which the fade-out starts and the second number is the time after the synchronized beat at which the fade-out ends and the element is stopped. If you specify only a single number, the time will be split in 1/3 before the beat and 2/3 after the beat.
BeatExactSequenced: (Par_string) With this flag you can activate beat-exact starts for transitions in which the second element is sequenced. Normally these transitions are not started beat-exact by TurboPlayer, but of course they can be scheduled beat-exact by using CrossfadeMixer and DigAIRange. This work can be avoided and TurboPlayer can make these transitions beat-exact by moving the start of the second element accordingly. See the chapter "Concepts" of TurboPlayer-TechManual for more information.
DefaultFadeOut: (Par_int, Default=2500) This fade-out time is used, when a stinger element does not specify a specific fade-out time, but [Standard] instead. (Valid since: 01.05.2004)
StingerFadesEverything: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=TRUE) By default TurboPlayer will fade all running elements when a stinger is being started. With this parameter you can limit the stinger feature to elements from the same rundown as the started stinger. Example: if the parameter is FALSE, a started stinger from the jingle list will not fade elements in the show list. (Valid since: 01.05.2004)
StingerFadesToClosed: (Par_string, Possible values=TRUE, FALSE, Yes, No, 1, 0, Default=No) Normally a fade-out due to a stinger is treated like all fades, meaning: the fade is limited to the MinimumOpenLevel of the line. Not until 250 ms after the stop was done, the fader is moved to a closed position. In case you want to move the fader directly to the closed position, set this parameter to "yes". (This parameter applies only to move-fader commands to a mixer console with variable positions.)

\TurboPlayer\Stinger\...

-
FadeOut: (Par_int, Default=2500) The time in which running elements will be faded down to the minimum level when a stinger of this type is started. (Valid since: 01.05.2004)
Name: (Par_string) The name of the stinger as it is displayed to the user. If the name parameter is empty, the key name is used instead. (Valid since: 01.05.2004)
Program: (Par_string) A comma-separated list of services, for which the stinger type should be available. The specification of a star '*' means: it is available in all services. (Valid since: 01.05.2004)
TriggeredFrom: (Par_string, Default=<Empty>) Since version 6.0 TurboPlayer supports stingers across different TurboPlayers. In this parameter enter a comma-separated list of computer names,

which are allowed to remotely trigger the stinger. If you enter a single star "*" TurboPlayers on all computers can trigger the stinger.
Please also check the parameters "SendMessagesForRundown" and "SendBroadcastMessages" in key "TurboPlayer\Communication\RemoteTurboPlayers", because triggering stingers works with play state messages, which must be sent between the TurboPlayers. In addition, it is necessary that the sending and the receiving TurboPlayer have the same service (program) loaded. It is _not_ necessary that the receiving TurboPlayer has the stinger element of the sending TurboPlayer loaded into one of his rundowns.
For remote stingers the parameter "StingerFadesEverything" in key "TurboPlayer\Stinger" is not evaluated. Remote stingers always work across different rundowns. (Valid since: 8.7.2021)

\TurboPlayer\TimeInfo

The section defines, which types of timing information are calculated by the TP Engine.

\TurboPlayer\TimeInfo\...

Each subkey of TurboPlayer|TimeInfo defines one set of timing data, which is calculated by the TP Engine
Class: (Par_string, Possible values=[valid class name]) For a TimeInfo item with Type=RemainClass, this parameter defines the class, to which this TimeInfo applies. Valid values are the DigaSystem class names, e.g. Audio, Music, News, Text, Commercial, etc.
ConsiderAudioTextInGroupDuration: (Par_string, Possible values=0, 1, Default=0) Only valid, when Type is "RemainGroup".
If the parameter is 1, the calculation of the remaining time in the current group includes the duration of the moderation text for audio elements.
Content: (Par_string, Possible values=Audio, Total, Default=Total) Only valid, when Type is "DurationSum".
Defines, which content of an element is included in the time calculation. Possible values:
- Audio: only the length of the audio
- Total: length of audio + length of text
DurationMode: (Par_string, Possible values=Link, Mark, Default=Mark) Only valid, when Type is "DurationSum".
Defines, how the audio length of each element is calculated. Possible values:
- Link: Time between link-in and link-out
- Mark: Time between mark-in and mark-out
End: (Par_string, Possible values=NextFixed, Show, Default=Show) Only valid, when Type is "DurationSum".
Defines, up to which element the time calculation is made. Possible values:
- Show: until the end of the show
- NextFixed: up to, but not including, the next fixed-start element in the show. If there is none, the end of the show is used anyway.
OnlyShowendIsFixpoint: (Par_string, Possible values=0, 1, Default=0) Only valid, when Type is "NextFixed" or "RemainNextFixed".
If 1, only the end of the current show is regarded as a fix point. Therefore, even if there is a fixed-start element upcoming in the show, the calculated time is the end of the show.
ReverseSign: (Par_string, Possible values=0, 1, Default=0) Only valid, when Type is "RemainNextFixed" or "GapOverlap".

For "RemainNextFixed":
Normally, the remaining time until a future fixed-start element is calculated as a positive time interval. Only if the next fixed-start element is in the past (a situation, which shouldn't happen in most situations), is the value prefixed with a "-" sign.
With ReverseSign=1, this behavior can be reversed. I.e., time intervals to upcoming fixed-start elements are calculated as negative values.

For "GapOverlap":
Normally, gaps have positive sign, and overlaps negative sign. With ReverseSign=1, this can be reversed.
Rundown: (Par_string, Possible values=[see description]) The rundown, for which the time information is calculated. Possible values are "Show", "Jingles", "Stack1", "Stack2", ..., "Stack15"

Scope: (Par_string, Possible values=All, Remain, Default=All) Only valid, when Type is "DurationSum". Defines, which elements are included in the time calculation. Possible values:
- All: Everything from the beginning of the show
- Remain: Only the remaining elements, which have not yet been played

ShowendIsFixpoint: (Par_string, Possible values=0, 1, Default=1) Only valid, when Type is "NextFixed" or "RemainNextFixed".
If 1, the end of the current show is also regarded as a fix point. Therefore, if no fixed-start element is found, the calculated time is the end of the show.
Type: (Par_string, Possible values=[see description]) The type of the calculated time information. Possible values are:
- "NextFixed": Starting time of the next fixed item
- "RemainNextFixed": Remaining time until the start of the next fixed item
- "DurationSum": Sum of durations of certain elements; the exact scope is defined by further parameters
- “GapOverlap”: Total length of gaps or overlaps in the current show
- “Crossover”: Countdown of time between Outro of one element and the Intro of the following one
- “RemainGroup”: Remaining time in current group
- “RemainClass”: Remaining time until an element of a specific class (parameter “Class”)

UseFixedGroupDuration: (Par_string, Possible values=0, 1, Default=0) Only valid, when Type is "GapOverlap".
If the parameter is 1, the gap/overlap calculation uses the predefined length of fixed-duration groups. Otherwise, the combined length of the group's elements is used.

\TurboPlayer\Volume

This section contains volume parameters for the recorder used within TurboPlayer. For a description of the parameters see the global "Volume" section.

8 LOGGING

8.1 Logfiles

The following DigaSystem log destinations are used by TurboPlayer:

BCE-Dump: (Par_string) A protocol of TurboPlayer: the destination for the "dump engine state" function. The dump function can be called to store the internal state (loaded elements, line and channel state, variables, etc) for an evaluation.
BCE-Error: (Par_string) A protocol of TurboPlayer: the error log of the engine (Broadcast Control Engine). If you activate this log, you should direct it into the same file as the normal technical log "BCE-TechLog" to have a better overview over the context.
BCE-IO-(module name): (Par_string) A protocol of TurboPlayer: the destination for logs of the IO module (module name). Errors of the module are written in both protocols, this one and the BCE-Error protocol. But this protocol can contain additional output of the module.
BCE-Messages: (Par_string) A protocol of TurboPlayer: contains a log of thread messages transferred between internal parts. This log is for debugging purposes only. If you activate it, you must also select the messages IDs of interest in the parameter "TurboPlayer\Logging\MessageIDs". This is necessary because you will get thousands of lines per second if you do not limit the message types to be written.
BCE-PlayLog: (Par_string) A protocol of TurboPlayer: This is an extra protocol, containing only information about start and stop of elements. The same information is included in the technical log already, but here the information is limited, to make it easier to evaluate, which elements were played and when they were played.
BCE-TechLog: (Par_string) A protocol of TurboPlayer: the technical log of the engine (Broadcast Control Engine)
BCE-TechLogEx: (Par_string) A protocol of TurboPlayer: supplementation to the technical log of the engine (Broadcast Control Engine). If you activate this log, you should direct it into the same file as the normal technical log "BCE-TechLog", because it only contains extended information to the technical log.

BCE-TreeOp: (Par_string) A protocol of TurboPlayer: This is an extra protocol, containing only information about BCS tree operations (inserts, updates, deletes of elements or higher nodes).
StandaloneCFM: (Par_string) This is the general logfile of the StandaloneCFM.exe, a helper application which allows to isolate MultiRec and CrossfadeMixer from TurboPlayer to increase system stability.
StandaloneCFM-Messages: (Par_string) This is an additional logfile of the StandaloneCFM.exe, to which sent and received messages can be logged. You have to use the parameter "StandaloneCFM\Logging\MessagesIDs" to specify the message IDs which should be included in the log.
TurboPlayer: (Par_string) A protocol of TurboPlayer: Despite the name it contains only the log of the GUI. Other logs can be found in the parameters "BCE-..." (BCE=Broadcast Control Engine)

The recommendation is to write the three destinations “BCE-TechLog”, “BCE-TechLogEx” and “BCE-Error” in the same file. All other destinations should be written to individual files.

8.2 Message log

8.2.1 TurboPlayer Message IDs

The internal parts of TurboPlayer use a messaging system for the communication. In the protocol destination "BCE-Messages" you can log each message which is being sent. This can mean some thousand messages per second. Therefore, it is possible to limit the logged messages with the parameter "TurboPlayer\Logging\MessageIDs". It is a comma-separated list of the message IDs which should be written. Below is a list with all known message types together with a short description. Of course, this protocol is only helpful if you are examining problems and you should adjust the written IDs accordingly.

ID	Message	Comment
General Messages:		
10000	ErrorMessage	Signals an error
10001	RequestDumpMessage	Sent by the GUI to request a dump from all engine modules
10002	ResetMessage	Performs a reset
10003	ChangeModeMessage	Changes one of the engine states: Pause mode, Play fadings mode, Fader start, Wrong way driver pause mode, Activation, Stack mode
10004	FullInfoMessage	Sent by the GUI to request full information about states, loaded trees and elements form the engine
10005	NextPlayItemMessage	Sent by the kernel to signal a change of the „next element“
10006	PlayStateMessage	Sent from kernel to GUIs to signal a play state change
10007	PlayTimeMessage	Regularly (once per second) sent from audio engine to kernel and from there to the GUIs or to IO modules to inform about play and remain time of playing elements
10008	ConnectionStateMessage	This message is used to signal established, disturbed or closed connections, e.g. to the BCS or to remote TurboPlayers.
10009	ActiveShowMessage	Message from tree manager to GUIs to inform about a change in the active show (the one with the last started on-air element)
10010	TreeOperationMessage	Sent to perform a tree operation
10011	GenericEventMessage	Sent between the application parts to perform or signal a generic out or in event
10012	GenericKeyMessage	Sent between the application parts to signal a pressed generic key (not used so far)
10013	OnAirRequestMessage	Sent between server proxy, tree manager and GUI to handle the request of another TurboPlayer to become active.
10014	SecondInitMessage	Used to perform a delayed initialization after startup or reset
10015	InternalInfoMessage	Used to send some general information (for macros) to the kernel.
10016	TerminateMessage	Used by IOManager to signal a program termination.
10017	LoadShowMessage	Used between TurboPlayers to perform slaved show loading.
10018	DelinkMessage	Used between engine and GUI for de-linking a remote show or track
10019	SetCursorItemMessage	Sent from GUIs to kernel to set the cursor to an element.
10020	PrepareChannelMessage	Sent between Kernel / TreeManager and GUIs to prepare an element to a specific channel. (Hint: till build 501 this message had the ID 50001)
10021	SubChannelFadeMessage	Sent between Kernel / audio engine for sub-channel fades (=CartMotion mixes in multichannel audio).
10022	StartAppMessage	Sent from Kernel to AppStarter subsystem to execute a TP_StartApp command.
10023	StopGroupMessage	Sent from TreeManager to Kernel and/or Kernel to GUI to signal the stop of a running group.
10024	CallRESTServiceMessage	Sent from Kernel to AppStarter subsystem to call a REST service from a TP_CallREST() macro command.
10025	ElemAssocPlayInfoMessage	Sent by TreeManager to the kernel and the GUIs to inform about play state changes or play time of element associations (groups, stories, transitions; only transitions are implemented so far).
10026	CustomGUIMessage	Sent by kernel to all GUIs which were specified in the macro command TP_SendCustomGUIMessage.

10027	QueryInfoMessage	Use by GUIs to query information from the engine. Currently only used to query the current (registry) config name from TreeManager.
10028	PlayInfoMessage	This message is sent between RundownKernel and ServerProxy within TreeManager in order to send information about play states and play times to/from BCS so that other BCS clients get this info.
10029	RecordStateMessage	Sent from IOManager to RundownKernel and then further to the GUIs in order to inform the kernel/the GUIs about starting/stopping/pausing live recordings of ROAD.
10030	RecordProgressMessage	Like before, but this is an extra message being continuously sent while a ROAD live recording is running. It contains the recording time and level information.
10031	LoadStateMessage	This ID is used since version 6.1 of TurboPlayer. Older versions used the ID 50000. The message is sent from kernel and informs the TreeManager, the GUI and connected remote TurboPlayers about load state changes of elements (loaded, load failed, unloaded).
TreeManager Messages:		
20001	ServerConnectMessage	Sent from GUI to tree manager to perform a BCS connect + login
20002	ProgramListMessage	Used by the GUI to request a list of known programs (=services)
20003	SetProgramMessage	Used by the GUI to set a specific program (=service)
20004	ShowListMessage	Used by the GUI to request a list of known shows
20005	GetShowMessage	Used by the GUI to request information about a specific show
20006	SelectTreeMessage	Used by the GUI or within the tree manager to load a new part of the BCS tree
20007	GuiNotificationMessage	Informs the GUI about changes in the loaded BCS trees
20008	SyncGetNodeMessage	Used by the GUI to request information about a specific node
20009	CheckShowListMessage	Internally sent by the tree manager timer to check the list of loaded shows
20010	(TreeOperationMessage)	Old value, no longer used, now -> 10010
20012	FetchAllJinglesMessage	Used in tree manager internally to fill the cache of all jingle groups
20013	AdjustTimesMessage	Sent by GUI to re-calculate the start/stop times while no element is playing
20014	SetContentMessage	Sent by GUI to set the full content of a rundown list
20015	LockNodeMessage	Old message, deprecated, replaced by 20016.
20016	LockNodeMessage	Used to transfer requests/info for/about locking of BCS nodes
20017	LoudnessParameterMessage	Used to transfer loudness settings from TreeManager to the GUIs.
20019	GeneratePlayInfoMessage	Used in TreeManager internally for a timer to regularly generate time information about element associations for the GUIs.
20020	TimedPollContentMessage	Used in TreeManager if polling for BCS content has been activated.
20021	LoadedShowsListMessage	Used by the GUI to request a list of the currently loaded shows.
20022	TimeInfoMessage	Used to send information from the time info windows of a GUI to the engine and from there to TPService/WebTP.
20023	RequestTimeInfoNotificationsMessage	WebTP can request specific time info notifications with this message, sent via TPService.
20024	SendTimeInfoToGUIsMessage	Sending time info messages to TPService/WebTP is done on a regular basis with a timer. This message is used within TreeManager by this timer.
20025	CallServerProxyWatchdogMessage	Used internally in TreeManager to regulary trigger the watchdog for the worker thread of the ServerProxy.

ServerMessages		
30000	ChangeNotificationMessage	Used in tree manager to start or stop BCS notifications
30001	ServerNotificationMessage	Informs the tree manager about changes in the loaded BCS trees
30002	GetNodeMessage	Used by tree manager to load a specific node from the BCS
30003	FindShowMessage	Used by tree manager to query a show with a specific offset
30004	CheckConnectionMessage	Used by tree manager timer to check the BCS connection regularly
30005	PerformDelayedOpMessage	Used in tree manager to execute tree operations which had to be delayed due to a BCS connection loss
30006	InsertNodeMessage	Used in tree manager to execute an insert tree operation
30007	MoveNodeMessage	Used in tree manager to execute a move or copy tree operation
30008	UpdateNodeMessage	Used in tree manager to execute an update tree operation
30009	DeleteNodeMessage	Used in tree manager to execute a delete tree operation
30010	ChangeNodeIDMessage	Used in tree manager to correct the temporarily used node IDs during a connection loss by the real IDs received from the BCS
30011	ResyncNodesMessage	Used in tree manager, initiates a tree sync after a BCS reconnect
30012	FutureShowlistMessage	Used in tree manager to request a list of the next n future shows from BCS
30013	FreeShowlistInfo	Contains a list with assignments of a show ID and the show start time. Used between TreeManager and server proxy.

KernelTreeMessages		
40000	ChangeRundownMessage	Sent from tree manager to kernel to change a rundown
40001	QueryPrelistenElementMessage	Sent from kernel to tree manager to query data for an element which must be prelistened but is not known to the kernel
40002	ProtocolTrackMessage	Sent from kernel to tree manager to make an entry in the protocol due to a start or stop. Besides it is used to skip elements above a started one
40003	ChangeElementMessage	Sent from kernel to tree manager to change the data of a single element
40004	PlayingRemoteMessage	Timer message between tree manager and rundown kernel for surveying running remote elements
40005	InsertFillerMessage	Sent from kernel to tree manager to request filling a gap with filler jingles
40006	TreeNotificationMessage	Sent from tree manager to kernel to update the XML data of the kernel (only if the global flag <KernelHasFullData> is true.)
40007	ResetTransitionMessage	Sent from kernel to tree manager to trigger the execution of the macro command TP_ResetTransition.
40008	CheckOverlaidFlagsMessage	Sent from kernel to TreeManager if a check and a potential automatic assignment of the overlaid flag for playing elements must be done.

KernelGuiMessages		
50000	LoadStateMessage (deprecated)	This ID was used up to version 6.0. Since version 6.1 the ID 10031 is used.
50002	PrelistenStateMessage	Sent between kernel and GUIs to select an item for prelistening or to signal a change in the current prelisten item / state
50003	PlayFadingsStateMessage	Sent between kernel and GUIs to change the <play fadings> state of an element
50004	HealthStatusMessage	Used if a GUI queries the current health status XML from the kernel.
50005	GuiRecordingMessage	Sent to a GUI in case of a TP_ControlGUIRecording command.
50006	RequestRemotePlayInfoMessage	Sent from a GUI to kernel to request play info messages for remotely playing elements.

KernelAudioMessages		
60000	LoadMessage	Sent between kernel and audio engine to load/unload an item or to signal a load state change
60001	InitChannelMessage	Sent between kernel and audio engine to prepare / unprepare / queue / unqueue an item or to signal such an event

60002	PlayMessage	Sent between kernel and audio engine to play / stop / pause / fade an item or to signal such an event
60003	PreloadTimerMessage	Regularly sent by the kernel timer to initiate loading and unloading of elements
60004	CloseLineMessage	Sent by the kernel timer to close a line, if necessary
60005	VirtualPlayTimerMessage	Used inside of kernel by the virtual play engine for external elements
60006	VirtualDurationMessage	Used insides of kernel to correct the duration of external and live elements
60007	ReportFileIDsMessage	Used within kernel to regularly synchronize the known file IDs of all preloaded elements with MultiPlayer.
60008	RemotePlayTimerMessage	Messages used by an internal timer within RundownKernel in order to handle the play time info of remote TurboPlayers and to check, whether play time info for the own GUIs must be sent.
IOMessages		
70000	IOInstanceInfoMessage	Sent from IO manager to kernel to report the list of loaded IO modules
70001	SelectSourceMessage	Sent from kernel to IO manager to select a mixer source
70002	MoveFaderMessage	Sent between kernel and IO manager to perform / signal a move fader
70003	PlayStateIOMessage	Sent between kernel and IO manager to perform / signal a play state change
70004	SetGainMessage	Sent from kernel to IO manager to set the input gain for a line
70005	SetSignalProcessingMessage	Sent from kernel to IO manager to select a specific audio processing for a line
70006	SetDisplayMessage	Sent from kernel to IO manager to display a text on a mixer button (not used so far)
70009	SetNextSourceMessage	Sent from the IO module to set the mixer source for the external element of the current cursor element
70010	PFLMessage	Sent to/from IO module to activate PFL for a line
70011	InEventMessage	Sent from IO module to kernel to inform about in-events
70012	ControlRecordingMessage	Sent from kernel to IO module to control recordings
70013	IOChangeModeMessage	Not used anymore.
KernelMessages		
80000	PrepareMessage	Used inside of kernel to prepare an element
80001	CheckStateMessage	Used inside of kernel to regularly check the current states
80002	InformStateMessage	Used inside of kernel to delay a load state/play state information to the GUIs
80003	KernelPlayStateMessage	Used inside of kernel to delay the start/stop of an element
80005	ExternalStartMessage	Used inside of kernel to delay the treatment of unexpectedly started external elements until they have been inserted in BCS.
80006	ExecuteEventMessage	Used inside of kernel for a timed execution of controls/out-events
80007	ExecutePeriodicTimerMessage	Used inside of kernel for periodic timers to start the execution
80008	ComputeNextElementMessage	Used inside of kernel for a delayed next-element computation
80009	ExitMessage	Used inside of kernel to execute internal events during termination
80010	CheckReserveElementMessage	Used inside of kernel to delay the treatment of reserve elements.
StandaloneCFM-Messages		
90001	ResultMessage	Used to report a result (success or error) from StandaloneCFM to TurboPlayer.
90002	InitMessage	Used for the initialization of the components with StandaloneCFM (MultiRecs, CrossfadeMixer).
90003	PropertyMessages	Used to set or query properties of one of the MultiRecs or of CrossfadeMixer.
90004	FunctionCallMessage	Used to call a function of one of the MultiRecs or of CrossfadeMixer, e.g. Reset, Load, Save,…
90005	EasyMessage	Used to call one of the functions of the EasyPlayer interface of one of the MultiRecs, e.g. Play, Stop, …
90006	EventMessage	Used to send an event from StandaloneCFM to TurboPlayer, e.g. Error, ButtonClicked, Modified, …

Head Office:

Erika-Mann-Str. 67
80636 Munich - Germany

Phone +49 89 540 139 0
Fax +49 89 540 139 50

office@davidsystems.com

Authors:	Markus Hengstebeck Tino Prosche Alexandra Bauer Christian Bauer Carlos Oliveira Sujata Sharma Andreas Parsch
File name:	TurboPlayerTechManual.doc
Last revision:	27. August 2024