DAVID systems

DigaSystem.
The Radio Operating System

Broadcast System Technical Manual

Technical Documentation

DIGITAL AUDIO VIDEO INTEGRATION DEVELOPMENT

CONTENT

1 System Overview ... 9

1.1 Module Overview ... 9

1.2 List of Files ... 11
1.2.1 Program files ... 11
1.2.2 Libraries ... 11
1.2.3 Miscellaneous Files ... 16
1.2.4 Additional runtime files ... 16
1.2.5 Customizable image files ... 16

1.3 General Installation Hints ... 17
1.3.1 System requirements ... 17

1.4 Known limitations and incompatibilities ... 18
1.4.1 IPv6 support ... 18
1.4.2 TLS support ... 18
1.4.3 64-bit modules ... 19
1.4.4 BCS and many clients ... 19
1.4.5 BCS and 24 h shows / shows across midnight ... 20
1.4.6 BUS FileCheck and audio compiled by CrossfadeMixer ... 20
1.4.7 DigaMOS and BCS 64 Bit ... 20
1.4.8 DigaMOS and BCS locks ... 21
1.4.9 DigaMOS and web modules ... 21
1.4.10 OnAIR TrackMixer and CrossfadeMixer features ... 21
1.4.11 DigAIRange and remote links ... 22
1.4.12 Undo handling and multiple deletes ... 22
1.4.13 Startup dialogs and invisible message boxes ... 22
1.4.14 TurboPlayer and relative starts ... 23

1.5 Right Objects ... 24
1.5.1 Objects ... 24
1.5.2 Rights ... 25
1.5.3 Access control lists ... 26
1.5.4 Administration rights ... 26

2 XML Data Structure ... 27

2.1 Overview ... 27

2.2 Description of the different nodes ... 29
2.2.1 Metadata of Nodes ... 29
2.2.2 Parts of Metadata ... 38
2.2.3 Fade list ... 48
2.2.4 Beat marker information ... 49
2.2.5 Class and MediaType ... 50

2.3 Minimum required fields ... 51

3 File Management ... 53

3.1 Introduction ... 53

3.2 Details ... 53
3.2.1 Media directories ... 53
3.2.2 Additional show media directory ... 53
3.2.3 Media directory profiles ... 53
3.2.4 Filenames ... 54
3.2.5 FileState ... 54

3.2.6 File reading ... 55
3.2.7 OnAir files ... 55
3.2.8 File generation ... 55
3.2.9 File writing ... 56
3.2.10 Remote shows / tracks ... 57
3.2.11 Deleting files ... 57

4 Start attributes ... 58

4.1 Introduction ... 58

4.2 Details ... 58
4.2.1 StartType ... 58
4.2.2 StartMode ... 59
4.2.3 Other fields and parameters ... 61
4.2.4 Usage of overlaid flag ... 63

5 Import / Export ... 65

5.1 The Import Function in DigAIRange and BUS ... 65
5.1.1 Manual, semi-automatic and full-automatic import ... 65
5.1.2 Merging modes ... 66
5.1.3 Detailed explanation of the merging modes based on a working class ... 67
5.1.4 Stories ... 69
5.1.5 EndpointStories ... 69
5.1.6 Show structure alignment ... 69
5.1.7 Import DLLs ... 70
5.1.8 Improvement DLL ... 70
5.1.9 Improvement search expressions ... 71
5.1.10 Emergency filename generation ... 72
5.1.11 Import options ... 73
5.1.12 Import protection matrix ... 79

5.2 Import DLL specific Settings ... 80
5.2.1 DirectImport.Dll ... 80
5.2.2 MuscImport.Dll ... 83
5.2.3 FlexIE.Dll ... 84
5.2.4 AdComImport.Dll ... 84
5.2.5 SelectorImport.Dll ... 85
5.2.6 SelectorStdImport.Dll ... 85
5.2.7 SepIImport.Dll ... 88
5.2.8 NewsScheduleXmlIE.dll ... 88
5.2.9 DirectBcsImport.dll ... 89

5.3 Export Dll specific Settings ... 91
5.3.1 DirectExport.Dll ... 91
5.3.2 XSLExport.Dll ... 92
5.3.3 FullExport.Dll ... 92
5.3.4 EasyXMLExport.Dll ... 93
5.3.5 FlexIE.Dll ... 94
5.3.6 FlexXML.Dll ... 94
5.3.7 SUISAExport.dll ... 95

6 Broadcast Server ... 97

6.1 General information ... 97

6.2 Installation ... 98

6.3 Command line options ... 99

6.4 64-bit BCS and WebSocket communication ... 101
6.4.1 WebSockets ... 101

6.4.2 WebSocket message format ... 101
6.4.3 64-bit BCS / Advanced tree lock ... 102
6.4.4 Upgrading an installation ... 102
6.4.5 Configuring Secure WebSockets via TLS ... 103
6.4.6 BCS and OpenID Connect / JSON Web Tokens ... 104
6.4.7 BCS and DPE Tokens ... 105

6.5 Logging ... 106
6.5.1 Version smaller than 5.4 ... 106
6.5.2 Version 5.4 or higher ... 106

6.6 Remote GUI ... 107
6.6.1 BCSGUI.EXE ... 107
6.6.2 BCSGUI in DigAIRange ... 108

6.7 Settings ... 109
6.7.1 Parameter description ... 109
6.7.2 Multiple BCS Instances on One Computer ... 119

6.8 Operation ... 119

6.9 Buddy Server ... 119
6.9.1 Setup ... 119
6.9.2 Limitations ... 120

6.10 Plug-In DLL ... 121
6.10.1 Standard Plug-In DLL ... 121
6.10.1.1 Assignment of Default Values ... 121
6.10.1.2 Adjustment of Deletion Date in Database Entries ... 121

6.11 Error-Codes ... 122

7 Broadcast Utility Server ... 126

7.1 Bus in General ... 126
7.1.1 Introduction ... 126
7.1.2 Requirements ... 127
7.1.3 64-bit BUS ... 127
7.1.4 Installation ... 127
7.1.5 Command line options ... 128
7.1.6 Program settings ... 130
7.1.7 Special test commands ... 131
7.1.8 Remote GUIs ... 132
7.1.8.1 DigAIRange ... 132
7.1.8.2 BUSGUI ... 134
7.1.9 The Task Concept ... 136
7.1.10 Macros and filter expressions ... 137

7.2 Copy/Convert Task ... 143

7.3 File Check Task ... 144

7.4 Import Task ... 151
7.4.1 Using Import Task for duplicating shows to a remote BCS system ... 154

7.5 Export Task ... 157

7.6 The Insert Tasks ... 159
7.6.1 Pool-Insert task ... 159
7.6.2 Show-Insert task ... 160

7.7 Auto-Replace Task ... 162
7.7.1 Settings ... 162
7.7.2 Variables and Wildcards ... 165

7.8 Counter Task ... 166
7.9 Delete Task ... 168
7.10 Align Task ... 170
7.11 Stress Test Task ... 171
7.12 Day Structure Import ... 173
7.13 Rebroadcast ... 175
7.14 Metadata scheduler ... 177
7.15 Synchronize ... 179
7.16 Consistency Check ... 184
7.16.1 Introduction ... 184
7.16.2 How it is used (in short) ... 184
7.16.3 Components ... 184
7.16.3.1 Checksets ... 185
7.16.3.2 Start triggers ... 185
7.16.3.3 Consistency checks ... 185
7.16.3.4 Reporting to a central XML file ... 185
7.16.3.5 Reporting back to BCS ... 186
7.16.4 Configuration of the main ConsistencyCheck task dialog ... 186
7.16.5 Configuration of the ConsistencyCheck general settings ... 190
7.16.6 Configuration of the individual ConsistencyCheck dialogs ... 190
7.16.6.1 Files ... 190
7.16.6.2 Gaps of elements ... 191
7.16.6.3 Overlaps of elements ... 193
7.16.6.4 Over- / under- filled groups with a fixed / planned duration ... 194
7.16.6.5 Show limits ... 195
7.16.6.6 TurboPlayer stop warning ... 196
7.16.6.7 TurboPlayer load failure ... 198
7.16.6.8 TurboPlayer start failure ... 199
7.16.6.9 Media / take duration ... 200
7.16.6.10 Loudness ... 201
7.16.6.11 Playout channels ... 202
7.16.6.12 Misc ... 203
7.16.6.13 Generic metadata ... 204
7.17 Consistency XML Writer Task ... 206
7.18 Radio AD-Server Task ... 208
7.19 Index task ... 211
7.19.1 Index task stages ... 211
7.19.2 Configuration ... 212
7.19.2.1 Communication module ... 213
7.19.2.2 Seamless re-creation of an index ... 215
8 DigAIRange ... 216
8.1 Command line arguments (case insensitive) ... 216
8.2 Configurations ... 216
8.3 Server Settings ... 217
8.4 User login ... 220
8.4.1 DigaSystem login ... 220
8.4.2 Login to BCS and BUS ... 220
8.5 Manage Dialog ... 222

8.5.1 Manage Services ... 222
8.5.2 Manage Day Templates ... 224
8.5.3 Manage Shows, Show Templates and Day Templates ... 225
8.5.4 Manage tracks ... 227
8.5.5 Team dialog ... 228
8.5.6 Access control list (ACL) ... 229
8.5.7 Manage other node types ... 230
8.5.8 Manage Remote Shows ... 230

8.6 Other administration dialogs ... 231
8.6.1 Manage Import Modules ... 231
8.6.2 Manage Export Modules ... 233
8.6.3 Day status ... 234
8.6.4 Synchronize structure ... 235

8.7 Special Settings ... 236
8.7.1 Service specific parameters ... 236
8.7.2 YearInterval ... 236
8.7.3 Full-text search module (BCSSearch.ocx) ... 236
8.7.4 Print Playlist via Browser ... 236
8.7.5 Audio Editors and DigAIRrange ... 236
8.7.6 Events and Channels ... 237

8.8 User masks ... 237
8.8.1 Step 1: requirements ... 237
8.8.2 Step 2: create DigaSystem Registry settings for a DigAIRange Custom Content Tab ... 238
8.8.3 Step 3: configure the visible Content Tabs for a user ... 238

8.9 Print templates ... 239
8.9.1 XSL templates ... 239
8.9.2 RTF templates ... 239

8.10 Time fields in DigAIRange ... 241
8.10.1 Total duration ... 241
8.10.2 Link- and mark-values ... 242
8.10.3 Computation info ... 242

8.11 Consistency check view ... 245

8.12 UI Schemes / Themes ... 247
8.12.1 UI Schemes in the registry ... 247
8.12.2 Available / default UI Schemes ... 247
8.12.3 UI Scheme values ... 248
8.12.3.1 Colors for views, dialogs and grids ... 248
8.12.3.2 Text colors ... 250
8.12.3.3 Menu colors ... 251
8.12.3.4 Button colors ... 251
8.12.3.5 Scrollbar colors ... 253
8.12.3.6 Other colors ... 253
8.12.3.7 Toolbar symbol colors ... 255
8.12.3.8 Class colors ... 257

9 Full-text search within BroadcastSystem ... 258

9.1 Overview / Modules ... 258
9.1.1 BUS.exe / BUS64.exe ... 258
9.1.2 ElasticIndex.dll / ElasticIndex64.dll ... 258
9.1.3 BCSSearch.ocx ... 258
9.1.4 BCSSearchService.exe ... 258

9.2 Installation ... 259
9.2.1 Starting with ElasticSearch ... 259
9.2.2 ElasticIndex.dll ... 260

9.2.3 BCSSearch.ocx ... 260

9.3 Search Expressions ... 262
9.3.1 Words ... 262
9.3.2 Phrases ... 262
9.3.3 Field names ... 262
9.3.4 Wildcards (* and ?) ... 262
9.3.5 Fuzzy search ... 263
9.3.6 Boolean and logical operators ... 263
9.3.7 Regular expressions ... 263
9.3.8 Ranges ... 264
9.3.9 Proximity searches ... 264
9.3.10 Reserved characters ... 264

10 Reusing broadcast content ... 265

10.1 What is re-use ? ... 265

10.2 Re-broadcast task of BUS ... 265

10.3 Recording of live elements ... 267

10.4 Re-Use workflow in DigAIRange ... 267

10.5 Metadata track ... 270
10.5.1 Subclips ... 270
10.5.2 Metadata scheduler ... 271
10.5.3 Export ... 273

11 Regionalization and Remoting ... 274

11.1 Introduction ... 274

11.2 Data handling ... 274
11.2.1 Shows vs. Tracks ... 274
11.2.2 Remote links ... 274
11.2.3 Relative starts ... 276

11.3 File handling ... 277
11.3.1 BUS ... 277
11.3.2 Data translation ... 278

11.4 Regionalization with EndpointStories ... 279
11.4.1 Introduction ... 279
11.4.2 DatabaseManager / ContentManager ... 279
11.4.3 BroadcastServer, SyncStart and neutral elements ... 279
11.4.4 Import / DigaSQL.dll ... 280
11.4.5 DigAIRange ... 280
11.4.6 TurboPlayer ... 280

12 Logs ... 281

12.1 Basics / DigaSQL logging ... 281

12.2 BCS Logs ... 282

12.3 BUS Logs ... 283

12.4 DigAIRange Logs ... 284

12.5 TurboPlayer Logs ... 285

12.6 BCSSearch and ElasticIndex ... 286

12.7 MultiRec and OnAirTrackMixer ... 287
12.7.1 In DigAIRange ... 287

12.7.2 In TurboPlayer 287

12.8 DigaSQL Logs 288

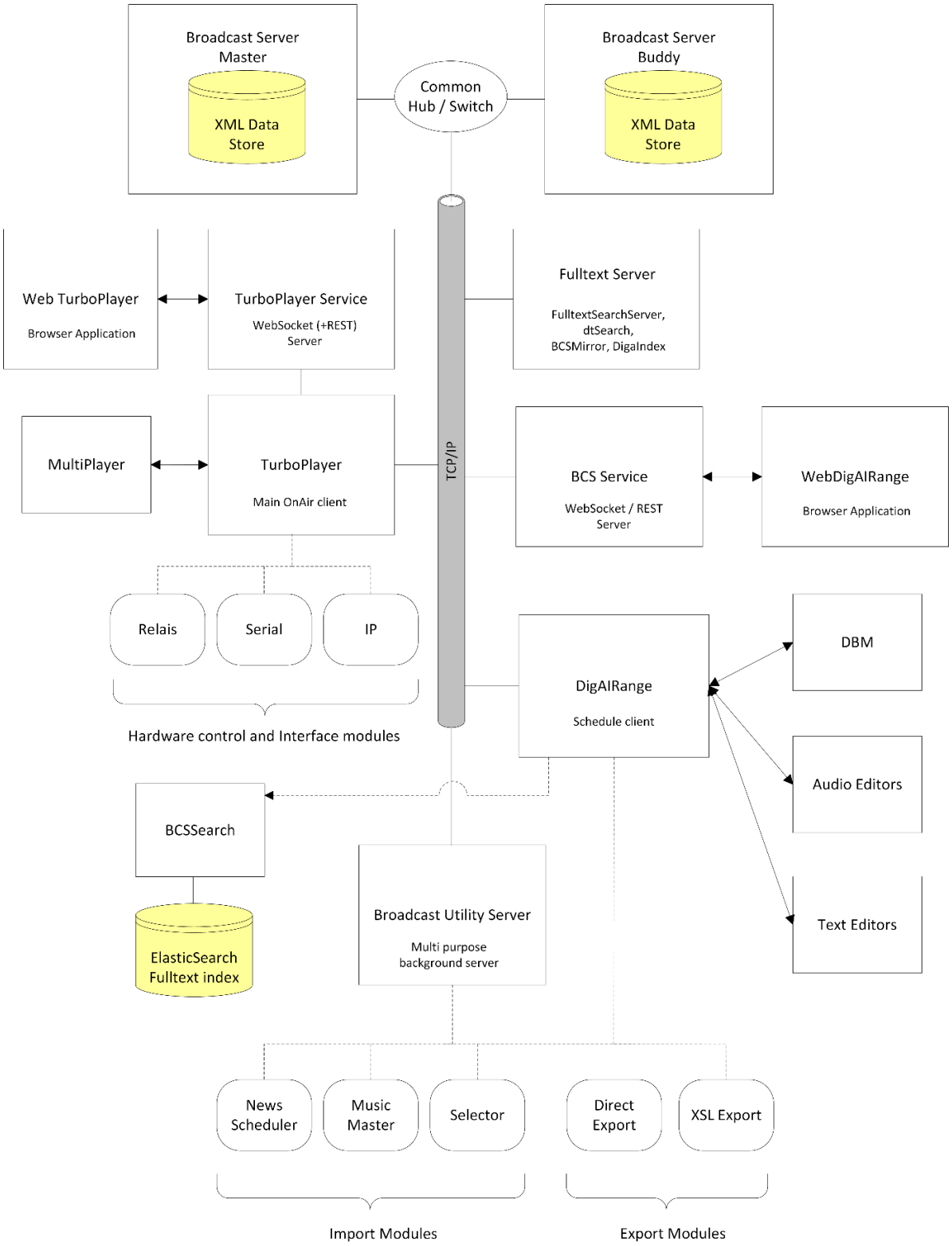
12.9 Log examples 290
12.9.1 BCS client update - log level 1 290
12.9.2 BCS client update - log level 5 291
12.9.3 TurboPlayer – playing a sequenced element 293

12.10 Process (“Mini”) Dump Files 295
12.10.1 MiniDump files via Windows Error Reporting (WER) 295
12.10.2 MiniDump files via ProcDump 296
12.10.3 MiniDump files via TaskManager 297

12.11 Data to be provided in error reports 297

1 System Overview

1.1 Module Overview



The **Broadcast Server** (BCS) is used for storage of all schedule related information. It is also responsible for the gathering and coordination of all data from any active client. This structure enables a real multi-user mode and guarantees high integrity of data. The communication between server and active clients is event-driven and based on TCP/IP. All clients are immediately informed about relevant modifications in their area. All data manipulation by clients is immediately transmitted to the Broadcast Server and from there to all active clients working on this data.
All data is stored in an XML based tree structure, which is suited to reflect a scheduling structure, and enables easy modification to future developments.
DigAIRange (the schedule client) allows the visualization and manipulation of all information stored in the Broadcast Server. Entries from other delivery modules (e.g. the DigaSystem Database Manager) can be imported by drag & drop, or new entries can be created within the scheduling client. DigAIRange makes use of the DigaSystem audio and text editors for changing elements. Besides it can be used from the supervisor for many administration tasks.
TurboPlayer (the OnAir client) is used for the distribution of a program schedule. It is the interactive interface between user and system, which visually displays the progress of the show and manages the processes necessary for the show. All relevant data is stored locally to guarantee reliability, as well as reflected in the connected Broadcast Server, enabling quick-change modifications to the show. TurboPlayer uses modules for controlling external devices ro receiving external events.
The **Broadcast Utility Server** (BUS) handles background tasks like automatic import/export or file checking/deleting. It is implemented as a separate program to keep the Broadcast Server simpler and thus more stable. Both, BUS and DigAIRange make use of the import and export modules which are responsible for converting between the XML format internally used and the format of any other system.
The ActiveX controls **CrossfadeMixer** (deprecated) and **OnAIR TrackMixer** are integrated into DigAIRange and TurboPlayer. They are graphical tools for editing crossfades and preparing transitions.
The programs **BCSService** and **TurboPlayerService** run as extra Windows services and provide a WebSocket (in parts also REST) interface to BCS and TurboPlayer. This is used by **WebDigAIRange** and **WebTurboPlayer** so far.
If a full-text search across the schedules is desired, this is possible with the help of the external software “ElasticSearch”. This search engine is accessed by BUS and ElasticIndex.dll for indexing the content and by BCSSearch.ocx and ElasticIndex.dll for searching. So far, these modules for searching can only be used within DigAIRange.

1.2 List of Files

1.2.1 Program files

Program	Directory	Description
Admin.exe	Digas	The Administrator tool. Needed for editing the DigaSystem registry and for editing rights
Admin64.exe	Digas64	64-bit variant of Admin.exe
BCS.EXE	Digas	The Broadcast Server
BCS64.EXE	Digas64	64-bit variant of Broadcast Server
BCSGUI.EXE	Digas	A (remote) GUI for BCS.
BCSGUI64.EXE	Digas64	64-bit variant of BCSGUI.
BcsSearchService.exe	Digas	A .NET module running as Windows service. It provides a REST / WebSocket interface for running fulltext queries on Elasticsearch. It is only needed by Web clients, not by desktop modules like BCSSearch.ocx.
BcsService.exe	Digas	A .NET module running as Windows service. It provides a REST / WebSocket interface to BCS.
BUS.EXE	Digas	The Broadcast Utility Server
BUS64.EXE	Digas64	64-bit variant of Broadcast Utility Server
BUSGUI.EXE	Digas	A (remote) GUI for Broadcast Utility Server.
BUSGUI64.EXE	Digas64	64-bit variant of GUI for Broadcast Utility Server
DigAIRange.exe	Digas	The scheduling client
TurboPlayer.exe	Digas	The OnAir client. There are many more modules in conjunction with TurboPlayer, see the extra technical manual.
TurboPlayerService.exe	Digas	A .NET module running as Windows service. It provides a WebSocket interface (partially also REST) to the engine of TurboPlayer.
DigaStat.exe	Digas	The statistics module
DigaText.exe	Digas	One of the DigaSystem text editors
InfoPad.exe	Digas	One of the DigaSystem text editors
RESTART2.EXE	Digas	Restarter tool needed by BCS to switch from master to buddy mode or by BUS to perform a self-restart in case of fatal errors.

1.2.2 Libraries

Library	Directory	Description
AbyPromoImport.dll	Digas	Import module for Antenne-Bayern Promo files
AdComImport.dll	Digas	Import module for XML files from the DigaSystem AdCom Manager
Audio32.dll	Digas	Helper dll of DigaSystem with routines for analysis and conversion of audio files.
Audio64.dll	Digas64	64-bit variant of Audio32.dll
BCSConsistencyCheck.dll	Digas	License DLL which is required for BUS task “ConsistencyCheck”. Since version 6.0 the dll is deprecated. The ConsistencyCheck is now always included without additional license.
BCSHelpers.dll	-	Contains some helper routines in C++ made available for C# modules. This includes especially metadata handling and file handling routines. So far only used by BCSService.
BCSLoudness.dll BCSLoudness64.dll	Digas	License DLL which is checked by all applications if a loudness-related function is going to be used.

BCS_MDPLicense.dll	Digas	License DLL which is required for BUS, FileCheck task, feature: Media Directory Profiles. Since version 6.2 the dll is deprecated. The media directory profiles are now always included without additional license.
BCSSearch.ocx	Digas	The user interface for the fulltext search system. (Hint: since version 2.0 of this module the IBCS.dll is no longer needed.)
CBSIImport.dll	Digas	Import module for the Wicks CBSI system (US tool for commercials)
CBSIExport.dll	Digas	Export module for the Wicks CBSI system
ClipboardFormatConversions.dll	Digas	.NET assembly used by OnAIR TrackMixer. It provides helper functions for the conversions of the formats DBE, DBX, BCS-XML.
cpprest140_2_9.dll	Digas	Helper dll with parts of Microsofts Casablanca (C++ REST) project. Currently only used by ElasticIndex.dll. version 1.0. **Not** used by newer versions of ElasticIndex.dll.
CrossfadeMixer.ocx	Digas	The crossfade editor module used by DigAIRange, DigaROC and TurboPlayer. Deprecated and replaced by OnAIR TrackMixer.
CrossfadeMixer_2.ocx	Digas	Version 2 of the crossfade mixer. Deprecated and replaced by OnAIR TrackMixer.
CxImageList.dll	Digas	Library for displaying many images (thumbnails).
DigaContentTabHost.dll	Digas	Library used by DigAIRange to display .NET user masks. Moderates between the .NET and the Win32 world.
DigaLegend.ocx	Digas	A module used from DigaStat for displaying statistical data
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.
DigaSQL64.dll	Digas64	64-bit variant of DigaSQL.
DigaStory.ocx	Digas	An ActiveX control used to visualize and edit stories – aka mini-rundowns consisting of text and media elements.
DigaTimeChart.ocx	Digas	A module used from DigaStat for displaying statistical data
DirectImport.dll	Digas	Import module for reading files in the native BCS XML format.
DirectImport64.dll	Digas64	64-bit variant of the DirectImport.dll.
DirectBCSImport.dll	Digas	Import module, which connects directly to a source BCS.
DirectBCSImport64.dll	Digas64	64-bit variant of the DirectBCSImport.dll.
DirectExport.dll	Digas	Export module for writing the XML data unchanged into files
DirectExport64.dll	Digas64	64-bit variant of the DirectExport.dll.
EasyXMLExport.dll	Digas	Export module which writes a single XML file which contains only the current, last and next element. Old module, was replaced by the FlexXML.Dll.
ElasticIndex.dll	Digas	Helper dll which talks to an ElasticSearch fulltext engine and provides a generic interface for the BCS side. (Version 2 or newer do no longer need the file cpprest140_2_9.dll.)
ElasticIndex64.dll	Digas64	64-bit variant of the ElasticIndex.dll to be used with BUS64.EXE.
EndPoints.dll	Digas	BCSService helper dll, reading DistributionEndpoints from registry.

FlexIE.dll	Digas	Standard module for text import and export. It is very configurable.
FlexXML.dll	Digas	Export module to generate XML output. Especially designed to create information about running elements, e.g. for Internet publishing.
FullExport.dll	Digas	Like the DirectExport.DLL but it exports all media files too
IBCS.dll	Digas	This DLL provides an easy COM Interface to the BroadcastServer, especially for simple clients. BUS, DigAIRange, DigaROC and TurboPlayer do not need it because they have an internal interface to the server.
IBOCExport.dll	Digas	Export module for the IBOC system.
KbdHook.dll	Digas	The DLL installs a system hook, which captures all keyboard events. Needed by DigaROC.
lame_enc.dll	Digas or Windows	For the creation of MP3 files BUS can use the LAME encoder. LAME can be found here: http://www.mp3dev.org.
Libmpg123.dll	Digas(64) or Windows	This is an audio code library for encoding or decoding MP3 files. It is needed by Audio32/64.dll. Attention: it exists in a 32-bit and in a 64-bit version, but both modules have exactly the same filename!
LoudnessDLL.dll	Digas	Contains the functions for a loudness analysis. Optionally used by BUS.
LoudnessDLL_x64.dll	Digas64	64-bit variant of the LoudnessDLL.dll.
MarketronExport.dll	Digas	Export module for the Marketron system.
MarketronImport.dll	Digas	Import module for the Marketron system.
MFCxxxU.DLL	Digas or Windows	"Microsoft Foundation Classes". Part of the Microsoft Visual Studio C++ Runtime – see chapter “Additional runtime files”.
MPConnect.dll	Digas	MultiPlayer interface, deprecated and no longer used.
MP2.SYS	Digas	A helper file for the audio conversion
MSCAL.OCX	Windows	Microsoft calendar control
MSVCPxxx.DLL	Digas or Windows	"Microsoft Visual C++ Runtime Library. Part of the Microsoft Visual Studio C++ Runtime – see chapter “Additional runtime files”.
MSVCRxxx.DLL	Digas or Windows	"Microsoft Visual C Runtime Library. Part of the Microsoft Visual Studio C++ Runtime – see chapter “Additional runtime files”.
MSXML3.DLL	Windows	Contains the Microsoft XML parser version 3. Needed by DigAIRange and XSLExport.Dll to work with stylesheets. Must be registered with RegSvr32.Exe !
MSXML3A.DLL	Windows	Helper DLL for MSXML3.DLL
MSXML3R.DLL	Windows	Helper DLL for MSXML3.DLL
MultiRec.ocx	Digas	The MultiRecorder module. Used for prelistening and crossfading
MultiRec_X.ocx	Digas	Version X of MultiRecorder. Current version is MultiRec_4.ocx.
MURAExport.dll	Digas	Export module for the swedish MURA system
MuscImport.dll	Digas	Import module for reading the XML files form MusicScanner. This is a DigaSystem tool which converts MusicMaster text files into XML. The DLL will be replaced by an import module which can work directly on MusicMaster text files.
MusicMasterIE.dll	Digas	Import and export module for MusicMaster. Uses ASCII/ANSI files with section strings for import and column formatted text for export

MySQLExport.dll	Digas	Export module which enters played and skipped elements into a MySQL database
NewsPlayerXLogExport.dll	Digas	Export module to create as-run logs in the format of NewsPlayerX.
NewsScheduleXmlIE.dll	Digas	Import and export module for XML files of the NewsSchedulerX / NewsPlayerX system.
OtmControl.ocx	Digas	The main ActiveX control for OnAIR TrackMixer. Must be registered. The ocx with the additional files Audio32.dll, ClipboardFormatConversions.dll, OtmControlLibrary.dll and System.Windows.Interactivity.dll must all be in the same folder as the programs which use the ocx – e.g. DigAIRange ! Absolutely have a look at Installation of OnAIR TrackMixer to install all necessary files.
OtmControlLibrary.dll	Digas	Helper assembly for OnAIR TrackMixer.
pIF.dll		Interface DLL for the ClassicPop system
RadioADServerLicense.dll	Digas	License DLL needed by BUS to run the RadioADServer task. Since version 6.2 the dll is deprecated. The RadioADServer task is now always included without additional license.
SelectorIE.dll	Digas	Import and export module for the standard format of Selector DOS V12
SelectorImport.dll	Digas	Import module for reading schedules from RCS Selector. The Dll uses the Selector XML interface which exists only for version 14.
SelectorStdImport.dll	Digas	Import module for Selector DOS files, section string based
SeplImport.dll	Digas	Import module for reading SND files from the 16 Bit DigAS News Scheduler or from SePl.Exe
StandardIE.dll	Digas	Old module, replaced by FlexIE.Dll. Configurable module to import / export text files.
SuisaExport.dll	Digas	Export module for the swiss SUISA system
SXLRT308.dll	Digas or Windows	Old module. The Dinkumware C++ Runtime library. Some of the DigaSystem programs used in the past. No longer used.
System.Windows.Interactivity.dll	Digas	Helper assembly for OnAIR TrackMixer.
TrafficCopExport.dll	Digas	Export module for the Traffic COP System.
TrafficCopImport.dll	Digas	Import module for the Traffic COP System.
WASCO.DLL	Digas	Interface for the IO card
WWFImport.dll	Digas	Import module for the WWF commercial system.
XSLExport.dll	Digas	Export module which treats the XML data with XSL-stylesheets and writes the result to a file.

Documentation and Help Files

File	Directory	Description
BCSNews-Installation.pdf	Doc	A step-by-step manual how to install all components of BCS system in order to replace a NewsPlayer setup.
BCSTechManual.pdf	Doc	This manual: the technical manual of the Broadcast System
BCSProgrammersManual.pdf	Doc	Documentation for programming with the interface of Broadcast Server. Only available for version 6.0 or newer.
BCSSearchHelp.rtf	Digas	Help document for an integrated usage within BCSSearch.ocx. Put it to the same directory as the ocx.
DigaROCTechManual.doc	Doc	The technical manual of DigaROC – deprecated
DigaROC.doc	Doc	The user's manual of DigaROC - deprecated
TurboPlayerTechManual.pdf	Doc	The technical manual of TurboPlayer
WebTurboPlayerTechManual.pdf	Doc	The technical manual of WebTurboPlayer
TurboPlayerService_Programmers Manual.pdf	Doc	A programmer's manual for accessing TurboPlayerService.
DigAIRange_x.y_Manual_en.pdf	Doc	The user's manual of DigAIRange. It is available in English andGerman. The current version is for 5.0
FlexIE.doc	Doc	The technical manual for the FlexIE.Dll.
FlexXML.doc	Doc	The technical manual for the FlexXML.Dll.
NewsScheduleXmlIE.doc	Doc	The technical manual for the NewsScheduleXmlIE.Dll.
DIGPARAM.RTF	Doc	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(x)	Doc	The release notes for the corresponding module.

Please be aware that newer documentation can be found in the internet on the learn platform of DAVID Systems (especially for the web tools and web services):

https://learn.davidsystems.com/

1.2.3 Miscellaneous Files

File	Directory	Description
*.SKF	Digas	Skinfiles for the MultiRecorder.
*.XSL	-	XSLT-Stylesheets for converting the internal XML data into other formats, e.g. HTML pages.
*.RTF	-	Printing in DigAIRange can be done with the help of RTF templates.
*.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.

1.2.4 Additional runtime files

The modules are built with Microsoft Visual Studio C++ for the Windows (Win32) API. Most of the modules are linked statically to the required libraries, which means that no additional dlls need to be installed. Since version 5.2. DigAIRange is linked dynamically. This is necessary in order to get the glue code in DigaContentTabHost for .NET user masks running. Therefore, it is now required to install the correct Microsoft Visual Studio C++ Runtime libraries. You can find it here:

> Visual C++ Redistributable for Visual Studio 2017, 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.

If you want to make use of .NET user masks you also need:

> Microsoft .NET Framework 4.5, which can be downloaded from the link :
> https://www.microsoft.com/en-us/download/details.aspx?id=30653
>
> DigaContentTabHost.dll in the same directory as the DigAIRange.exe, at least version 1.1.7.0

1.2.5 Customizable image files

For the visualization of some fields (custom fields, events, mixer sources) the programs can make use of configurable images. This affects only the mentioned fields. For most other fields the programs use images, which are included within the executable, and which cannot be configured. If you want to provide custom images, here are some guidelines for image files:

- Use the file formats: BMP, PNG or EMF (for desktop programs) and PNG or SVG (for web programs). Other file formats (like JPG, ICO or TIFF) are supported, too, but they are not recommended. Unfortunately, the used technology for the Win32-API (desktop) and for web programs do not support a single usable format for vector graphics. If you need both, you will have to provide both formats, SVG and EMF. You can use the free tool Inkscape to convert between these formats.
- Always make the images a square. Otherwise, the programs will stretch the images into a square, which might not look optimal.
- The size (pixels) of the images should be big enough, according to your needs. The programs will display the images with approximately the line height in rundown views and this line height is implicitly configurable by selecting a font. With the current implementation (which might change in the future) the programs will display an image without resizing it if the image is square, not bigger than the output space and not smaller than 90% of the output space.
  If you have high-DPI screens, you should multiply the size with the maximum used zoom level. For regular fonts sizes (like Arial 10pt) and 100% zoom level, an image size of 24*24 pixels is a good starting point. You will have to make experiments to see, which size gives the optimal result. It is not possible to provide images in multiple different sizes and make the programs chose the best one. Instead, the programs will always load the images in their given/natural size and will stretch them when they must be displayed.
- For PNG files you have to use a color depth of 32 bits – 8 bit for RGB + 8 bit for the alpha channel. The alpha channel allows to make the background around symbols transparent.
- BMP files support a transparency, too. In principle, BMP files can have an alpha channel, too, but this is a very special feature with not much support and compatibility problems. Therefore, it is not recommended to make use of it. Instead, you should create these images with 3x8 bit RGB values. The special/reserved

color RGB=(255, 0, 255) is used as “this pixel should be transparent”. Unfortunately, this allows only the two extreme values fully transparent/fully opaque, and you cannot create smooth borders around symbols with a gradient to transparency. Therefore, it is not recommended to use BMP files if you want to achieve a nice transparency around symbols. Better use the PNG format.

- EMF/SVG files are vector graphics, and they can be perfectly stretched to any real size. Nevertheless, these images have a natural size, too. If you start a new document in a vector graphics editor, you should select a square with a reasonable size, e.g. 10 * 10 mm or 100 * 100 pixels. Possibly, the editor provides a template like “Icon 48 * 48”. This would be a good starting point.
- Do not paint the symbols themselves too big. They should not fill the available space of the image (the “canvas”) completely. It is recommended to leave a small transparent border around symbols. How much, need to be evaluated by trying out different values. If you work with pixels, start with at least 1 or 2 pixels around the symbols. And pay attention to not put the symbol into a corner of the image. It should be centered.

1.3 General Installation Hints

1.3.1 System requirements

The **BroadcastServer** needs at least Windows 10 / Server 2019 or one of its successors. A server version is not necessary. BroadcastServer is available as a 32-bit variant or as a 64-bit variant. Nowadays, the 64-bit variant is always recommended. The 32-bit variant should only be used if you have to upgrade an existing installation step by step to 64-bit.

Minimum hardware requirement:	Recommended hardware:
2 GByte RAM, single core CPU	>= 16 GByte RAM, Xeon, >= 16 cores

Be aware that the minimum requirement is only sufficient for a very small system with a few clients. If you have many clients, a full program with many elements or many remote links you must spend as much memory and cpu power as possible.

You should also know that we can only recommend to run BCS on a physical computer with professional server hardware and integrated hard disks, either mirrored or as RAID array. This is especially important if BCS should run in a cluster with a master/buddy pair. Here there is even one more requirement: both computers of the cluster must be connected by a single switch, which handles all client connections. Otherwise, it cannot be guaranteed that the synchronization between the two BCS servers of a cluster will work correctly. Especially in a combination of virtual machines and the usage of a NAS instead of local hard disks, customers have encountered problems with the synchronization. In such an installation multiple problems with network caching or incompatibilites between Windows and a Linux-based NAS can arise.

The **BroadcastUtilityServer** needs at least Windows 10 / Server 2019 or one of its successors. A server version is not necessary. BroadcastServer is available as 32-bit variant or as 64-bit variant. The necessary hardware performance depends heavily on the running tasks. If you have a small system and run only a file-check, copy and an import-task e.g., a very slow system might be sufficient. As a general rule: more memory is needed for every task running, especially the file-check needs a lot of memory. Processor speed is needed especially for the import tasks or if you want to convert audio files.

Minimum hardware requirement:	Recommended hardware:
1 GByte RAM, single core CPU	>= 16 GByte RAM, Xeon, >= 8 cores

1.4 Known limitations and incompatibilities

This chapter describes important limitations and incompatibilities when using specific modules within BroadcastSystem. Sometimes, they might arise because of real technical limitations which do not allow a better solution. But mostly, these problems exist because BroadcastSystem is a “living” system which is permanently extended with new features, changes or bugfixes and the ongoing development does not occur in all modules at the same time. Therefore, problems arise if you use different modules which are not yet compatible in every detail.

As the state of the development changes permanently, please check the date of this document on the last page in order to know whether the information here is up to date.

IMPORTANT: This list is NOT a complete list which contains every known problem ! Only some important problems are listed.

1.4.1 IPv6 support

Affected modules:	BCS and all clients
Solved in version:	BCS 6.0
Problem / consequences:	IPv6 support is not supported for SeplProtocol communication.
Details:	Beginning with version 6 BCS and the clients can use WebSockets (instead of the old SeplProtocol) with an implementation based on boost::asio. This library supports IPv6.

1.4.2 TLS support

Affected modules:	BCS 6.0 and newer
Problem / consequences:	When using encrypted WebSocket connections (WSS, TLS) it can happen that the connection is lost because of received data garbage. The probability increases significantly with the data load on the connection.
Details:	In BroadcastSystem we make use of OpenSSL for implementing data encryption for the IP communication (TLS). Unfortunately, the OpenSSL library has a severe design flaw: it uses the same buffer for decryption of received data as for encryption of data to be sent and this buffer seems to not be protected for multithreaded access. This is no problem for a http communication, which is always done in a synchronized way. But with WebSockets data communication can happen asynchronously in both directions and then it can happen that received data overwrites data to be sent or vice versa. This is more likely when the traffic increases. At the moment no solution is available. Of course, it would be best to fix the design flaw in OpenSSL, but it is unclear, whether this is going to happen. A possible workaround might be to use two open connections, one for each direction, but this is not yet implemented.

1.4.3 64-bit modules

Affected modules:	BCS and BUS
Solved in version:	BCS 6.0, BUS 6.2
Problem / consequences:	BCS in pre 6.0 versions and BUS in pre 6.2 versions are not available as 64-bit (x64) application.
Details:	For BCS version 5.x or older the memory BCS can use is limited to 3 GByte. If you want to use a lot of services, pools, metadata, etc. and/or at the same time keep many days of the rundown in memory this might be a problem. Please check the BCS chapter below for more information how to tweak your system. Beginning with version 6 BCS is available as a 64-bit application. See below in the chapter for BCS installation for more information. Beginning with version 6.2 BUS is available as 64-bit application. BUT: BUS is depending on a lot of possible dlls – there are especially a lot of import and export dlls. As a 64-bit application can only use a 64-bit dlls, all these dlls must be converted, too. This was not done yet. If you need to use BUS with a dll, which is not yet converted, contact DAVID Systems!

1.4.4 BCS and many clients

Affected modules:	BCS
Solved in version:	BCS 6.0
Problem / consequences:	For BCS version 5.x or older the number of clients BCS can handle is limited. Though it might work for you with many more clients, you might also encounter problems starting with 50 clients.
Details:	The reason for this problem is that old code is used for handling IP. Though this code is very well tested and very reliable it is still based on a programming model for Windows 95. This model does not work well when many IP connections have to be served. Beginning with version 6, BCS and its clients can use a WebSocket-based communication instead of the old one called "SeplProtocol". If you use WebSockets, the number of possible clients is only limited by the resources of the computer, especially the CPU and the network adapter. In addition, version 6 / 64-bit BCS has an advanced mechanism for locking the internal data, which allows simultaneous access for clients in many cases. See parameter description for: "BroadcastServer\...\General\UseAdvancedTreeLock" for more information.

1.4.5 BCS and 24 h shows / shows across midnight

Affected modules: BCS

Affected versions: All with differences between the versions

Problem / consequences: When you are using shows for the whole day (24 hours or nearly as long) or if you define shows which overlap 2 days you might encounter problems in the times BCS computes: BCS assigns a wrong date to nodes. This might not be noticed during scheduling, but later TurboPlayer will not play elements or play them too early / too late.

Details: In order to get correct results, you should limit the duration of shows to the following rules:

- Generally, try to keep shows short, especially if there will be many elements in them.
- A 24h-show is o.k. if it starts and ends at midnight.
- If you really need a show which runs across midnight, see that it does not start before noon (12:00:00) and does not end later as (01:00:00).
- If you need to put a midnight-crossing show into a day- or show-template, please be aware that the templates do not have dates in the timestamps. Therefore, the time computation within the templates will be wrong for nodes after midnight. This might be corrected when the template is used for a real day, you need to try it out.

1.4.6 BUS FileCheck and audio compiled by CrossfadeMixer

Affected modules: BUS

Affected versions: All

Problem / consequences: Audio files which have been compiled by CrossfadeMixer are not treated by BUS FileCheck and only generate errors in the log.

Details: Neither the audio classes nor Audio32.dll can handle the compiled audio files of CFM when audio data must be extracted from the file. It would be possible to implement, because these files consist of a primary wave chunk (which is being played) and additional project information, but this was not done yet. Therefore, a format conversion, a loudness analysis and a waveform generation do not work in BUS. Only the first analysis and a pure copying to additional media directories work (but only if no loudness analysis or waveform generation is configured).

1.4.7 DigaMOS and BCS 64 Bit

Affected modules: DigaMOS in combination with BCS64

Solved in version: 4.4.312.0

Problem / consequences: DigaMOS cannot connect to BCS64.

Details: Old versions of DigaMOS used IBCS.dll for the communication with BCS. This dll encapsulates only the old TCP/IP classes for the communication via SeplProtocol. This old protocol is only supported by a 32-bit BCS, but a 64-bit BCS supports only WebSockets.

1.4.8 DigaMOS and BCS locks

Affected modules: DigaMOS if the users want to set BCS locks.

Affected versions: All

Problem / consequences: DigaMOS is not able to react correctly on existing user locks of BCS nodes. If it cannot write to all BCS nodes which are under control of MOS a desynchronization of the rundowns on both sides of MOS will occur.

Details: So far neither in DigaMOS nor in BCS routines were implemented which can handle the case of existing user locks when DigaMOS wants to write to such a node at the same time.

The only way to avoid problems is to teach your users to never use BCS locks in rundown which are synchronized by DigaMOS.

1.4.9 DigaMOS and web modules

Affected modules: DigaMOS in combination with any web application like: WebDigAIRange, WebTurboPlayer or any self-written web application.

Affected versions: All

Problem / consequences: If web applications are used together with DigaMOS, node operations like insert, move, copy or delete might not be handled correctly. A desynchronization of the rundowns on both sides of MOS might occur.

Details: In the MOS protocol node operations like insert, move, copy, delete must be handled differently as in BroadcastSystem. In BCS system some operations might be split up into multiple individual parts which is not possible with MOS. Therefore, DigaMOS need to get information about operations which belong together. For that purpose, the transaction ID is used which must be generated according to chapter 2.6. So far, no web application and also not BCSService or TurboPlayerService can generate these transaction IDs.

1.4.10 OnAIR TrackMixer and CrossfadeMixer features

Affected modules: OnAIR TrackMixer

Affected versions: All

Problem / consequences: OnAIR TrackMixer is intended to be a replacement of CrossfadeMixer but not all features have been implemented so far.

Details: Not implemented is:

- The page for live recordings of CFM is missing. Therefore, the rebroadcast workflow as described in chapter 10.4 is not available.
- Handling of elements without a file is partly missing. CFM had displayed a virtual sinus waveform for these elements. OTM does it at least for some type of elements.

1.4.11 DigAIRange and remote links

Affected modules:	DigAIRange, BCS
Affected versions:	DigAIRange all versions, BCS beginning with version 4.7.264.0 of 2014-09-15
Solved in versions:	DigAIRange 5.5.732.0 + BCS 4.9.276.0
Problem / consequences:	Replacing a day or a show (e.g. from the templates) does not work if a show or a track uses remote links. The BCS error “An operation on a mixture of local and remote nodes was tried which is not allowed.” Is displayed.
Details:	With the mentioned BCS version the routines for locking nodes were completely rewritten. It was not included to set a lock for a node which has children with remote links, because this would imply a very complicated logic of distributed locking-transactions across multiple servers. DigAIRange makes use of locks when replacing days/shows but nevertheless: for the mentioned workflow of replacing a day/show these locking- transactions would not be needed. Unfortunately, BCS is not yet able to detect this special type of operation in order to allow it despite of remote links. This detection simply is missing and needs to be developed.

1.4.12 Undo handling and multiple deletes

Affected modules:	DigAIRange, DigaStory
Affected versions:	All of DigaStory, most old ones of DigAIRange
Solved in versions:	DigAIRange 5.8.780.0 + BCS 5.8.370.0, not yet in DigaStory
Problem / consequences:	If the user makes multiple delete operations and afterwards tries to undo them this might not work and the elements are appended to the end of the rundown.
Details:	The reverse operation of a delete is an insert. If such an operation is stored in the undo manager the ID of the previous and/or parent node is stored in order to known at which place a node must be restored by the insert. But in an insert operation BCS assigns new IDs to nodes. Therefore, it can happen with multiple deletes and inserts that the IDs which are stored in the undo operations are no longer valid and it is undefined at which place a node must be restored. Therefore, the undo manager can only insert them at the end of the list.

1.4.13 Startup dialogs and invisible message boxes

Affected modules:	DigAIRange, TurboPlayer
Affected versions:	All
Solved in versions:	DigAIRange 5.5.732.0 + TurboPlayer 5.5.2120.0
Problem / consequences:	At startup DigAIRange and TurboPlayer both display a window which signals the startup progress. Unfortunately, these windows have the “always-on-top” attribute. Therefore, any messages which possibly are displayed during startup are behind the startup window.
Details:	If you encounter the situation that DigAIRange or TurboPlayer hang during startup, please have it in mind that there might be a message box in the background.

1.4.14 TurboPlayer and relative starts

Affected modules:	TurboPlayer
Solved in versions:	5.4.2095.0 + 5.3.2073.3 + 4.9.1829.7
Problem / consequences:	Elements started relative will start at the wrong time if links are used.
Details:	When TurboPlayer performs a relative start (Relative/Relative, meaning across multiple tracks or via remote TurboPlayer) the link-in of the master element is not taken into account as it should happen according to the definition of the metadata fields. Therefore, the relative elements will start at the wrong time if the master element has a link-in.

1.5 Right Objects

As usual in DigaSystem, all modules use the rights system of the DigaSQL.DLL. Rights are granted per service (program) and are checked by the server, which is responsible for the service. Access control lists for shows in the tree are available too.

1.5.1 Objects

The object type "ShowPlan" is mainly used. A lot of objects of this type can be created and up to eight rights can be granted for each object. The first name of each object is the servicename (program) which must be unique in the system. The following objects are possible:

- **Service**: This is the general object for the whole service and must always be present. Typically, this object is checked if there is no specialization for a specific class. For example, if someone wants to edit a news element, the news object is checked first. If the news object is not present, the right is checked with the service object. This is valid for the “Read” and “Write” right. For the other rights like “OnAir” or “Manage” the service rights either override the rights for a specific class or are independent.
- **Service\News, Service\Text, Service\Music, Service\Cart, Service\Commercial, Service\Control, Service\Command, Service\Graphic, Service\Video, Service\Audio, Service\Story, Service\Magazine, Service\Promotion, Service\Live, Service\Line, Service\Info**: This are the objects for each class. The write and manage right are evaluated, see “Rights” below. The class rights apply only in the normal schedule in scheduling tracks. For other nodes like pools there are special right objects which apply independent of the class:
- **Service\ScheduleTracks**: For schedule tracks of shows. The manage right is needed to create or delete such a track. The read / write rights are not evaluated.
- **Service\SnapshotTracks**: For snapshot tracks of shows. The manage right is needed to create or delete such a track. The write right is needed to write into such a track. Read is not evaluated.
- **Service\ProtocolTracks**: For protocol tracks of shows. The manage right is needed to create or delete such a track. The write right is needed to write into such a track. (Hint: the “OnAir” right for the service is also sufficient to create and write into protocol tracks.) Read is not evaluated.
- **Service\PreProductions**: Object for the pre-production nodes. The “read”, “write” and the “manage” right apply.
- **Service\Jingles**: Object for the jingle nodes. The “read”, “write” and “manage” right apply.
- **Service\DayPools:** Object for all day pools. The “read” and the “write” right apply.
- **Service\ShowPools:** Object for all show pools. The “read” and the “write” right apply.
- **Service\Pools**: This object is an old one, which was replaced by individual objects for day pools and show pools. If you do not define the two separate objects, you can still use this one for both, day and show pools.
- **Server\GeneralPools**: These pools are stored per server and not per service, so the object specification includes the server name. By default, this is the Windows/NetBIOS name of the computer on which the server is running. If you have a master/buddy server running, you can use the parameter “ServerNameForGlobalRights” to achieve that both servers use the same server name. It is recommended to use the same name for the master BCS, for the parameter “ServerNameForGlobalRights” and for the BCS key name in the DigaSystem registry. Otherwise, you might have to create multiple right objects with different names for one purpose. E.g. DigAIRange will evaluate the registry key name for the BCS to read out the right object, which allows the user to open the manage dialog.
  Hint: Every user can always read, write and manage his personal pools; there is no right object for this pool type. If you do not want to use these pools, you either do not create a personal pool root for a specific user or you can hide the corresponding tree item in DigAIRange for such a user.
- **Server\GlobalJingles**: Since build 243 BCS supports jingles on the same level as services – the "global jingles". Therefore, you need an extra right object. It has the usual rights: read, write and manage. The computer-name problem described for the general pools above applies here too.

One hint: you do not have to create or rename all the objects described so far manually, this is done automatically by DigAIRange when you create or rename a service (program).
But there are three additional right objects which must be created manually. All are of type “Action”:

- **DigAIRangeChangeSettings**: This right is needed for opening the Settings dialog in DigAIRange. If a user can open the dialog, he can also change the settings. If a user does not have this right, the supervisor must change the settings for him. Therefore, DigAIRange displays an admin password dialog when someone without rights wants to change the settings.

- **DigAIRangeLockShow**: In DigAIRange a user can either lock a single element or a whole show. Locking a single element is always possible without the right to do it, because it is sometimes necessary to set a lock to avoid data loss. On the other hand, locking a whole show can be dangerous. Therefore, you can restrict the use of this feature to some advanced users only. Limitation: neither a show nor the elements in it can be locked while the show is “OnAir”.
- **BCSSearch_Navigate:** This object is evaluted by the BCSSearch.ocx which displays the results of a fulltext search. When you double-click a line in the result list DigAIRange will "navigate" to the corresponding element, meaning: the parent node is opened in the upper list view and the element is selected. As this might create a high server load (for each element in a new day the BCS must load the whole day into memory) you can limit this feature to certain users. As usual: if the object does not exist, every user can perform the navigate operation.

Since version 3.0 of DigAIRange an additional object type does exist: "**BCS**". Here objects can be created with the name of the BCS in the registry (please check the parameter "BroadcastServer\...\General\ ServerNameForGlobalRights"). These objects are evaluated by BCS and by DigAIRange. If an object exists for a BCS and the current user does not have the "use" right on it, the BCS is not listed in the tree view and similar lists within DigAIRange. A second right which can be granted to a BCS object is the “manage” right. It allows a user to: perform every operation on a BCS like the special user “ADMIN” could do, connect as a remote GUI and connect as a buddy BCS (see the chapters 6.5 and 6.8 for more information on these topics).

Since Release 5.6 / 2019.1 a new object type was introduced: "**BUS**". The names used for the objects are free, because in the settings of BUS you can configure, which right object should be evaluated. Nevertheless, it is recommended to use the name of a BUS instance as it is listed in the DigaSystem registry as subkey of the the top-level key: "Digas\BUS". Admin and DPE-Admin allow to grant one right onto a BUS object: Manage. If a user has this manage right, the corresponding BUS instance will appear in the tree view of DigAIRange and the user can connect with a remote GUI to the running BUS instance. BUS itself will check the right, too, when a user tries to establish a connection via remote-GUI.

1.5.2 Rights

Depending on the object type up to eight rights can be granted to each showplan object:

- **Read**: The user can open and see the elements and subnodes. For class objects this right is not used.
- **Write**: The user can create, change and delete the elements and subnodes.
  For groups and stories and if the BCS setting “Settings\GroupClassGrantsAccessRightToAll” is TRUE there is a special meaning: If someone has the right to write elements of a specific class he can also work with ALL allowed elements in a group of this class.
  This right is also needed for the import: Merging an imported schedule with the internal data is always based on a primary class of the import. The write-right for the primary class is necessary.
- **Write group**: This service-right is necessary to create or change any group (the group “header”, not the elements in the group). In addition, the class rights do apply if the group has a specific class.
- **Write story**: This service-right is necessary to create or change any story (the story “header”, not the elements in the story). In addition, the class rights do apply if the story has a specific class.
- **Set element ready**: Needed for a user to set the sendstate of an element to “cleared”.
- **Set show approved**: Needed for a user to declare a show “approved” and/or “protected” or to remove these states and set a show back to “changed”.
- **Write to approved show**: Needed for a user to make changes in an approved and protected show.
- **Change current show:** Needed for a user to make changes in a show which is currently on air (OnAir detection is time based only !) ATTENTION: If you make tests with the rights, you might typically use the current show. If your user does not have this right, other checks like for class rights are obsolete. This is a common stumbling block.
- **OnAir**: A special right which exists only for services. It should only be granted to real OnAir clients and includes the right to read everything, write to every existing element or group and create new elements in log tracks.
- **Manage**: For node objects it means a user can call the manage dialog and edit the list of subnodes, e.g. the list of jingle groups or tracks. For class objects it means a user can create, change and delete groups and stories of the class. Since version 1.2 this right is also necessary to change start attributes of elements in groups and in stories. For rundown objects (tracks, pools, pre-productions) it is the right to create, delete such a node, edit the properties of the node, change remote links and to replace the whole node itself. For service objects the manage right is the “supervisor” right which is the right to change everything in the service, especially the right to create shows, tracks and templates. Limitation: the manage right for a service is not sufficient to change the top-level properties of the service itself, e.g. rename a service or delete it or change media directories. For this you have to be Admin or you must have the manage right for the BCS itself.
  For objects of type "BCS" and "BUS" the manage right is needed to connect to a running instance via remote GUI and to execute program commands or change any settings.

1.5.3 Access control lists

The rights and objects so far are defined with the Admin/DigaSQL.DLL/rights registry. But there is also the possibility to define access rights within the broadcast tree itself. Each show has an access control list. It can be edited with DigAIRange, either in the manage dialog of the show or in the team view for the show. The ACL is a list of names of DigaSystem users and groups. If no name is listed at all, the access to the show is free. If names are listed, only the corresponding users or members of the listed groups have write access to this show. This is important: the ACL controls only the write access to the show, reading is always possible. If a user is listed, his detailed rights in the show still depend on the other right objects, especially the class objects.
As additional feature a chief role can be assigned for a show. Each show has assigned team members. One of the team members is the chief. The chief is a DigaSystem user which can be granted rights for shows. If a user is listed as chief for a show, he has the sum of all rights: his personal ones plus the manage right for the show.
A “higher” right (one of the administration rights or the OnAir right) supersede an existing ACL. This means: a user, who has such a higher right, need not be listed in the ACL in order to have write access.

Since December 2006 you can also assign ACLs to groups (blocks) within a rundown. They work like show ACLs. If both, a show and a group ACL is present, it is sufficient that a user is in one of them to give the user the right to work with a group. That means, if a user is listed in a show ACL, you cannot deny him the right to a specific group by not listing this user in the group ACL.

Since July 2020 (BCS 5.8.377.0 / 6.0.401.0, DigAIRange 5.9.801.0) a jingle group can have an ACL, too. The behaviour is identical to the ACL of a show.

See chapter 8.5 about the DigAIRange manage dialogs for information how to set an ACL.

1.5.4 Administration rights

Important: You should be aware that there are four types of administration rights, which override lower-level rights!

- The highest possible right is of course the DigaSystem “Admin”. He can do whatever he likes with every server, services, shows and all type other of nodes.
- The next level is a user, who has manage right for a BCS.
- Next comes the manager of a service. He has the same rights as the admin but only within his service.
- The lowest level is the chief of a show. He can work with everything in his shows and can edit the access control lists.

2 XML DATA STRUCTURE

2.1 Overview

The whole plan is realized as a single big tree. In each level there may be certain elements, some of them having children, some of them don't. The following tree structure is used, the number in brackets specifies how many times each node can be on the same level.

```
Root (1)
       DefinitionRoot (1)
              Section (n)
                     Definition (n)
       PoolRoot (1)
              GeneralPoolRoot (1)
                     Pool (n)
                            [Rundown]
              PersonalPoolRoot (n)
                     Pool (n)
                            [Rundown]
       GlobalJingles (1)
              JingleGroup (n)
                     Element (n)
       Program (n)
              DayTemplates (1)
                     DayTemplate   (n)
                            Show   (n)
                                   Pool (1)
                                          [Rundown]
                                   Track (n)
                                          [Rundown]
              ShowTemplates (1)
                     Show   (n)
                            Pool   (1)
                            Track (n)
                                   [Rundown]
              PreProductions (1)
                     PreProduction (n)
                            [Rundown]
              Jingles (1)
                     JingleGroup (n)
                            Element (n)
              ConsistencyCheckResults (1)
                     Show (n)
                            Track (n)
                                   Inconsistency (n)
              Year (n)
                     Month (n)
                            Day (n)
                                   Pool (1)
                                          [Rundown]
                                   Trash (1)
                                          [Rundown]
                                   Show (n)
                                          Pool (1)
                                                 [Rundown]
                                          Track (n)
                                                 [Rundown]
```

Where [Rundown] is a placeholder for this structure:

```
[Rundown]
       Element (n)
       Story   (n)
              Element  (n)
       EndpointStory (n)
              Element  (n)
       Group   (n)
              Element  (n)
              Story    (n)
                     Element  (n)
              EndpointStory (n)
                     Element  (n)
```

2.2 Description of the different nodes

Though not listed, each node has a special property "TagName". This is the name in the left column and is used to distinguish the node types.
Some nodes which can hold many properties are separated into parts. This is done for a better overview only. These parts do not appear as subnodes, instead each property has a name which starts with the part. The actual name follows behind an underscore. E.g. the start time of an element is in the part "Time part" and the start property is called "Time_Start". Some of the nodes have the prefix “PD_” which means “private data”. These tags are private for the BroadcastServer and are not listed here (except the information for remote links). If you ever encounter a field starting with “CD_”: this prefix is reserved for “client data”. These fields should never be stored by BCS, they should only be visible within a client (e.g. in a dump log file including node data).
Some parts are separated by a dot. This is true for parts which are only copied from database metadata but which is not used within BroadcastSystem. Currently this affects the “MEDIUM” and the “Clip” parts.
Important hint for JSON: This chapter describes the format, which is used in XML. When JSON is being used (e.g. in BCSService) the node structure and the field names are mostly identical but with the following differences:

- Tagnames, which contain a dot are converted, the dot is converted to 2 underscores. Example: “NPR.Refno” -> “NPR__Refno”. This is done because JSON is generated to be used with JavaScript in the browser and there a tagname with a dot could not be used as an object member like: “object.NPR.Refno”.
- Tagnames, which contain a minus are converted, the minus is converted to 3 underscores. Reason is the same as for the dot.
- The JSON data types number and boolean are used. If a field contains a pure number (e.g. a duration in milliseconds) the field is written as a JSON number. If a field contains a flag (0 or 1 in XML) a boolean value (true/false) is written in JSON.

2.2.1 Metadata of Nodes

Root (and all nodes)	ID	Unique ID of a node. Each node holds a DWORD which is unique in the same level. The whole ID consists of all DWORDS appended, separated by a slash “/” and beginning from the root up to the node of interest. (The root ID is always 00000000)
	GUID	Unique GUID of a node. Each node can hold such a "global unique identifier", which is a 38-character string as it is generated by the Windows tool UUIDGEN.EXE. At the moment the GUID is optional and is only generated for some nodes, esp. elements and groups.
	ImportedNodeID	Some of the nodes can have this field. It represents the full node ID as it was read from the import. It is used for identifying nodes in a re-import. It depends on the import settings whether this field is being generated or not.

Definitions are used to store a list of identifiers which are maintained by the administrator, e.g. the list of all departments.

DefinitionRoot	ID	Always 00000001
	GUID	(see Root)

Section	ID	ID
	GUID	(see Root)
	Name	Name of the section

Definition	ID	ID
	Name	Name of the definition
	GUID	(see Root)
	Color	Three decimal, comma separated RGB values. Used for ressorts only.
	DigaStat_ShowTitle	Internally used from DigaStat, 0 or 1
	DigaStat_ShowAlways	Internally used from DigaStat, 0 or 1

Pools are used as a sort of clipboard, container or personal storage.

PoolRoot	ID	ID (always 00000002)
	GUID	(see Root)
	Path_Dir1	The directory for the media files

GeneralPoolRoot	ID	ID (always 00000001)
	GUID	(see Root)

PersonalPoolRoot	ID	ID
	GUID	(see Root)
	Owner	User name of the owner (8-letter user ID).

Pool	ID	ID
	GUID	(see Root)
	Name	The name of the pool
	Info	Comment
	LastWrite	Date of last update (use: e.g. Auto-Delete functions)
	NoTimeAdjustment	Flag to switch off the BCS time calculation for this pool, 0 or 1

GlobalJingles	ID	ID (always 00000003)
	GUID	(see Root)
	Path_Dir1…n	A list of directories for the media files - only used for global jingles.
	ACL	Access control list. Name of DigaSystem users (and group numbers) which have write access to the jingle group.

Program	ID	ID
	GUID	(see Root)
	Name	Name of the program / service, e.g. "B5"
	Info	Comment
	Path_Dir1…n	A list of directories for the media files
	Path_OnAir	A comma separated list of numbers of the directories which are located on an OnAir computer
	Time_DurationMode	See Time Part.
	PrimaryJingles	0 = use service-specific jingles in playout engine (default) 1 = use global jingles
	MainAudioChannels	The number of available audio channels for playout of main rundown elements. Should be the minimum value which is available in all studios. The same value exists for each show, whereas the show value has priority. This value can be empty.
	TransitionAudioChannels	The number of available channels for playout of transition elements (drop-ins, voice-overs, jingles, overlays,…) which are typically mixed down on a single line/fader. Should be the minimum value which is available in all studios. The same value exists for each show, whereas the show value has priority. This value can be empty.

DayTemplates	ID	ID (always 00000001)
	GUID	(see Root)

DayTemplate	ID	ID
	GUID	(see Root)
	Name	Title
	Weekday	Day of week for automatic use in empty schedules. The date has always priority before weekday definitions Format: english name: Monday, Tuesday, ...
	Date	Exact date for automatic use in format "YYYY-MM-DD"
	FirstValidDate	First valid date if an interval has been defined.
	LastValidDate	Last valid date if an interval has been defined.
	MaxDaysInFuture	Maximum days from <today> the template is valid.

ShowTemplates	ID	ID (always 00000002)
	GUID	(see Root)

PreProductions	ID	ID (always 00000003)
	GUID	(see Root)

PreProduction	ID	ID
	GUID	(see Root)
	Name	Name
	Info	Comment
	NoTimeAdjustment	Flag to switch off the BCS time calculation for this pool, 0 or 1

Jingles	ID	ID (always 00000004)
	GUID	(see Root)

JingleGroup	ID	ID
	GUID	(see Root)
	Name	Name
	Info	Comment
	MaxJingles	Maximum number of Jingles in the group, 0 = Infinite
	AllowedClasses	An enumeration of allowed classes for the block, separated by commas
	ACL	Access control list. Name of DigaSystem users (and group numbers) which have write access to the jingle group.
	MediaDirProfiles	A comma-separated list of names of media directory profiles. These profiles are evaluated by BUS FileCheck task.
	NoTimeAdjustment	Flag to switch off the BCS time calculation for this group, 0 or 1

ConsistencyCheckResults	ID	ID (always 00000005)

Year	ID	ID
	GUID	(see Root)
	Number	Year, i.e. 1999

Month	ID	ID
	GUID	(see Root)
	Number	Month, i.e. 12

Day	ID	ID
	GUID	(see Root)
	Number	Day, i.e. 31
	Weekday	Weekday, english name: Monday, Tuesday, ...
	Time part	This is redundant, but for an easier evaluation of the full date a time part is automatically generated by the server
	ChangedFromTemplate	Flag indicating client changes. If set, the day is no longer identical to the template day originally inserted.

Pool	ID	ID (always 00000001 for day- and showpool)
	GUID	(see Root)
	Name	Name of the pool
	Info	Comment
	NoTimeAdjustment	Flag to switch off the BCS time calculation for this pool, 0 or 1

Trash	ID	ID (always 00000002)
	GUID	(see Root)
	NoTimeAdjustment	Flag to switch off the BCS time calculation for this trash, 0 or 1

Show	ID	ID
	GUID	(see Root)
	ShowID	An additional ID field which can be filled by the customer as desired. The system does not fill this field automatically.
	Name	Title of show
	Info	Comment
	Team part	Part with a list of team-members
	Time part	Part with time information. If the show is part of a template, the dates should be "0000-00-00"
	JingleGroup	Full GUID of the jingle group to use with this show. (This is the old value, the new one is the next value.)
	JingleGroupName	The Name of the jingle group to use with this show.
	FillerGroup	Full GUID of the jingle group containing fillers for the show. (This is the old value, the new one is the next value.)
	FillerGroupName	The name of the jingle group containing fillers for the show.
	PD_REMOTE part	Part with information needed for managing "Remote Shows" (shows stored on other server); this part must be treated as "read only" by all clients!
	ACL	Access control list. Name of DigaSystem users (and group numbers) which have write access to the show
	Path_Dir1	One additional OnAir directory can be specified for each show
	MediaDirProfiles	A comma-separated list of names of media directory profiles. These profiles are evaluated by BUS FileCheck task.
	ChangedFromTemplate	Flag indicating client changes. If set, the show is no longer identical to the template show originally inserted. Possible values: Unchanged=0, Changed=1, Approved=2, Approved + protected=3. The special value SetUnchanged=100 can be used by clients in UpdateNode operations to force the BCS to set the field to Unchanged.
	ShowStatusSetBy	Information about the user who changed the show status (=ChangedFromTemplate) the last time. Format: User\Workstation\Timestamp
	Rebroadcast	A flag (0,1) indicating that the show is a repetition of an elder show.
	ClassFlag	A comma separated list of DigaSystem class names. This field is used for general purpose class flags for the show.
	TeamEWWR	A flag (0,1) indicating if the free list of team members is editable with write right only, or if manage right is necessary.
	ImportFlag	Bitfield with flags used by the import routines internally. Bit 0: “automatic import”, is not used so far. Bit 1: “import protection”, exclude node when running an import.
	MainAudioChannels	See <Program> tagname.
	TransitionAudioChannels	See <Program> tagname.
	ConsistencyCheckHint	This is a free-text field. Any text can be entered. In BUS ConsistencyCheck task it is possible to filter shows to be checked by evaluating this field.

A show below the **ConsistencyCheckResults** has only a small subset of fields:

Show	ID	ID (of the result node itself, not of the examined show)
	NodeID	The full node ID of the show, for which consistency check reports were generated and can be found in the children.
	Name	Title of show
	Time_Start	Start time of show in format YYYY-MM-DD hh:mm:ss.fff
	Time_Stop	Stop time of show in format YYYY-MM-DD hh:mm:ss.fff
	Time_LastCheck	Time when this show was checked the last time in format YYYY-MM-DD hh:mm:ss.fff
	Consistency	One of "GOOD" or "BAD".
	ExecutedCheckTypes	A comma-separated list of the executed check types.

The shows below node <ConsistencyCheckResults> are sorted by start timestamp.

Track	ID	ID
	GUID	(see Root)
	Number	Number of track, 0 exists always and is the main (audio) track with the ID 00000002 The real elements as they were sent are written to a track with number n + 1000 (protocol/logging) Snapshot tracks are in range 2000-2999
	Name	Name of the track, human readable title
	Info	Comment
	Type	Specifies the sort of elements in the track: – Audio: Element for the on air module – RDS: Elements for the radio data system – Internet: Elements for the internet – AudioLog, RDSLog, ... : a corresponding log track At the moment only “Audio” is being used.
	PD_REMOTE part	Part with information needed for managing "Remote Shows" (shows stored on other server); this part must be treated as "read only" by all clients!
	ImportFlag	Bitfield with flags used by the import routines internally. Bit 0: “automatic import”, is not used so far. Bit 1: “import protection”, exclude node when running an import.

A track below a show in the **ConsistencyCheckResults** has only a small subset of fields:

Track	ID	ID
	Number	As above.
	Name	Name of the track, human readable title, only if set
	Consistency	One of "GOOD" or "BAD".

One report about a found problem by **ConsistencyCheck** is reported as an "inconsistency":

Inconsistency	ID	ID (of the result node itself, not of the node with a problem)
	Type	One of: MissingFile, CorruptFile, StopWarning, LoadFailure, StartFailure, Gap, Overlap, OverfilledGroup, UnderfilledGroup, ShowStartExceeded, ShowStopEceeded, MediaDurationMismatch, LoudnessProblem, FreeXMLCheck.
	NodeID	The full node ID of the node, which has the reported problem.
	Title	The title of the node, which has the reported problem.
	Second_ID	Like above, but only filled, if two nodes are part of the reported problem. E.g. for a gap or an overlap.
	Second_Title	
	Time_Start	The time of the problem in format YYYY-MM-DD hh:mm:ss.fff
	Time_Stop	If a problem has a duration, Time_Start gives the start and this time gives the end. Format YYYY-MM-DD hh:mm:ss.fff
	Time_Duration	Duration of the reported problem in milliseconds.
	Time_Found	The time when a check did run and found this problem in format YYYY-MM-DD hh:mm:ss.fff
	Description	A plain text description of the problem.
	Description_DE	The description in German - if it was created. Fallback should be "Description".
	CheckID	An identifier of the BUS task, which has reported the problem.
	Checkset	The name of the checkset, which has reported the problem
	RequestName	If the inconsistency was found by a check, which was externally triggered, this field contains the "request name", which was used for triggering the check.

The inconsistencies below a track are sorted by start timestamp.

All node types within a rundown (Group, Story, EndpointStory, Element) have most of the metadata fields in common. The table for "element" is the most complete one, the table for "group/story" lists only some fields – especially those which exist only for a group or a story or which have a deviation from the fields of an element.

Hint: groups are intended to give rundowns a structure, e.g. by a group for news, then a group for music, then a group for commercials. A story is intended as a mini-rundown, a sequence of text and audio elements belonging together, because these elements handle one common topic. An EndpointStory is intended to collect multiple elements for different endpoints (like TurboPlayer, Twitter, Facebook) and/or different regions. The children are typically started at the same time but afterwards mini-rundowns with multiple elements for one endpoint/region can be constructed.

Group / Story / EndpointStory	ID	ID
	ImportedNodeID	(see Root)
	GUID	(see Root)
	Title	Description
	Info	Comment
	SendState	On Air state: Placeholder, Planned, Cleared, Sending, Sent, Skipped. TurboPlayer does not play group or story nodes themselves, only the child elements. Therefore the send state of groups and stories is ignored and does not matter.
	File part	Groups and stories do not have a primary file, but they could have a presentation (RTF) file. Especially for a story this is not really useful because a story is a sequence of presentation text elements and audio elements.
	Time part	Part with time information. Some attributes are inherited to the children. E.g. a fixed start on time of a group will automatically be inherited to the first element within the group.
	Fade part	Not present for groups / stories.
	Audio part	Not present for groups / stories.
	RegioBlock	0 or 1. Is 1 if the group is within a regio-scheduling track and is linked to a regio element on the main track. Not used for stories or endpoint stories.
	Class	The main class of the group / story.
	AllowedClasses	An enumeration of allowed classes for the group / story, separated by commas
	JingleGroup	Full GUID of the jingle group to use with this group. (This is the old value, the new one is the next value.) Not used for stories.
	JingleGroupName	The name of the jingle group to use with this group. Not used for stories.
	FillerGroupName	The name of the jingle group containing fillers to be used within the group. Not used for stories.
	FillerAlgorithm	Can be "Standard" or "Random". Default is "Standard". Defines, how TurboPlayer fills gaps with fillers.
	Reserve	“No” or “Used”. This flag is copied to the reserve flag of all new group elements. Not used for stories.
	BufferFlag	0 or 1. Is 1 if elements in this group can be shortened by the OnAir module if necessary. Not used for stories.
	DefaultStartMode	The default start mode used for new child elements. Can be empty (start mode and type of new elements are unchanged) or manual or sequenced. If set, floating will be used for the start type. Not used for stories.

Element	ID	ID
	ImportedNodeID	(see Root)
	GUID	(see Root)
	Title	Description
	Info	Comment
	Program	The name of the program(service) the element was intended for.
	ShowName	The name of the show the element was intended for, or in a trash: the show in which the element was deleted.
	Time part	Part with time information
	Source	Source, one of: For internal elements: Harddisk, Digas. For external elements: CD, Microphone, Telephone, Tape, DAT, Minidisk, Satellite, Line
	MixerSource	For non-audio elements the source of the mixer console. Must be one of the names in the “MixerSources” subkey in the Digas registry. If the name is preceded by a '#' it is a group of sources. If the name is a '*' it means: played live without any mixer source.
	Class	For these two fields see the extra description "Class and MediaType" below.
	MediaType	
	FileState	NoFile (Extern), Missing, Invalid, Virtual, Recording, External, Remote, Existing, Accessible, Mirrored, ToCopy, Requested (See chapter about file management for more info)
	RemoteFileState	FileState on the original server for "remote show" elements
	SendState	On Air state: Placeholder, Planned, Cleared, Sending, Sent, Skipped
	DistributionEndpoints	A comma-separated list of endpoints ID’s assigned to the element. Available endpoints are defined in the DigaSystem registry in key Common\DistributionEndPoints\... Only elements with an endpoint, which is intended for playout, will be played by TurboPlayer and will get a playout duration. Other endpoints are intended for additional stuff, like material for social media export. Backward compatibility: If this value is not filled, a TurboPlayer/playout endpoint is assumed automatically. This field is also used for specifying the playout region(s) if you use regionalization.
	FCTS	The "File change timestamp" which is updated whenever a file is created or replaced. This field is needed together with the next one for the file management among multiple remote systems.
	Loc_FCTS	The timestamp of the local files. This value is written by BUS whenever it has synchronized the local files with the remote files.
	TLCI	Text length computation information. This is a private field of BUS if text length computation is active. It should never be used by other clients, though some of the desktop clients can set the field in some operations. The field has the format: <last write timestamp of rtf file>\<used speaker for length computation>. If “*” is specified for the speaker the standard speaker was used.
	Reserve	Specifies if this element is a reserve element. Values are: No, Unused, Used. Default is No.
	Separator	Flag indicating a separator. Used together with the class "Info" to specify the following separators: 0=No separator, 1=Hour separator, 2=Start of import block, 3=End of import block
	ExecutionInMode	Flag for control elements. Specifies in which activation modes of TurboPlayer the element will be played/executed. Possible values are: "Active" (=default if empty), "Passive" and "Active+Passive".
	DBRef	Database back reference "DSN\Table\RefNr". RefNr must be a decimal string.
	IntendedBroadcastDate	This field is filled when the element is coming from a DigaSystem database. It is the translated database field “BROADCASTDATE”. This denotes the broadcast date as it was intended for the database element, but this does not have a deeper meaning within BroadcastSystem.
	DeleteDate	A date (format YYYY-MM-DD) when this element should be deleted from the database. This field is not evaluated within the BroadcastSystem.

	DeleteUser	The user (format Workstation\User) who deleted a given element. This field is only present and used by Day Trash, to be able to identify where exactly the delete action comes from.
	DeleteTimestamp	The timestamp (format YYYY-MM-DD HH:MM:SS.MMM) when the element was deleted. Like the DeleteUser field, this is only used by the Day Trash and represents the moment the deletion operation was processed by BCS.
	Output	Information about the “output channel” = sound processing to use for on-air playing. One of the registry keys below <OnAirApplication>\ChannelsOut
	SendRights	Rights concerning the broadcasting
	Presenter	The presenter
	File part	Part with corresponding media- and textfiles
	Create part	Part with specifications of creation and changes
	Fade part	Part with fading information
	News part	Part with news information
	Music part	Part with music information
	Advert part	Part with commercial information
	Cart part	Part with information of the CartWall
	Audio part	Part with information about the audio file
	Command part	Part with command information
	Control part	Part with control information
	Request part	Part with information about an (archive)-request
	MEDIUM part	Part with info about additional files received from the database.
	Clip part	Part with subclip information received from the database.
	Stinger	Flag indicating a Stinger, 0 or empty = no stinger, 1= default stinger, *registry_key_name*=stinger with special settings defined in the DigaSystem registry
	Dummy	Only used by import modules. 0 or 1. Can be set to indicate dummy elements which are matched against placeholders but skipped afterwards
	OnAirHidden	A flag (0,1) to make the element invisible in on-air applications when a special filter is being set.
	OnAirPlayedBySta	Entered only for elements which have been played by TurboPlayer. Content is the DigaSystem station name.
	OnAirPlayedByStu	Like before. Content is the studio name (if specified).
	OnAirPlayedByUsr	Like before. Content is the DigaSystem user name.
	OnAirPlayedOnAir	Like before. Is a flag (0,1) indicating, whether TurboPlayer was in mode "OnAir" when the element was started.
	MemberOfGroup	Only for elements in a protocol track. The title of the group if the original element was a group member.
	ImportedBy	A string indicating the modules which imported this element, example: “BUS\MuscImport.dll\DigaSQL.dll”
	ImportedAsRemote	A flag used by BUS to mark elements, which were imported and file handling for files on a remote BCS system is requested.
	SID	The "Scheduled ID": the full ID of the corresponding element in the schedule track – only for elements in protocol tracks.
	FillerID	If an element is inserted by an automatic filler routine this is the full ID of the original element within the filler group. Not in use anymore.
	EmbeddedStartID	An ID (typically a GUID) which is used for the embedded starts in TurboPlayer. The ID links subclips of recordings to corresponding live elements in a metadata track.
	PCID	ID to identify the element for the play counter.
	PlayCount	A number specifying how many times the element has been sent. This is meant to be per day and is only found in elements in the day pool.

	ImportFlag	Bitfield with flags used by the import routines internally. Bit 0: “automatic import”, is not used so far. Bit 1: “import protection”, exclude node when running an import.
	VoiceOverFlag	Can be N(o) (= empty), T(ransition) or D(rop-In)
	AutoReplace	A flag (0,1) indicating a placeholder which can be replaced by the BUS auto-replace task.
	BufferFlag	0 or 1. Is 1 if the element can be shortened by the OnAir module.
	RegioBlock	0 or 1. Is 1 if the element is the master element in the main track with dependent regio groups on the regio tracks.
	BeatMarkers	Beat markers in a special format, see “Additional description” below
	Color	Comma separated RGB values, only used for Jingles
	NODELETE	Delete protection flag from the database. Without meaning in the BC system.
	READY	Ready-to-send flag from the database.
	PERFECT	Perfect flag from the database.
	FORARCHIVE	For-archive flag from the database.
	ARCHIVED	Archived flag from the database.
	ROAYLTIES	Royalties flag from the database.
	RoyaltiesText	An additional text describing any royalties
	PERSONALRADIO	Personal-radio flag from the database.
	ASI	A string used by BUS task "additional schedule import" to store, which elements have been treated already.

2.2.2 Parts of Metadata

Fade Part	MarkIn	Offset in milliseconds between file start and real title start
	MarkOut	Time in milliseconds between file start and the point, where the fade out ends
	FadeIn	Duration in milliseconds of the fade-in after mark-in
	TypeIn	Type of fade-in; text parameter for the fading type (like "logarithmic")
	FadeOut	Duration in milliseconds of the fade-out before mark-out
	TypeOut	Type of fade-out; text parameter for the fading type (like "logarithmic")
	Intro	End of intro time (voice-over allowed) in milliseconds after take start
	Intro2	A second intro time sometimes used
	Outro	Start of outro time (voice-over allowed) in milliseconds after take start
	Outro2	A second outro time sometimes used
	LinkIn	Time in milliseconds after mark in. The link-in point is identical to the link-out point of the previous take. (Negative values mean: link-in is before mark-in)
	LinkOut	Time in milliseconds befor mark out. The link-in point of the next take is matched to the link-out point. (Negative values mean: link-out is behind mark-out)
	HookIn	Time in milliseconds after takes start. Used as mark-in for building hooks (teaser, …)
	HookOut	Time in milliseconds after takes start. Used as mark-out for building hooks (teaser, …)
	RampIn	A text describing how the take starts. For the user only.
	RampOut	A text describing how the take ends. For the user only.
	Amplification	Fade level in the middle part of a fade-trapezium. If not present, 0 is used. Format is dBFS as decimal value, like "-6.0".
	AmplificationIn	Fade level before fade-in starts. If not present, minimum possible value is used. Format like Amplification.
	AmplificationOut	Fade level after fade-out ends. If not present, minimum possible value is used. Format like Amplification.
	Gain	An additional gain value which is added to ALL other amplification values or ALL levels of a fadelist, if present. The on-air application sets this value as input gain of the mixer console (if available) or as an extra offset of the MultiPlayer. Format like Amplification.
	DefaultMarkIn DefaultMarkOut DefaultFadeIn DefaultFadeOut DefaultLinkIn DefaultLinkOut	The original values from the database are stored here. The broadcast server uses the default values to generate the real fade values if they are missing and the crossfade editor provides a function which switches back to the default values.
	MarkOutGenerated	A temporary flag indicating that the mark out values were taken unchecked from a data source and must be replaced by the result of a file check.
	List	Contains a fade list, see chapter “Additional description” below.

Time Part	StartType	This field describes how the element is treated during planning. These values are possible: – Unused: Time information is not relevant – Floating: The start of the element is put at the end of the previous element – BackwardFloating: The element is moved until its end is identical to the start of the next element. – Variable: The broadcast server will keep the start time in the range specified by 'FirstStart' and 'LastStart' as first and last possible start time – FixedStart: The start time is fixed to a specific time – FixedEnd: The end time is fixed to a specific time – Fixed: Both, start and end time are fixed – OffsetFloating: The start time is the start of the previous element plus the “StartTimeOffset”. The element floats together with the previous element. – Relative: The start time is defined by the start time of an element in another track. Optionally the field “StartTimeOffset” can be used to define an offset for the start time.
	StartMode	This field describes how the element is started by the OnAir module. The following values are possible: – Unused: Information is not relevant – Manual: Manual start by the presenter – Sequenced: Automatic start after the end of the previous element – StartOnTime: Automatic start if the start time is reached – EndOnTime: (blind start) element is started at the given end time minus the duration of the element. – External: Element, which starts if an extern start trigger (e.g. GPIO) is received. – Relative: automatic start with a “StartTimeOffset” behind the previous element. The MarkIn point of the relative element MUST BE BEHIND the MarkIn point of the previous element! (So far only used with start type “OffsetFloating” and “Relative”) – SyncStart: the element is started synchronized with another element. The other element must have the same sync-start ID. It can be defined, which field contains this ID.
	OnAirStartType	Like start type, but this field is set only and automatically by the broadcast server for elements in groups which require a specific start type, e.g. the first element in a FixedStart-group has also a FixedStart.
	OnAirStartMode	See ‘OnAirStartType’
	Start	Time of start, format "YYYY-MM-DD HH:MM:SS.MMM", if the date is not used it should be "0000-00-00"
	Stop	Time of stop
	FirstStart	First possible start time, only used when 'StartType' is Variable, empty for other types
	LastStart	Last possible start time, only used when 'StartType' is Variable, empty for other types
	StartTimeOffset	Used for the Relative/OffsetFloating start. This is the offset in milliseconds between the two LinkIn points.
	Overlaid	A flag (0/1). If true the element lies over other elements and its duration is ignored in calculation of total show duration. Overlaid elements are typically completely overlapped by other running elements but they can form a mini-rundown for themselves. See the chapter about start attributes for a detailed explanation.
	StartOnTransitionChannel	A flag (0/1) which is set for transition elements (drop-ins, voice-overs, jingles, overlays…) It is a hint for TurboPlayer to prefer starting this element on a special transition channel

		which is normally mixed with all other transition channels onto a single output line.
	DoNotUse	A flag (still to be specified). The on-air application will set it to signal an element cannot be loaded and should not be included in the time calculation because it is not played.
	IIRS	Ignore-if-real-start: If this flag (0 or 1) is 1 and a real start time is present too, the BCS will ignore this element in its time calculation. Used by on-air applications on played elements which are completely overlapped by previously started elements. This flag is deprecated and has been replaced by an extended usage of the Overlaid flag.
	NoStop	A flag (0 or 1) indicating the on-air module to ignore the duration and the mark-out value and to continue after the planned stop (this feature is also called "ignore mark-out").
	MuteStart	A flag (0/1) to start an element muted (minimal fader position). Typically used for a music bed with fixed end.
	Joined	A flag (0/1) telling the on-air application that the element must be started sample accurate behind the previous element (typically for movements of classical music).
	ReferenceNode	For StartType “Relative”: The relative ID of the node (must be in another track under the same show!) which is used as the “reference node” of the “relative” node. The special substring ".." is used to specify <one node up>.
	ReferenceGUID	The GUID of the referenced node (for relative nodes only).
	BacklinkList	A semicolon-separated list of IDs of all nodes with StartType “Relative”, which link to this node.
	Duration	Duration in milliseconds. This duration is calculated between fading points: start is "LinkIn" (or "MarkIn" for the first take), end is either "LinkOut" (or "MarkOut" for the last take). The text duration might be added to the audio duration -> DurationMode.
	TakeDuration	The full duration of the audio file in milliseconds.
	TextDuration	Duration of the corresponding presentation. The audio duration might be added to the text duration -> DurationMode.
	AltDuration	An "alternative duration". This is used for elements with a distribution endpoint, which causes the elements to not be played out on-air. Therefore, their regular duration is empty/zero. This field allows to store a duration for these elements, nevertheless.
	DurationMode	Tells the system how to calculate the total duration. Can be: 1=Only audio, 2=Only text, 3=Audio + Text duration. This value applies for elements and the program(service) as default for elements. For text elements 2 is normally used and the duration of the text is stored in TextDuration.
	FixedDuration	A flag (0 or 1) which specifies whether the duration of the element should be constant, especially for groups.
	Event	The event which starts the element. Used for the external start mode. Here only a key name is specified. Details for the event are listed in the DigaSystem registry in “<NameOfOnAirApplication>\EventsIn\This_Key_Name”
	Warning	Time when the on air module should display a warning if the element was not started (typically for variable start)
	RealStart	Real start time in format “YYYY-MM-DD HH:MM:SS.MMM”. It is entered by the OnAir module after the start of the element. For time calculations the normal time values are ignored if real values are present.
	RealStop	Real stop time in format “YYYY-MM-DD HH:MM:SS.MMM”. It is entered by the OnAir module after the start of the element (which is still a guess) and after the end of the element.
	RealDuration	Real duration in milliseconds. It is entered by the OnAir module after the start of the element (which is still a guess) and after the end of the element.

	RecStart	Time in format “YYYY-MM-DD HH:MM:SS.MMM” when the start of the corresponding element can be found in the recording. Normally identical to <Time_RealStart>.
	RecStop	Time in format “YYYY-MM-DD HH:MM:SS.MMM” when the stop of the corresponding element can be found in the recording. Normally identical to <Time_RealStop>.
	OriginalRecStart	These two fields are used by BUS task “Metadata Scheduler”. They store the original rec values and therefore allow another playout (which overwrites the rec values).
	OriginalRecStop	
	ImpStart	Time in format “YYYY-MM-DD HH:MM:SS.MMM”. When an element was imported from another system the original start timestamp might be kept in this field. This field is not evaluated within BroadcastSystem, but it might be needed in the communication with the other system.
	ImpStop	The original stop time, see <ImpStart>.
	MarkStart	These 3 fields are not present in BCS-XML, they are only computed on-the-fly during the export for convenience. It’s like Start/Stop/Duration (real values are preferred) but the values refer to the mark positions (the original fields refer to link positions).
	MarkStop	
	MarkDuration	
	Scheduled	The date and or time when the element is being scheduled. This is a pure informational field which is not evaluated by the system.
	LastPlayed	Used only within day pools. Specifies the time of the last on-air start per day in format “YYYY-MM-DD HH:MM:SS.MMM”.
	LastFillerUse	Used within filler jingle groups. Specifies the time of the last automatic insertion in format “YYYY-MM-DD HH:MM:SS.MMM”.

Team Part	Presenter	Presenter
	Speaker	Speaker
	SpeakersFactor	The speech velocity factor of the speaker (1.0 - 9.0)
	Director	Director
	Chief	Chief
	ChiefUID	User ID of chief (for automatic evaluation of chief rights)
	Technician	Technican
	Producer	Producer
	Author	Author
	Magaziner	Author of magazine entries (ABY)
	Task n	In case of a free team list the tasks of the members are editable and are stored in this list instead of the normal tags. (n=0,1,2,…)
	Name n	
	Prename n	
	Tel n	

Clip Part	Subclip information received form a database entry is stored here. All subclips are enumerated, starting with 1. The word “Clip” is separated by a dot! These fields are supported:	
	Clip.<n>.Start	The start position of subclip <n> within the take in format HH:MM:SS.mmm
	Clip.<n>.Duration	The duration of subclip <n> in format HH:MM:SS.mmm
	Clip.<n>.Title	The title of subclip <n>.
	Clip.<n>.Type	The type field was intended to differentiate types of subclips. So far this field is not used in most cases. Instead the layer number is mostly used to distinguish different types.
	Clip.<n>.Layer	A number for the “layer”. This is mainly used to have different types of subclips. E.g. all embedded starts have layer 1, all general editor markers have layer 2, …
	Clip.<n>.Comment	The comment of subclip <n>. In many cases this field contains additional metadata for a subclip, packed in JSON or XML.
	Clip.<n>.EmbeddedStartID	If a subclip should be treated as an embedded start by TurboPlayer this field contains a GUID. The corresponding live element on the metadata track must have the same ID in the global field EmbeddedStartID.

File Part	Filename0	Original file from the database (never changed)
	Filename1	Assigned text or media file (a file in the corresponding media directory)
	Filename n	Additional links to mirror files, Important: all filenames from 1 to n are treated equally, no one is preferred
	Presentation0	Original RTF file from the database (never changed)
	Presentation1	Link to RTF-File with additional presentation (a file in the corresponding media directory). If an element is a text element the text is the main media and the links to the text files are stored in “FilenameX” and NOT in “PresentationX”!
	Presentation n	Additional links to mirror files, Important: all filenames from 1 to n are treated equally, no one is preferred
	Filesize n	Size in Bytes of 'Filename n'
	PrimFilename	Optional field. Can be written from clients which set one of the filenames. Denotes the number of the filename written. (All further mirror files are created from BUS)
	PrimPresentation	See above
	OnAirFile	A comma separated list of numbers of the files which are located on an OnAir computer
	OnAirPresentation	The same for the text files (but only for additional presentation text, not if the element is a text element, then OnAirFile is used).
	Converted	A flag (0/1). 1 = files in media directories were converted by BUS and therefore they are different from the original database file no 0
	Prelisten	Low quality audio file for prelistening. For video elements a low-res video file for previewing can be specified.
	Waveform	Waveform file used for displaying or streaming to web clients waveform data for the element.
	Thumbnail n	Link to an image file usable as thumbnail or in a tooltip. n = 1 ... 5 Attention: in contrast to the multiple filenames above, the thumbnails differ in their size. Therefore the size is listed in the data which has the format: "<width>x<height>|<Filename>"

RemoteFile Part	(see **File Part**)	File Part data of the original server for "remote shows"

IIFile Part	(see **File Part**)	Initial inserter file part, data of the original server for "remote shows", only used by a slave BCS internally, when new entries in a remote show / track are inserted.

MEDIUM	…	The medium information from the DigaSystem database is stored in these fields. A hierarchy is created by separating the levels with a point ‘.’ Example: “MEDIUM.FILE1.FILEREF” These fields can occur for nodes Group, Story and Element.

…	…	User definable fields from the DigaSystem database are stored with the same field names as in the database. A hierarchy is created by separating the levels with a point ‘.’ Example: “BBC.ArchiveID” All DB custom fields have two levels exactly.

Create part	Generator	Creating program: scheduler, Advert-dispo, Musicrotation,...
	Creator	Name of user
	CreateTime	Date + time of creation within the database, format "YYYY-MM-DD HH:MM:SS"
	ChangeUser	User who has made the last change in the DATABASE !
	ChangeTime	Date + time of last change within the DATABASE !
	ChangeNumber	Counter, counts number of changes (not used so far)

News Part	Author	Author
	Editor	Editor
	Speaker	The DigAS user which is going to speak the presentation
	Production	Production
	Source	Source of the original entry (not the main field "Source" !)
	Ressort	Subject
	SubRessort	Sub subject
	Department	Department
	FirstWords	FirstWords
	LastWords	LastWords
	Keywords	Keywords
	Language	Language
	Correspondent	Correspondent
	Origin	Origin
	OrderedBy	The person who ordered the entry

Advert Part	Motive	Motive
	ForeignMotive	Any external motive ID
	Customer	Customer
	Productgroup	Productgroup
	Product	Product
	Spotlength	Length of spot in full seconds

Cart Part	Ramp	The ramp time in seconds
	Donut	The donut time in seconds
	Tag	The tag time in seconds
	Priority	The priority of the cart
	Seasonal	A hint in which season the cart should be played
	Weekday	A hint on which weekdays the cart should be played. A bitfield printed as decimal value: Bit 0 = monday, Bit 6 = sunday
	Project	The name of the project
	ExtDevice	A command for controlling an external device. So far this historical field is not evaluated from the BroadcastSystem.

The audio part contains a description of the corresponding audio file. This information is only copied from the database and can be incorrect, especially, because each element can have multiple assigned files with different formats.

Audio Part	Format	A string, with a format description, like: “LINEAR”, "MPEG-LAYER2”, “WMA”, AAC”, “OGG”, “FLAC”, …
	Mode	Can be: Mono, Stereo, Joint Stereo, Dual Channel, Left Channel, Right Channel, 5.1, 5.1 with Stereo, 7.1, 7.1 with Stereo
	Samplerate	The sampling frequency in Hz: 16000, 32000, 44100, 48000
	Bitrate	The total bitrate in KBit/s: 32, 48, ... 384.
	Resolution	Bit depth of each sample: 8, 16, 24.
	Filetype	Can be: MUS, WAV, BWF, DALET, AIFF, IPTC-IIM, ID3V2, MPEG4, RAW
	ILK	Integrated Loudness in LUFS/floating point format with one position after decimal point (e.g. "-23.0").
	LRA	Loudness range in LU/floating point format with one position after decimal point.
	DBTP	Maximum True-Peak Level (Decibel Truepeak) in dBFS/floating point format with one position after decimal point.
	MAXML	Maximum Momentary Loudness Level in LUFS/floating point format with one position after decimal point.
	MAXSL	Maximum Short-term Loudness Level in LUFS/floating point format with one position after decimal point.
	IgnoreLoudness	Flag: 1 or 0. If set to 1 the loudness values are not taken into account when the element is being played or compiled. This is useful for manually leveled elements, which should not be corrected automatically.

Music Part	Performer	Artist
	Singer	Singer
	Soloist	Soloist, principal
	Speaker	Speaker
	Narrator	Narrator
	Instrumentation	Used instruments
	Choir	Choir
	Ensemble	Ensemble
	Orchestra	Orchestra
	Members	Member(s) of the orchestra
	Composer	Composer
	Arranger	Arranger of music
	Texter	Author of lyrics
	Editorial	Editorial
	Conductor	Conductor, director
	Leader	Leader
	Producer	Producer
	Mixer	Mixer
	SoundAssistant	Sound assistant
	SoundEngineer	Sound engineer
	Supervisor	Recording supervisor
	Category	Category
	Type	Type of music
	Tempo	Tempo of music
	Style	Style
	Era	Era
	Intensity	Intensity
	Mood	Mood
	TargetGroup	Target group
	Gender	Gender
	GenericTitle	Another title
	SubTitle	Another title
	SeriesTitle	Another title
	Work	Work (if element is part of a work)
	Movement	Movement (if element is part of a work)
	RecordPlace	Place of record
	RecordDate	Date of record in format "YYYY-MM-DD"
	PublicationYear	Year of publication
	Publisher	Publisher
	Company	Company
	Owner	Owner of the rights
	Label	Label
	LabelCode	Label code
	Album	Album
	CDID	Identification for the CD, typically a number
	CDNumber	Number of CD if element is part of a multi-CD work
	Track	Track of CD
	MusicID	Identification for the title (from Selector, MusicMaster...)
	GemaID	ID for GEMA, ISRC, etc
	Loop	Element can be played in loop mode, 0/1

The request part contains additional information when an element has been "requested", e.g. from an archive system.

Request	OrderID	An ID to identify the element during the request
	Result	The result of the request operation. The following values are defined so far: None (or empty), Success, MissingMedia, NotFound, Error
	Message	A plain text explanation for the result – or any additional information returned by the requested system
	ShortMessage	A short version of the message.
	StartTC	The start of the requested part, format: HH:MM:SS.MMM
	EndTC	The end of the requested part, format: HH:MM:SS.MMM
	Link	An URL where any additional information is available.
	Mail	An e-mail address to which result mails are sent (if this function is provided by the archive module.)

A control element is used to control some sort of device. Multiple commands per element are possible:

Control Part	Operation n (n >= 1)	Subkey in the DigaSystem registry below <NameOfOnAirApplication>\EventsOut. Details are defined in this key.
	Reference n (n >= 1)	Possible values: MarkIn, MarkOut, Stop, LinkIn, LinkOut, Intro, Outro
	Offset n (n >= 1)	Offset in milliseconds. Positive values are behind the reference point, negative in front of them.
	Parameter n (n >= 1)	A string with an event specific parameter string (usually empty)

A command element is used to specify a stage direction.

Command Part	Destination	Target Person
	Command	Command
	Hint	Hint

Remote data is used by BCS to manage links to a remote BCS.

PD_REMOTE Part	CONNTYPE	One of: SP, WS, WSS. This is: SeplProtocol, WebSockets or secure WebSockets.
	SERVERADDR	TCP/IP address or name of the remote server
	SERVERPORT	TCP/IP port of the remote server
	PROGRAM	Program of the show on the remote server
	SHOWID	GUID of the show on the remote server
	TRACKNUMBER	The track number in case of a track link
	LOGIN	User name for login to remote server
	PWD	Encrypted password for login to remote server
	ENFORCEMATCHINGTRACKID	A flag (0,1) indicating that the top level ID must be identical, otherwise a remote link cannot be established.
	RemoteFileAccess	A flag (0,1) indicating that there is a direct access from the remote system to the media files of the main system. Therefore the media files need not / will not be copied to the remote system. Attention: For historical reasons this tag is not a member of the PD_REMOTE part. Therefore the tagname is "RemoteFileAccess" without a PD_REMOTE in front of it.
	STATUS	Status of connection; possible values are: – 0: Connection to the remote server is OK. – 1: Establishing a connection to the remote server failed. – 2: Login at the remote server failed. (Wrong user name and/or password) – 3: No matching node (show or track) on the remote server was found. – 4: A remote connection could not yet be established (a client should never see this value). – 5: The connection was terminated by the remote server. – 6: The referenced "remote server" is identical to the own/calling one. – 7: The ID of the remote track is wrong. – 8: Remote server is busy. Linkup will be attempted again at a later time. – 9: Protocol (SP/WS/WSS) to be used is not supported. – 0xFFFFFFFF: unspecified error

Other private data of BCS

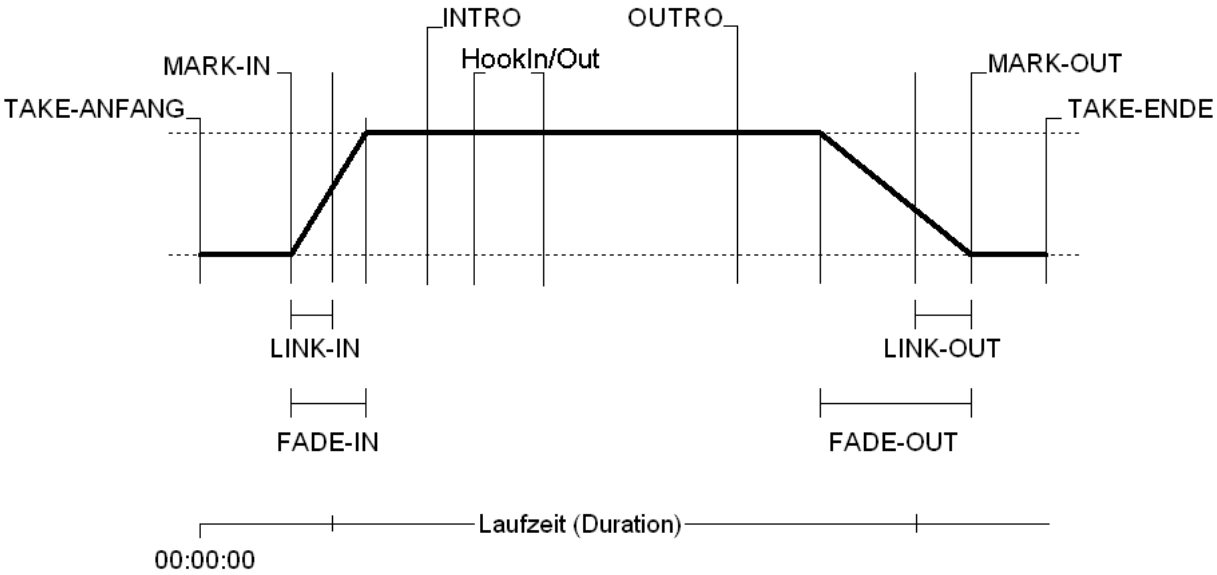
All nodes	PD_LastChangeTime_Hi	These two fields must be combined to an unsigned 64-bit integer. This number represents the timestamp as a Windows FILETIME when the node was changed the last time. The time is the local time of BCS. If a node is changed, this field is set for the node itself and all higher nodes up to the root.
	PD_LastChangeTime_Lo	

	PD_UNT_INFO	This field is not present in normal metadata. It is only set in notifications of type UpdateNodeTree if a client did request this information when requesting notifications with the flag <RequestExtendedTreeInfo>.
		The field has 1 or 3 characters set, with these meanings:
		– Character 1:
		  – O: Node is old.
		  – N: Node is new. In this case no more characters are specified.
		– Character 2:
		  – K: Node kept its position (within the list of children of the current parent).
		  – P: Node has changed its position but has still the same parent.
		– Character 3:
		  – U: Metadata of node is unmodified.
		  – M: Metadata of node is modified.
		  – I: Metadata of node is modified, but only fields, which are not relevant for a fulltext index (typically set for the time computation of nodes in a rundown when another element is modified).

Additional Description

Please note that in chapter 4 about start attributes you can find detailed explanations of some of the metadata fields.

Fade parameters
Here is a picture to make the definitions of the fade parameters more clear:



This picture represents a fade trapezium, which is described by the values seen in the picture. You can also define a fade curve (see above) which replaces some of these values, if present.

2.2.3 Fade list

A fadelist is a list of “samples”: the x value is the playtime in milliseconds, the y value is the attenuation in tenth dB. The first value is always the MarkIn with x=0, the last value is always the MarkOut. X and y values are separated with a colon ‘,’ the sample list is separated with semicolons ‘;’. A pipe symbol ‘|’ is used to separate the fade-in and fade-out part (which is only evaluated by the CrossfadeMixer, the list itself is continuous). Here is an example:

```
<Fade_List>0,-800;1305,-340;5000,0|75333,0;81384,-800</Fade_List>
```

If a fadelist is present, it replaces the ordinary fade values: FadeIn, TypeIn, AmplificationIn, Amplification, FadeOut, TypeOut, AmplificationOut

2.2.4 Beat marker information

Beat markers are stored as a comma-separated list of positions within the audio file in milliseconds. The 0 position is the start of the audio data in the file, **not** the mark-in. This is an example:

```
<BeatMarkers>455,913,1355,1813,2276,2728,3189,3630,4080,… </BeatMarkers>
```

The format can be more complicated because ranges and flags can be used. The flags are separated by a dot "." from the position or the range and optional flag data is separated with a colon ":". Multiple flags can be appended, separated by another dot.

For specifically produced music beds or similar content the distance between beats might be constant. Therefore, an optimization of the required string length is possible: for a range of equidistant beat markers only the first position, the last position and the number of beat markers – including the first and the last one which are given with the 2 positions – can be specified. The number of beat markers is given with the count flag: "CNT:5". A string with a range and 9 beat markers in this range would look like this:

```
<BeatMarkers>0-4000.CNT:9,… </BeatMarkers>
```

So this means beat markers at the positions: 0, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000.

Another flag which can be used is the crossfade point: "CFP:1" or "CFP:0". This flag specifies whether a crossfade to another music element can be done at this beat marker (=1) or not (=0). The default is that a crossfade is possible at every beat marker, therefore the case =1 need not be specified in the string.
Here is a more complex example:

```
<BeatMarkers>0-4000.CNT:9,5000-8000.CNT:4.CFP:0,9000,12000.CFP:0</BeatMarkers>
```

2.2.5 Class and MediaType

In early versions of the BroadcastSystem there was only one field "Class" which mangled two meanings: a classification like music or commercial and the media (or file) type like audio or video. This problem was solved in May 2007 with the introduction of a new field "MediaType" for elements. Due to backward compatibility many possible values appear for both fields now, but the main goal is achieved: you can change the classification without changing the media type.

In the following table the defined media types are listed. The media type can have subtypes which are separated with a point (e.g. Text.Presentation or Video.WithAudio) The subtype is not used so far, but a client which evaluates the media type must be aware of the point and should read till the point only.

MediaType	Comment
Unknown, <empty>	No media type has been set
Audio	An element with an audio file.
Text	An element with an RTF text as primary (!) file. Hint: an additional presentation text can be added for most of the other media types, but this is not meant here.
Graphic	A picture, an image.
Video	An element with a video file.
Live	The element will be played “live”, meaning: the on-air application displays only play time counters but no file is being played and no external device is being controlled.
Line	For content which is played during the show while simultaneously being received as a transmission via a line.
Control	Control elements are used as container for macro commands to be executed while playing. No file is associated with such an element. Control elements are always being played “live”.
Command	Command elements are intended as an instruction for the presenter or the technician. So far this element type is not being used.
Info	Info elements are intended as some additional information for the presenter or the technician. Info elements are not played.
Multiple	Multiple types, e.g. a ZIP or any project

The following classes are defined: (Classes present for backward compatibility are marked as "obsolete".)

Class	Comment	Matched to MediaType (if only an old Class value is present)
None, <empty>	No class has been set	<not matched>
News		Audio
Music		Audio
Commercial		Audio
Cart	A jingle	Audio
Magazine		Audio
Promotion		Audio
Audio	Not specified audio (obsolete)	Audio
Text	A text, typ. a presentation (obsolete)	Text
Graphic	A picture (obsolete)	Graphic
Video	A video (obsolete)	Video
Live	A live element (obsolete)	Live
Line	Played from a line (obsolete)	Line
Control	A control element (obsolete)	Control
Command	A "stage" direction (obsolete)	Command
Info	A pure info without media file	Info
Videoproject		Multiple
Sequoiaproject		Multiple
Videobatch		Multiple
DigaCutMT		Multiple
ZIP		Multiple
Story		<not matched>

For placeholders the media type is not generated from the class. A placeholder can have any media type, including "Unknown". For elements with a media file the media type is automatically derived from the file, the media type cannot be changed by the user. For elements without a media file the user can set the media type as he likes. Other nodes than elements – especially groups - do not have a media type.

2.3 Minimum required fields

In case you have to create new BCS-XML or you have to convert some other format into BCS-XML you do not need to create all the fields as they are specified above. There is a minimum set of fields which is required for a correct handling within BroadcastSystem.

Here is a list of the required fields for the major node types and the major use cases together with some comments:

Show	Show nodes must always be created when generating XML for the import.	
	Name	Every show should have a name. If no name is available, use the start time in format “hh:mm”.
	Time_StartType	Must always be “Fixed”.
	Time_Start	The start timestamp in format “YYYY-MM-DD hh:mm:ss.fff”. Depending on the workflow it might be sufficient to specify only the time and omit the date, but generally the date is required.
	Time_Stop	End of show in same format as the start.
	Time_Duration	A duration in [ms] must always be given and it must fit to the difference of stop and start time of the show.
	ShowID	This field is optional. It might be required if the XML is intended for the import and the importing routine is configured to evaluate the field “ShowID”.

Track	In XML for the import track nodes are optional. The importing routine is capable to create missing track nodes during the import, but this can only be done for a single track – aka a single rundown. If multiple tracks should be imported in a single step it is necessary that track nodes are present in the imported XML.	
	Number	The track number. 0 for the default/main track, 1,…, 999 for any additional track.
	Type	Always set to “Audio”.

Group / Story	Title	Groups and stories should always have a title.
	Class	The class is essential because it is the main criteria for the import routine to distinguish between imported and unaffected stuff.
	AllowedClasses	Without this field it is not possible to insert anything into a group/story because no class is allowed. If you do not have a specific class, generate a comma-separated string with all classes listed above.
	SendState	Set to “Planned” or “Placeholder”.
	Time_StartMode	Set to one of the start modes. If you are not sure use “Manual”.
	Time_StartType	Set to one of the start types. If you are not sure use “Floating”.
	Time_FixedDuration	Optional field for groups. “1” must be specified if a group should have a specific duration because without the flag BCS will compute a new group duration from the content of the group.
	Time_Start	The start is necessary if a start type “FixedStart” is used. The stop is necessary if a start type “FixedEnd” is used. The duration is necessary if the group should have a fixed duration. In principle all 3 time fields should be given, not only a part of them. BCS can correct missing data but the result of the correction might not be as intended.
	Time_Stop	
	Time_Duration	

Element	Title	Elements should always have a title.
	Class	The class is essential because it is the main criteria for the import routine to distinguish between imported and unaffected stuff.
	MediaType	The media type is optional. It will be generated automatically from the class. Nevertheless, you should always specify a correct media type if it is known.
	SendState	All possible send states can be used, but for generating import data “Planned” must be used.
	Time_StartMode	Set to one of the start modes. If you are not sure use “Manual”.
	Time_StartType	Set to one of the start types. If you are not sure use “Floating”.
	Time_Start	

	Time_Stop	The start is necessary if a start type “FixedStart” is used. The stop is necessary if a start type “FixedEnd” is used. Otherwise the time fields can be omitted. In principle all 3 time fields should be given, not only a part of them. BCS can correct missing data but the result of the correction might not be as intended.
	Time_Duration	
	File_Filename0	In case a filename is known it must be given in field Filename0, the FileState must be set to “Existing” and Source must be set to “Harddisk” (or “Digas”).
	FileState	
	Source	
	Fade_MarkIn	Mark values need only be specified together with a file. If you are unsure specify 0 for the mark-in and the duration for mark-out.
	Fade_MarkOut	
	Music_MusicID	If the filename is not known all 3 file fields and the 2 fade fields above must be omitted. The import routine can fill in the missing data by a lookup in the DigaSystem database. For this lookup one of the ID fields can be filled.
	Advert_Motive	
	Music_CDID	
	DBRef	

Here is an example XML file with minimum fields:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Show>
    <Name>Morning show</Name>
    <Time_StartType>Fixed</Time_StartType>
    <Time_Start>2004-05-27 09:00:00.000</Time_Start>
    <Time_Stop>2004-05-27 10:00:00.000</Time_Stop>
    <Time_Duration>3600000</Time_Duration>
    <Element>
        <Title>News intro hot / morning</Title>
        <Class>News</Class>
        <MediaType>Audio</MediaType>
        <SendState>Planned</SendState>
        <FileState>Existing</FileState>
        <File_Filename0>\\AUDIOSERVER\AUDIO\News\DMLNI84.MUS</File_Filename0>
        <Source>Harddisk</Source>
        <Fade_MarkIn>0</Fade_MarkIn>
        <Fade_MarkOut>55745</Fade_MarkOut>
        <Time_StartMode>Manual</Time_StartMode>
        <Time_StartType>Floating</Time_StartType>
        <Time_Start>2004-05-27 11:08:13.130</Time_Start>
        <Time_Stop>2004-05-27 11:09:08.875</Time_Stop>
        <Time_Duration>55745</Time_Duration>
    </Element>
    <Group>
        <Title>News</Title>
        <Class>News</Class>
        <AllowedClasses>News</AllowedClasses>
        <SendState>Placeholder</SendState>
        <Time_StartMode>Manual</Time_StartMode>
        <Time_StartType>Floating</Time_StartType>
        <Time_Start>2004-05-27 10:09:25.570</Time_Start>
        <Time_Stop>2004-05-27 10:09:35.396</Time_Stop>
        <Time_Duration>9826</Time_Duration>
        <Element>
            <Tagnames_as_above>...</Tagnames_as_above>
        </Element>
    </Group>
</Show>
```

A hint: In older version of BCS Tech Manual you could find here a description of the transaction IDs and of generic messages. This information has been moved to BCS Programmers Manual.

3 FILE MANAGEMENT

3.1 Introduction

This chapter describes how media files are handled in the DigaBroadcastSystem. Media files are the main content like audio, bitmaps, video or (presentation-)text. Like in the DigaSystem database this content is not stored in the describing metadata. Instead, only references to the media files are stored in the XML data. If you look at the description of the used XML data structure you will notice that an element can reference multiple files and in addition multiple text files. The Broadcast Server itself does not at all handle these files, it only stores the references. The media files are handled by the clients and by BUS.

3.2 Details

3.2.1 Media directories

In principle, the idea is to assign one or multiple directories to some of the nodes in the tree. The two node types “Program” (=”Service”) and "GlobalJingles" include a list of directories. For node type “GeneralPool” at least one media directory can be defined. The list can be edited with the manage dialogs in DigAIRange. The list is stored within the tree in the path part (Path_Dir1, Path_Dir2...). It is not possible to assign directories for the individual subnodes. Instead, the names of the subdirectories are defined:

- For every day a directory with name "YYYYMMDD"
- For the (service-specific) jingles a directory with name "Jingles"
- For the pre-productions a directory "PreProductions"
- For the templates a directory "Templates"

All needed subdirectories are created automatically.
These “media directories” are used to store files, which are assigned to scheduled entries. This is especially true for files, which are created within the broadcast system, e.g. new text or edited audio. Besides, the directories are used for mirror files, which are created for safety reasons. At least one of the media directories must be specified (Path_Dir1). All further directories are optional (Path_Dir2, ...).

3.2.2 Additional show media directory

It is possible to specify one additional media directory in the metadata of each show. This can be useful if you have many on-air studios and each show is being sent from a different studio. In this case the show directory can point to a computer in the studio and background tasks in BUS can copy all media files onto this computer so that the files are available directly in the studio. Other studios are not burdened with these files.

Important hint: this feature was not present from the beginning. Therefore, the support of the clients is not yet complete. If an existing file is located in a show directory all modules can handle it. But if a directory for a new file must be found, only these modules can take the show directory into account:

- BUS since build 208 of 2004-02-26
- DigAIRange since versions 4.9.593.0 and 4.8.580.0 of 2015-07-09

3.2.3 Media directory profiles

Since Release 5.5 BUS supports media directory profiles. These directories are only known to BUS when creating new copies of files. All other clients cannot use them for creating files, but if a file in such a directory has been entered to the metadata, the clients can access the file for reading. Media directory profiles extend the idea of the additional show media directory, which is limited to a single directory only. With the profiles it is possible to define multiple ones and to assign one or multiple profiles to each show. BUS will then take care that all these directories are filled with files. For more information see chapter 7.3 File Check Task of BUS below.

3.2.4 Filenames

The filename for media files is generated automatically with a continuous counter for each media directory. The counter is a simple 4 Byte (DWORD) long file with name “Number.cnt” which is read and incremented in exclusive file access. The counter contains the last number which was used by a client for the creation of a media filename. Therefore, a client which needs a new number must first increment the counter of the file and then use the new number. The counter value is printed in a numerical system with 32 numbers: 0-9, A-V (DigaSystem standard). A file which was initially created in a BCS media directory has only the running number as the name. A file which was copied from a different place into a media directory keeps the original filename (as sort of “backlink”) and the running number is appended with an underscore in between.

A special hint for the users of Windows 2000 or newer: If you use UNC names for your files you will have a problem to access the files if the network connection is interrupted. This is of course true for the files on the network. But worse, even files which are located on the local harddisks are no longer available. This is especially bad for the OnAir computer if the files to be played are kept in local directories. The solution is to install the “Microsoft Loopback Virtual Network Adapter”. It appears as a second network adapter and allows resolving UNC names without network.

3.2.5 FileState

The field *<FileState>* is used to specify the state of the corresponding file for each entry. The following values are defined:

- **NoFile**: No file has been assigned to the entry (Info, live element)
- **Missing**: A file has been specified but it cannot be found
- **Invalid**: A file has been specified but the type/content of the file is invalid (e.g. no audio)
- **Remote**: One ore multiple files exist on the remotely linked system, but no files do exist locally.
- **Virtual**: (Announcement) The filename must be known already. BUS will check the file periodically and will correct the state when the file is increasing or stops increasing. Together with newer versions of DBM there is a different workflow: the prelisten/Filename2 field will hold a reference to an empty text file (.TXT). The text file will be the filled with the filename of the real media file (and additional information) as soon as the media file does exist.
- **Recording**: Entry is just in record. This state is assigned by BUS while the file is monitored periodically, and the file size increases from check to check.
- **External**: Either the media data is available on an external medium (e.g. a CD) or there could be a file available for this entry but first it has to be transferred to the DigaBroadcastSystem.
- **Existing**: The file is existing somewhere in the network, but it has not been checked whether the content is valid and whether it can be opened/played by the OnAir system.
- **Accessible**: The file is existing. In addition, its content has been verified and it is in one of the allowed directories (which can be accessed from the OnAir system). Possibly, the file had to be copied or converted to achieve this state.
- **Mirrored**: In addition to the previous state all desired mirror files have been created.
- **ToCopy**: This is a special state which is set by DigAIRange sometimes if an entry is copied within the schedule (e.g. from one day to another one). The state is used to signal BUS the necessity to copy this file. Otherwise, it is treated as “Existing”.
- **Requested**: This is a special state for elements which are "External" (they are treated like external) but which will be available soon, e.g. because a de-archiving has been initiated.

Remark: Checking of files and audio conversion is done by BUS mainly. A client, which inserts a new entry in the schedule simply checks for the existence of the corresponding files and assigns the state “Existing”. Only BUS will increase the state to “Accessible” after successful checking / copying / converting. Or it can assign an “Invalid”, if necessary. Whether an audio conversion is performed, is defined by a prescribed audio format per service for BUS. Exception: if no BUS is available in the system the client (typ. DigAIRange) can set the state to “Accessible”, but this does not mean it will check the content ! Nevertheless, there are some cases in which the normal client has to create file copies in the media directories too, see the next sections.

3.2.6 File reading

If a client wants to read a file, it reads the list of specified filenames first and tries to open one of the files. This is done with the help of a non-blocking file access module to avoid hanging network calls. The module can check the existence of multiple files at the same time. The decision to pick one of the specified files should be done in 3 steps:

1) The client tries to open the file which is declared as primary file.
2) If the primary file is not available a simultaneous check is done for all other specified files except the primary one and file 0.
3) If none of the files is available a fallback to the database file 0 is done.

The file with number 1 or higher files are always preferred against the File 0 because they are the (possibly) edited versions whereas the File 0 is the original database file, which is only used if no other file is available. Filename0 is dropped if the file content differs from the other files (e.g. because the file was converted or edited). This is necessary to support the "Switch between files during playout" feature of MultiPlayer. If the BCS files were converted and are no long identical to the file 0, the flag "File_Converted" is set to 1. Besides, some files can be marked as "OnAir" files. They should be treated properly by each client, see below. For prelistening or previewing there might be an extra field <*File_Prelisten*> which can be used by a client.

3.2.7 OnAir files

It is possible to put a mirror directory on the on-air computer (the one which plays the audio files). This is done to have the media files on the local harddisk of the player. Such a directory must be visible from the network because BUS will copy the media files to it. The supervisor has to assign the special on-air attribute to these mirror directories which is stored in the program node in the <*Path_OnAir*> tag. In addition, the file part for each element contains two tags <*File_OnAirFile*> and <*File_OnAirPresentation*> which lists all files (presentations) which are located in an on-air directory. These files / directories are treated differently:

- On-air clients prefer on-air files when playing elements.
- Normal clients prefer non on-air files when playing elements.
- BUS uses special pause values for accessing on-air directories to keep the load small.
- The delete task of BUS can delete on-air day directories earlier than normal day directories.

The additional media directory which can be assigned to a show is treated as an on-air directory, too.

3.2.8 File generation

If an entry has an assigned RTF text (class "text" or the additional presentation) a copy of the RTF file is always created in one of the media directories. This is normally done by the inserting client (e.g. DigAIRange or the importer modules) when the entry is inserted into the tree, or latest when a BroadcastSystem client saves a text. Changes by editing a text have only an effect on the file copies assigned to a single tree element. The original database entry or other copies of the entry in the tree are not affected.

When an entry is inserted into the tree, audio files are not copied to the media directories at first. Only a link to the original file is saved (*File_Filename0*). The Broadcast Utility Server will create one or multiple copies, depending on the settings, and will list each filename in the tree (*File_Filename1*, …). Within BUS you can define a list of allowed directories for each service. They are one of the criteria to decide whether files are copied into the media directories. There is one exception for a client creating new entries: If a file is located in a path on a local disk the client which creates the entry has to make the first copy, because BUS will probably not be able to access the local file.

For DigAIRange there are parameters, which define the copy behavior: in the key "DigAIRange \ FileCopy" four values exist for days, templates, jingles and pre-productions. Here you can define e.g. that DigAIRange must make a copy of the media files if someone inserts an entry very far into the future. Problem is that BUS checks only a few hours or days into the future and if the destination show is not yet checked, the files remain uncopied for a long time. In this time someone could delete the referenced database entry and the files would be lost. Therefore, it is necessary that DigAIRange makes the file copy when inserting the entry.

3.2.9 File writing

We need to distinguish audio/video/image files on one side and text files on the other side. There are two reasons. First one is that audio/video/image editing is always non-destructive with new file creation, whereas text editing modifies existing files and has special features like locking files by holding them open and a content synchronization during editing. Second reason is that for text files different tagnames must be written. If an element has media type "Text" the main file is the text file and the fields to be written are <*File_Filename*…>. If the media type is anything else the text can only be an additional presentation text for the element. In this case the fields to be written are <*File_Presentation*…>

One step during file writing which is always required is the selection of the used media directory. If a client wants to write a new file, it checks the specified media directories, which of them is available and uses one of them to store the file. It can prefer directories with lower number but it does not have to do this. Media directories which are tagged as "OnAir" should not be used if a non-OnAir directory is available. The client stores the filename in the tag <*File_FilenameX*> (or …Presentation…, X can be chosen free but normally 1 is used). Besides the client stores a hint which number is the primary file by writing the tag <*File_PrimFilename*>=X. All other references to mirrored files – including remote filenames – are removed. Only the Filename0 which points to the database file is kept. Additional data must be written, see below. Afterwards BUS will create mirror files of this primary file and store links to them in the other tags <*File_FilenameY*>. BUS can also take care of remote files, format conversion, creation of prelisten and waveform files and it will set the file state.

Metadata fields written by a client:

- <*File_Filename1*> points to the new primary file. If a file from a DigaSystem database is entered the field <*File_Filename0*> is used instead.
- If present, all other Filenames 1-n are removed.
- All remote filenames 1-n are removed.
- <*File_PrimFilename*> is set to the number of the primary file (in this example 1).
- <*File_OnAirFile*> is set accordingly.
- <*File_Filesize1*> is set accordingly. If present, other file size fields are removed.
- <*MediaType*> is set according to the main referenced file.
- <*FileState*> is set to "Existing". If a file is known to be invalid, state "Invalid" is set.
- If <*RemoteFileState*> is "Mirrored" it is set back to "Existing", too. This signals the remote system that a new file replication must be performed.
- <*Source*> is set to "Harddisk". "Digas" is used if a Filename0 to a file in a DigaSystem database table is available.
- <*File_Converted*> must be set to 1 if the client enters a new primary BCS filename in filename1 and at the same time an original database file referenced in filename0 when the content of these files is not identical.
- The duration is set. This includes <*Time_TakeDuration*>, <*Fade_MarkIn*>, <*Fade_MarkOut*>, <*Fade_DefaultMarkIn*>, <*Fade_DefaultMarkOut*>.
- If known, format describing metadata like the media type, audio format or the bitrate are set, otherwise these fields are emptied.
- If present, references to the prelisten and the waveform file are removed from the metadata.
- If present, loudness metadata is either being corrected or otherwise removed from the metadata.
- Field <*FCTS*> is set to the time when the file was written. If an older file replaces a newer file, the time in FCTS must get an exclamation mark "!" appended. This forces a remote synchronization, though the file change time does not become newer.

Special rules for text:

- For additional presentation text the tags <*…Filename…*> are not used, instead <*…Presentation…*> is used. Most other fields are not written (e.g. <*FileState*>) because they refer to the main file, not the additional one. Only <*FileState*>, <*RemoteFileState*> and <*FCTS*> are set in order to re-trigger file replication by BUS.
- If a text file is opened for editing and should be held open the primary file should always be used, because otherwise different clients might have different files open and might think they are the only one holding it open.
- A LOI file (lock information) should be written as long as a text file is kept open. This is a text file in Windows INI format for fields: User, Computer, Date and Time. This file is used to display information about the locking user/computer if another use wants to access the file, too.
- Flag <*File_TemporaryText*> is set if the file for a text element was written only temporary – meaning: not triggered by a user-save, but a background writing to make other clients update the content if they display the same text.
- The new duration is computed, taking the correct speaker and its speed into account. This duration is set in field <*Time_TextDuration*> (for text elements and for additional presentation text).
- The field <TLCI> (text length computation info) should be set accordingly.

- The fields <*Time_DurationMode*> and <*News_Speaker*> might be set accordingly.

3.2.10 Remote shows / tracks

It is possible to link a show or a track to a remote server, which means that the local BCS connects to the remote server and keeps the local metadata (nearly) identical on both systems. The metadata is completely identical if you set a flag (in the link properties) which says that the local system has full file access to the remote system. In this case only the metadata is mirrored. If the flag is not set, the metadata and the media files are mirrored. Therefore, additional metadata on the local system is necessary for the management of the local files. This metadata is not reflected back to the main/remote system. In principle, the file describing fields are doubled. The normal part (*File_*..., *FileState*) always denotes the properties for the local system. An additional part (*RemoteFile_*...) and the additional field (*RemoteFileState*) specify the properties for the remote system. The remote fields are evaluated only by BUS when copying, the other clients use the regular fields. The remote fields are generated automatically by the BroadcastServer when it mirrors the show data. The clients have to obey a sophisticated logic when writing file data to achieve a correct functionality with remote links. (See chapter 11 about remoting for more information.)

3.2.11 Deleting files

Deleting of media files is done only by BUS. All day subdirectories are deleted completely after an adjustable period of time. Sometime before, media files can be deleted while keeping the presentation files. Regular clients never delete a file ! With this solution it is accepted that individual dead files exist until the directory itself is deleted or at least cleaned up. For non-day nodes BUS checks once a day all files in the directories. If there are media files, which are not listed in the corresponding tree node, they are deleted.

4 START ATTRIBUTES

4.1 Introduction

In DigaBroadcastSystem there are start attributes. They are divided into "StartType", "StartMode" and some additional helper fields and parameters. To keep type and mode apart please remember one important thing: **The StartType defines the behaviour of the system during scheduling, the StartMode defines the behaviour of the on-air client during playout.** During scheduling it is the responsibility of BroadcastServer to do all time computations; on-air playout is done by TurboPlayer.

4.2 Details

4.2.1 StartType

The following start types are defined:

StartType	Symbol	Description
Floating		The start time of the element is dynamically adjusted to the previous element. Easily spoken: the start time is identical to the stop time of the previous element. More detailed: the fade parameters are taken into account, which leads to a situation in which the link-out point of the previous element is identical to the link-in point of the floating element. If there is no previous element in the show, the start time of the show is taken.
Variable		The start time of the element is kept within the interval [FirstStartTime, LastStartTime]. The two times can be adjusted by the user for the element. Within the interval the element is floating but at the border times it becomes begin- or end-fixed.
BeginFixed		The start time of the element is fixed to a specific time. As a result, there might be gaps or overlaps to the previous or the next element.
EndFixed		The end time of the element is fixed to a specific time. As a result, there might be gaps or overlaps to the previous or the next element.
Fixed (both)		This is a type, which is only present for compatibility reasons. Normally you should only use BeginFixed or EndFixed.
BackwardFloating		Like Floating but with reverse direction: the end time of the element is made identical to the start time of the next element (again more detailed: the link points are made identical). The last element in such a backward floating chain must always be an element which is EndFixed.
OffsetFloating		These are two special types, which are only allowed in conjunction with a relative start mode.
Relative		The first one is used for elements, which can float in time but should be started relative to another element taking an offset into account (e.g. a short drop-in during the first seconds of a long music element). This always goes in combination with the Overlaid flag being set and the VoiceOverFlag being D(ropIn). The second one is used for relative starts across multiple tracks (e.g. simultaneous start of a main audio element and multiple assigned elements for different on-air regions).

4.2.2 StartMode

The following start modes are defined:
The exact behaviour depends on the current activation mode of TurboPlayer (Passive, LiveAssist or Automatic) and on some other metadata fields and registry parameters.

StartMode	Symbol	Activation mode	Description
Manual		LiveAssist:	Element has to be be started manually by user.
			If the StartType is <BackwardFloating> and the parameter "BWFLiveAutoStart" is on the element is treated as <StartOnTime>.
		Automatic:	In combination with <Floating> the element is started automatically like being <Sequenced>.
			In combination with any of the fixed start types or <BackwardFloating> the element is treated as <StartOnTime>.
			In combination with <Variable> the behaviour becomes more complex: when the first time is reached TurboPlayer will skip all direct previous elements which have not been played so far and will set the start mode to <Sequenced>. During the time interval a normal sequenced start is possible. If the end of the interval is reached and the element is not yet running, one of 3 possible actions can happen: start, skip, none -> see below "ActionOnVariableEnd".
Sequenced		LiveAssist:	The element is started at the end of the previous element. Fade parameters are taken into account: the link-in point of the element is made identical to the link-out point of the previous element.
			If the StartType is <BackwardFloating> and the parameter "BWFLiveAutoStart" is on, the element is treated as <StartOnTime>.
			If the overlaid flag is set, the element is started behind the previous overlaid element. This allows constructing sequences of overlaid elements (e.g. small drop-ins while a long element is running). In case of the error situation that there is no previous overlaid element, this combination is treated like a relative start with offset 0.
		Automatic:	Like in LiveAssist.
			Exception: in combination with StartType <BackwardFloating> the element is treated as <StartOnTime>.
StartOnTime		LiveAssist and Automatic:	Element is started automatically at scheduled start time.
		Passive:	Element is not started automatically. Exception: control elements can be started automatically, if the field "ExecutionInMode" is set to "Passive" or "Active+Passive".

EndOnTime		LiveAssist and Automatic:	Element is started automatically. The start time is computed to meet the desired stop time if the element plays exactly for the planned duration.
		Passive:	Element is not started automatically. Exception: control elements can be started automatically, if the field "ExecutionInMode" is set to "Passive" or "Active+Passive".
External		LiveAssist and Automatic:	Element is started when an external trigger is received. Typically, the trigger is a signal on a GPIO input line.
			If the StartType if <Floating> the element can only be started, if it is the next element.
			For all fixed elements, the registry parameter "ExternalLookAhead" time specifies at which time before the scheduled start time the element becomes "hot" and can be started by an external event.
			If the StartType is <Variable> the element can only be started within the start interval.
		Passive:	Element is not started automatically. Exception: control elements can be started automatically, if the field "ExecutionInMode" is set to "Passive" or "Active+Passive".
Relative		LiveAssist and Automatic:	The start depends on the start of another element: both starts should be simultaneously or with an additional offset – if desired. The master start can be of any start-type/-mode, the relative/slave start will be automatic. A relative element depending on an element of the same track will typically have the attribute combination: Relative, OffsetFloating, Overlaid, Dropin.
SyncStart		Floating	The start depends on the start of another element: both starts should be executed simultaneously. In contrast to the Relative start mode the elements are not linked by a relative node ID. They start together if they have the same content in the SyncStartID. This is one arbitrary metadata field. You can configure in the registry value BroadcastSystem\SyncStartField which field this should be. You can use any existing field or a custom field or define a new field, like “SyncStartD” or “SSID”. So far, the start type of a SyncStart element must always be “Floating”. For elements in an EndpointStory this is enforced by BCS. Hint: TurboPlayer could perform a synced-start before this start mode was introduced. You can use the parameter TurboPlayer\SyncPlayOptions to configure, whether this historical feature still works or whether TurboPlayer should only start elements with a start mode SyncStart in a synchronized way.

4.2.3 Other fields and parameters

There are some additional metadata fields or registry parameters, which are relevant for starting elements. Here is a short description: (The registry parameters are not described in detail, please look into the current parameter documentation "DigParam.rtf".)

AbsoluteStartIsStinger, StingerFadesEverything (TurboPlayer registry parameter)

With these parameters, you can define the behaviour when a stinger is being started. (Stinger: see below)

ActionOnVariableEnd (TurboPlayer registry parameter)

This parameter tells TurboPlayer what to do when the end of a variable start interval is reached and an element is not yet running. There are three possible actions:

- Start: The element is started.
- Skip: The element is skipped.
- None: The element is set to floating but it is neither started nor skipped.

BeatExact... (TurboPlayer registry parameter)

There are multiple TurboPlayer parameters to define the details of a beat-exact start. See the extra chapter in TurboPlayerTechManual for more information.

BWFLiveAutoStart (TurboPlayer registry parameter)

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.

EmbeddedStartID (metadata field)

This field is normally assigned by BUS task “Metadata scheduler”. It links a certain subclip of a recording to one of the live elements on a metadata track so that the live element can be started automatically by TurboPlayer if the playout position of the recording reaches the position of the subclip. For more information, see chapter “Reusing broadcast content / Metadata track” below.

FixedDuration (metadata field/flag)

This flag can be set for a group. Normally the duration of a group is computed as the sum of the durations of all elements in the group. But if this flag is being set, the duration is kept at the specified value. Therefore, a gap or overlap can result at the end of the group.

IgnoreMarkOut (metadata field "**NoStop**")

This flag indicates that the mark-out of an element should be ignored. There are multiple TurboPlayer parameters which define the exact behaviour: "IgnoreMarkOutClasses", "IgnoreMarkOutMediaTypes", "SequencedAtIgnoredMarkOut"

.

Joined (metadata field/flag)

Can only be set for sequenced elements and tells TurboPlayer that the element must be started sample accurate behind the previous element (typically for movements of classical music).

MuteStart (metadata field/flag)

If set TurboPlayer will start the element muted (with minimal fader position). The TurboPlayer parameter "MuteStartSkipsPrev" defines whether previous elements are skipped during start or not.

Overlaid (metadata field/flag)

A flag internally set when an element is completely overlapped by a different element (e.g. a short drop-in being overlapped by a long music element) and should be ignored in time calculation of normal (=non-overlaid) elements. Typically, the first overlaid element has the attributes: StartType=OffsetFloating, StartMode=Relative, VoiceOverFlag=D(ropIn), Overlaid=1. It is possible to construct a sequence of overlaid elements, for which the next elements have the attributes: StartType=Floating, StartMode=Sequenced or Manual, VoiceOverFlag=D(ropIn), Overlaid=1. It is also possible to have multiple relative elements below a normal element. This allows to adjust any possible delay between overlaid elements. In any case, the overlaid elements must lie on the same tree level and must directly follow their main/master element.

SequencedAcrossPlayed (Deprecated TurboPlayer registry parameter)

This parameter is outdated. Since builds BCS: 254.0, DigAIRange: 500.0, TurboPlayer: 1300.0 it is possible to construct a sub-sequence of overlaid elements (see above). This makes the behaviour, which is controlled by this parameter obsolete.
Old description: Normally the mode <Sequenced> means: start behind the *previous* element. In some cases it might be necessary to allow TurboPlayer to refer such a sequenced chain on *one of the previous* elements while there are already played elements in between. This can be achieved with this parameter.

SetNextCursorToOverlaid (TurboPlayer registry parameter)

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.

StartOnTransitionChannel (metadata field/flag)

A flag (0/1) which is set for transition elements (drop-ins, voice-overs, jingles, overlays…) It is a hint for TurboPlayer to prefer starting this element on a special transition channel which is normally mixed with all other transition channels onto a single output line. If the flag is not set, TurboPlayer will (depending on the other settings) prefer to start an element on a channel, which has a 1:1 assignment with a line/fader because this allows the user to control each element with a fader. This flag can be set for a bunch of elements, which are all seen as belonging to one transition.

StartStopAutomatic, StartStopLiveAssist, StartStopPassive (TurboPlayer registry parameter)

These parameters define which type of start is allowed in the corresponding activation mode (user input, events, timer start, etc.)

StartTimeOffset (metadata field)

The offset when the start of an element triggers the start of a second element. Used for relative start mode/type.

Stinger (metadata field)

When a stinger starts all other running elements are faded out and stopped. The field contains the stinger type, which defines the fade-out time. With the TurboPlayer parameters "AbsoluteStartIsStinger" and "StingerFadesEverything" you can specify whether all absolute (=time-) starts should be treated as a stinger and whether only elements on the same rundown should be faded out.

SyncStartField (DigaBroadcastSystem registry parameter)

To start multiple elements simultaneously you can use relative starts. Unfortunately, they require a link to the master element which must be set – normally this has to be done manually. A synchronous start can also be achieved if there is a metadata field with identical content for all elements to be started. This is the SyncStartField. It can be any arbitrary existing regular or custom field or you can define a new one. The field, which should act as this ID can be configured in parameter: BroadcastSystem\SyncStartField.

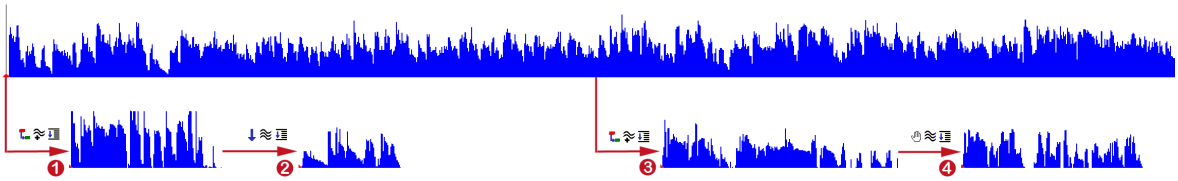
VoiceOverFlag (metadata field)

This is a helper flag which can have the values:

- N(o) or empty: flag is not set. This is the case for main rundown elements.
- T(ransition): element is a non-overlaid sequenced element which was put in between of two main rundown elements, typically a spoken voice-over between two music elements.
- D(ropIn): element is a short overlaid element with relative start lying below a main rundown element, typically drop-ins like station jingles.

4.2.4 Usage of overlaid flag

In the chapter above you can find a textual description of the overlaid flag. Here you can see a little graphic to illustrate how the overlaid flag is intended to be used:



The overlaid flag must be set for elements, which are overlapped by other elements. In the picture, you can see a long main element on the top (e.g. a music bed) and 4 overlaid elements below. TurboPlayer will play the overlaid elements while the long element is playing, too.

You can use start attributes to construct chains of overlaid elements. Not all of the available start attributes are allowed in this context, but these combinations can be used:

1) The first overlaid element should always have the start attributes:
   - Mode: Relative
   - Type: OffsetFloating
   - Flag: Overlaid

   This means the element will be started together with the main element. If the start should be delayed, the Offset field (Time_StartTimeOffset) can be used to specify the delay between the start of the main element and the start of the overlaid element. (Hint: no, negative offsets are not possible.)

   An overlaid element must be a successor of the master / main rundown element. It must lie on the same tree level as the main element. There must be no unplayable element (e.g. info) in between.

2) The next element can be a sequenced element which starts behind the first element. It must have the start attributes:
   - Mode: Sequenced
   - Type: Floating
   - Flag: Overlaid

   The start is handled by TurboPlayer like a sequenced transition between normal elements. The fade attributes are evaluated, too. Therefore, it is possible to make crossfades between the two elements, to create gaps between them or to use a fade curve.

   You can put as many sequenced elements, as you like behind the first overlaid element. Nevertheless, if the scheduled start of an overlaid element lies behind the stop of the corresponding main element, the overlaid element will never be started but instead be skipped when the next main element starts.

   A sequenced overlaid element must be the direct successor of the previous overlaid element. It is not allowed to have both elements on different levels in the tree, to have different parent nodes for both elements or to have any unplayable elements (e.g. info) in between of the two elements.

3) The third Element in the graphic above is a relative-start element like element 1. This is allowed and it is useful to start an overlaid element at a specific position within the main element. In principle this could also be achieved with a negative link-in to the previous overlaid element, but this is not very intuitive and could easily be destroyed if a chain of multiple elements is involved.

   Usually you will have a start offset which is different from the offset for element 1 and element 3 will start after the stop of element 1. Nevertheless, you can achieve that multiple overlaid elements play at the same time. This is no problem as long as TurboPlayer has enough channels to play all desired elements at the same time.

4) The last element in the graphic shows the usage of the start mode "Manual". The other start attributes are identical to the sequenced element 2.

   Such an element will not be started automatically by TurboPlayer (unless you are in automatic mode) which allows the user to start the element whenever he likes. Some settings have an influence on how the user can select these elements and start them. Like for normal elements the rundown mode determines whether the elements must be played in the given order (RundownMode=Sequenced) or whether they can be selected and played arbitrarily (RundownMode=MultiSequenced or Once). TurboPlayer will put the <next> cursor on overlaid elements only if the parameter TurboPlayer\ SetNextCurosrToOverlaid is active (which is the default). Otherwise, the user can only start normal elements.

Some additional words regarding the scheduling with CrossfadeMixer: CrossfadeMixer is not capable of handling multiple scheduled overlaid elements. If you put multiple elements on the third or fourth track, CrossfadeMixer will compile these elements into a single element. Only the compilation appears within the rundown. It is not intended to change this behaviour as CrossfadeMixer is deprecated. The new OnAIR-TrackMixer handles all the attributes automatically.

Nevertheless, you can use CrossfadeMixer to schedule all of the transitions mentioned above. To edit a relative-start overlaid element fill the CrossfadeMixer by dragging the overlaid element onto the third track. For the first overlaid element it is also possible to drag the main element onto the second track. To edit the sequenced transition between two overlaid elements, drag the first overlaid element of the transition onto the first track, or the second element onto the second track. In all cases you are able to edit exactly the start attributes of *one* overlaid element but you will not see or hear the full chain of all overlaid elements.

5 IMPORT / EXPORT

5.1 The Import Function in DigAIRange and BUS

This chapter describes how importing schedules with DigAIRange or BUS from other systems works and explains the settings. The basic idea is that the BCS schedule is the central list, which is filled from different sources. Of course, this list can be filled manually with entries from a database. But here we want to focus on the import function which can read lists from other systems and merge them with the BCS list. To be able to read different formats the function uses import DLLs.
Very important hint: For the semi and full-automatic import it is absolutely essential that the structure of the imported list and the destination show do fit. Of course, it would be ideal if they are identical, but in reality they will always differ. Rule: If both structures are more similar the import result will be better.

5.1.1 Manual, semi-automatic and full-automatic import

In DigaSystem you can start an import in three different ways. The simplest is the manual import which is done with DigAIRange. If you open one of the import keys the list of import shows (or hours) is displayed in the tree below the key. Possibly the DLL is configured to query missing information, so a dialog box appears to ask you about the date e.g. you can drag these import shows into the upper or lower list view. After importing the data, the list view displays the schedule. You cannot change it but you can drag individual elements into the other list view. In this mode the user is responsible for his actions, the merging process as it is described below is not used. This mode is useful if only a single (or some) elements have changed and must be re-imported. Or it can be used to look at the original imported data before it is merged to the BCS list.

The semi-automatic import is done with DigAIRange, too. Instead of dragging an import item into a list view you drop it in the tree view onto a show. DigAIRange will ask whether to merge both lists or replace the destination list. If you select replace, all existing elements at the destination are lost. This is only useful if you have an import source which yields a complete list. But normally each import source is responsible for a single class only, e.g. music or commercials. Therefore, the normal operation is to merge both lists. It has some advantages, depending on the merge mode used: the structure of the destination is kept, elements, which do not have the primary imported class are not touched and the import can be repeated as many times as you like. The merging is described in detail later.

The full-automatic import uses exactly the same merging function as the semi-automatic import. But the user does not have to trigger the process, it runs automatically. Therefore, it is part of BUS. When you configure an import task you specify a poll time. Every x seconds the import task will poll the import DLL, which will return a list of all import items it knows about, with a change counter for each item. Each item, which is new or has been changed is (re-) imported. An example: an item can be a file in a specific directory and the change counter can be the last write time. Each item contains one or more shows. The import task tries to find a corresponding show for each show in the item and merges them. One additional hint: Not every import DLL can run in full-automatic mode. The DLL must provide some special functions and it must be possible to configure it in a way that it does not have to display any dialog boxes.

5.1.2 Merging modes

There are six different ways to merge an imported list into a destination list:

- **Fill elements one after the other by imported elements**: In this mode, the merging routine builds two lists with all elements of the specified class and replaces them one after the other. It does not matter whether an element is in a block or not, the first element, which appears in the destination list, is replaced by the first element in the import, the second by the second and so on.
- **Replace interval limited by info elements completely by import**: In the destination list you can create two info elements and mark them as import start and import end. All elements between these separators will be replaced by the import, independent of the class. The import and the destination can contain multiple separator blocks. The merging routine identifies the right blocks by the title or the start time of the start info element, which must be exactly identical in the import and the destination. If there is only a single block in the import and the destination this is not necessary, because it is unambiguous anyway. It is even not necessary that the import and the destination show contain the separator info elements, the full list will be automatically enclosed in separators with title “Import-Separator”. Therefore, you can use this mode to import full shows into destination shows (without any separator on any side). In this case, it is very important that the show structure at the destination is exactly identical to the show structure of the import.
- **Replace tracks at destination completely by imported tracks**: This is the simplest merge mode because the whole content of the destination is fully replaced by the imported content. In other words: no merge happens at all. This mode exists mostly for distinctiveness. You can achieve nearly the same behavior by using the previous mode (Replace interval limited by info elements) while not using info elements at all. But as this possibility is a little bit hidden, this extra mode was created. A single little difference does exist nevertheless: the interval mode will try not to overwrite the file information of elements. Therefore, the import tries to identify identical elements in the import and in the destination and if matched it will keep the file metadata. The “Replace completely” mode always replaces everything at the destination, even the file information.
- **Synchronize destination tracks completely with imported tracks**: This is a mode, which is similar to the previous one. The import routine does not work on specific classes only, but instead treats everything within a track. The previous mode is very simple because it simply replaces the whole content. This mode is more intelligent because it analyses the content on both sides and can pay attention to multiple conditions of protections at the destination, like an import-protection flag or elements which are just being played or which were sent. In addition, the import routine tries to identify identical elements on both sides in order to maintain metadata like filenames when an element is only moved to a different position. There is an extra chapter below, which explains this mode in detail.
- **Fill blocks one after the other by imported blocks**: In this mode, the merging routine builds a list with blocks from the import and the destination. A block is either a real group or is generated automatically from a non-interrupted list of elements of one class. When the blocks are identified, the first block is copied to the first block, the second to the second, and so on. In each block, the elements are replaced one after the other.
- **Find matching blocks of the import and the plan from the start time**: This is nearly the same as before, but the simple assignment first on first, second on second is replaced. Now the merging routine tries to find matching blocks by looking at the start times of each block. For each block it looks for the best fitting block on the other list.
- **Find matching blocks of the import and the plan from the title**: This is nearly the same as before, but corresponding blocks are identified with the block title. In this mode the order of the blocks does not matter, the match can be done if the block order in the import and the destination is different. But the title must be unique! If a title appears multiple times, the behaviour is undefined.

Hint: We use the terms “block” and “group”. What’s the difference? A group is a node type/tagname, which is different from a single element. A group can be a container for child elements and it is visible as an extra hierarchy in the XML data. A block is a list of consecutive elements with the same class. These elements are on the same level, therefore there is no hierarchy visible. Unfortunately, both terms are not used very precisely. For the purpose of importing both things can often be mixed: sometimes the import can treat a block in the same manner as a group. This is the case if a block is identified only by the import class. It also works with the start time as criteria because a block has the times of the interval given by the outermost elements. Mixing blocks and groups does not work if you want to use the title to identify groups. This will work only with real groups, because only groups can have a title.

5.1.3 Detailed explanation of the merging modes based on a working class

The merging starts with two schedule lists, one in the BCS, which is the destination and one from the other system which is the source. The destination list is not overwritten; it is merged with the source. This makes it possible to re-perform the import many times without destroying the structure or other elements of the BCS list. Before we continue with more theory here is an example. The first picture shows the destination list, which has obviously not yet been filled with an import and the second picture shows the list of a music importing DLL:

Start m	Start ty	Start	Stop	Duration	Class	P/A/E	Title
		01:00:00	01:00:00	00:00:00			News
		01:00:00	01:00:00	00:00:00			Placeholder
		01:00:00	01:00:00	00:00:00			Musicblock 1
		01:00:00	01:00:00	00:00:00			Placeholder
		01:00:00	01:00:00	00:00:00			Promotion
		01:00:00	01:00:00	00:00:00			Musicblock 2
		01:00:00	01:00:00	00:00:00			Placeholder
		01:00:00	01:00:00	00:00:00			Promotion
		01:00:00	01:00:00	00:00:00			Commercials
		01:00:00	01:00:00	00:00:00			Placeholder
		01:00:00	01:00:00	00:00:00			Musicblock 3
		01:00:00	01:00:00	00:00:00			Placeholder
		01:00:00	01:00:00	00:00:00			Traffic Service

Start	Start	Start	Stop	Duration	Clas	P/A/E	Title
		00:00:00	00:02:00	00:02:00			# News
		00:02:00	00:05:47	00:03:47		BRYAN ADAMS	BEFORE THE NIGHT IS OVER
		00:05:47	00:09:13	00:03:26		CLEOPATRA	COME AND GET ME
		00:09:13	00:13:09	00:03:56		RICKY MARTIN	PRIVATE EMOTION
		00:13:09	00:13:09	00:00:00			# Promo
		00:13:09	00:16:48	00:03:39		WES	ALANE
		00:16:48	00:20:16	00:03:28		DAVID GRAY	BABYLON
		00:20:16	00:23:36	00:03:20		TEXAS	SUMMER SON
		00:23:36	00:23:36	00:00:00			# Promo
		00:23:36	00:27:23	00:03:47		SURVIVOR	EYE OF TIGER
		00:27:23	00:31:38	00:04:15		KIM WILDE	YOU CAME
		00:31:38	00:32:38	00:01:00			#Traffic info

Now the task of the import function is to merge both lists. This is done in two main steps. In the first step corresponding blocks in both lists are searched. In the second step the elements within each block are copied to the destination. The most important thing is that the import function will treat only elements of a single class, the primary class ! Elements with a different class are never imported or changed. In the example above the primary class is music. In the BCS list the import routine will identify three music blocks. This is quite easy because there are real groups in this list. In the source list there are no such real groups. This does not matter because the music blocks are obviously separated by non-music elements. Therefore, we can identify three music blocks in this list too.
O.k. This was simple, wasn't it ? But live is not always simple. Have a look at the start times. You will see that the first source element starts at 0:00 while the first destination element starts at 1:00. Normally this problem should not appear because you would usually import exactly the right elements starting at the correct time. Either we have made an error and imported wrong elements or it was our intention to import elements from another show/hour.
The import routine provides two modes how to identify corresponding blocks. The first mode means: "Fill blocks one after the other with imported blocks". That means the block times are totally ignored and the first imported block will be filled into the first destination block and so on. The administrator can explicitly select this mode but it is also used when the times of the imported blocks are outside of the time range of the destination show or if no valid date/time information is specified at all. If the import function must use this mode implicitly the user will be warned and he can stop the operation. The second mode means: "Find matching blocks of the import and the plan from the start time". In this mode the import routine tries to find the best matching destination block for each source block.

Best matching means, it calculates the start time difference and uses the minimal combination. This mode can only be used, if the start times of the blocks are roughly fixed or the routine will pick wrong blocks easily. By default, you should use the first mode.
For the first mode, there is still a special consideration. Sometimes the shows of the source and destination list are not identical. E.g. in the BCS list, you have a single show from 0:00 to 6:00 but the import list contains a show for each hour. Here you can still use the first mode because the import routine will not write all imported hours into the first destination hour. Instead, it starts to find corresponding blocks at the start time of the imported show. That means if you import e.g. the 2:00 show it will start scanning the destination list at 2:00 but after this first match the time information is ignored and the blocks are assigned continuously. (See also the import detail option “Stay within hour separators” below.)
Of course, it can happen that there are too many blocks either in the source or in the destination. The import function can automatically create missing blocks or delete surplus blocks in the BCS list. Both operations are critical. For the creation of new blocks, the function tries to find the right place from the start time. If time information is not complete, the block is created at the end of the list. Deletion of blocks is always dangerous. Therefore, you should be careful with this feature. Better, pay attention that the structure of the BCS and the imported list is always identical.
In the second step the elements of each source block are copied to the corresponding destination block. No time information is considered at all. The elements are imported from the beginning to the end of the block. In the example above the first element of the first music block is "Before the night is over". It will overwrite the first music element at the destination, the placeholder. In case it is not the first time the destination list is filled it would already contain some real music elements. They would be replaced by the imported elements in the same way as the placeholder is replaced. If there are too much or not enough elements at the destination the import routine can automatically create or delete elements. You can find two flags for this feature in the settings dialog. In contrast to the same flags for the blocks, these ones can be used much better because they are less dangerous. In the example we must create missing elements, because we have only a single placeholder for three real elements.
It is possible that the list of elements is interrupted by elements of a different class. If you have real blocks (aka groups, as in the example destination) it does not matter. If there are foreign classes in a group, they are simply ignored and do not change the class of the whole group. If a block is formed only by a list of consecutive elements (as in the source example) a foreign element will always end a block. There is only one exception for this rule: elements of the class "Info" are regarded as if they belonged to the previous element and do not interrupt a block. Besides, even a consecutive list of elements of one class is seen as interrupted (hence separated into blocks) if there is a gap of more than an adjustable time between two elements.
Now we can close this part with the destination list of our example after the import:

Start m	Start ty	Start	Stop	Duration	Class	P/A/E	Title
		01:00:00	01:00:00	00:00:00			News
		01:00:00	01:00:00	00:00:00			Placeholder
		01:00:00	01:11:09	00:11:09			Musicblock 1
		01:00:00	01:03:47	00:03:47		BRYAN ADAMS	BEFORE THE NIGHT IS OVER
		01:03:47	01:07:13	00:03:26		CLEOPATRA	COME AND GET ME
		01:07:13	01:11:09	00:03:56		RICKY MARTIN	PRIVATE EMOTION
		01:11:09	01:11:09	00:00:00			Promotion
		01:11:09	01:21:36	00:10:27			Musicblock 2
		01:11:09	01:14:48	00:03:39		WES	ALANE
		01:14:48	01:18:16	00:03:28		DAVID GRAY	BABYLON
		01:18:16	01:21:36	00:03:20		TEXAS	SUMMER SON
		01:21:36	01:21:36	00:00:00			Promotion
		01:21:36	01:21:36	00:00:00			Commercials
		01:21:36	01:21:36	00:00:00			Placeholder
		01:21:36	01:29:38	00:08:02			Musicblock 3
		01:21:36	01:25:23	00:03:47		SURVIVOR	EYE OF TIGER
		01:25:23	01:29:38	00:04:15		KIM WILDE	YOU CAME
		01:29:38	01:29:38	00:00:00			Traffic Service

5.1.4 Stories

Since version 4.4 BroadcastSystem supports stories. Despite of any possible meaning of the word "story" during the production process, we understand it in BroadcastSystem as a mini-rundown, a sequence of text and other media elements to be seen as a unit for one topic, and the text is intended to be read on-air. In 2013 we introduced a new node type "Story" which can appear top-level in a rundown or inside of a group (see chapter 2.1, describing the XML structure).

Regarding the import this node type implies a new problem. In principle a story can be used now in every place where formerly only an "Element" could be. Therefore, there can be 3 levels below the rundown level: group, story, and element.

In order to avoid too complex import logic and to be backward-compatible some determinations were made:

- The logic regarding groups/blocks/elements as described above is unchanged. During block matching a story is treated like a single element.
- This also implies: a story can replace an element and vice versa.
- A story is seen as an all-or-nothing unit. This means that a story is always imported as a whole and many import settings (see below) do not apply (e.g. create/delete missing elements: this is always assumed to be yes within a story).
- Nevertheless, some elements at the destination are kept, even if they are not (or are no longer) part of the imported story. This applies to elements with send states "sending", "sent" or "skipped" or for elements with import protection. These elements are moved to the beginning of the destination story before the import elements are inserted or aligned.
- The merge function tries to do its best to keep important metadata at the destination when a reimport is performed. Especially the file specifications, fade values and the ID should be kept.

5.1.5 EndpointStories

Since version 6.1 BroadcastSystem supports EndpointStories. They are used to combine the output to different endpoints into a single container. For example, an EndpointStory could contain all elements for a regionalized playout to different regions. The import dlls and the import routines handle EndpointStories like regular stories (see above). A difference is made for the improve step, which is being executed by DigaSQL.dll. This step is the database lookup in order to get all needed metadata. Here it could happen that an EndpointStory and all its children have the same ID (e.g. MusicID). In DigaSQL.dll versions before 3.20.3493.0 the dll did stop with the database search when the very first item was found. In the described case with multiple elements with the same ID this is probably undesired. Therefore, since version 3.20.3493.0 the DigaSQL.dll first assembles all items which can be found in all configured lookup tables. If there is exactly one EndpointStory in the found items, it is picked and the dll forgets about individual items, which were found, too. The improvement does not fail here because multiple matches were made. As a result, the improvement step of the import will always pick the EndpointStory from the database, even if individual elements with the same ID are found.

5.1.6 Show structure alignment

Since version 4.9 the automatic import in BUS is able to make the structure of shows at the destination identical to the structure of shows from the import. This feature is activated by checking the two check boxes “Create missing shows at the destination” and “Delete surplus shows at the destination” of the configuration dialog in BUS (see chapter 7.4).

The basic idea is to make the shows at the destination identical to the imported shows but the destination shows are not all deleted to be created anew afterwards. Instead the import tries to identify shows which are already present at the destination. This allows to run an import multiple times and avoids that content is deleted though it does not change during a re-import.

There are 3 criteria which can be used to match identical shows:

- the name of the shows,
- the start and stop times of the shows and
- the field “ShowID”.

The comparison is strict, meaning: string comparisons (for the name and the ID) are case sensitive and when the times are compared the start time and the stop time must be identical up to 1 millisecond. If you select more than one criterion, all of them must be identical to give a match.

Here is an example of a comparison by times:

	Import		Destination		
	00:00 – 05:00	Night shift	00:00 – 05:00	Night shift	
created at dest. ->	05:00 – 09:00	Morning show	05:00 – 06:00	Early bird	<- deleted
			06:00 – 09:00	Bavarian sunrise	<- deleted
	09:00 – 11:00	Happy morning	09:00 – 11:00	Happy morning	
	11:00 – 14:00	Lunch with Tina	11:00 – 14:00	Lunch talk	<- kept
	14:00 – 17:00	Tea time	14:00 – 17:00	Tea time	
created at dest. ->	17:00 – 18:00	On the road			
	18:00 – 21:00	Prime time	18:00 – 21:00	Prime time	
	21:00 – 24:00	Late night show	21:00 – 24:00	Late night show	

“Early bird” and “Bavarian sunrise” will be replaced by “Morning show”. The show “Lunch talk” will not be changed because a show with the time 11:00 – 14:00 exists on both sides with exactly the same times. The different name does not matter as long as only the times are compared. “On the road” will be created at the destination.

There are some minor options to define which shows should be treated by the import. These options are listed in the description for the BUS settings dialog. It is not necessary that an import contains all shows of a whole day. If you do not select the option “Delete surplus shows” the import task can handle import items with a single show, too. If a matching show is being found its content is replaced or merged, if no matching show is found it is created in the destination. This allows importing one show after the other. Then it is up to the user operating the system, which creates the import data to assure that all required shows exist in time.

The show structure alignment runs as an extra step in advance of the merge step. This allows seeing the options for the show structure alignment and for the merge step independently. There is only one important interference. If you activate show structure alignment it can be assumed that the show structure of the import and of the destination is identical after the first step. Therefore, the merge step will find exactly one single destination show for each import show. This can be different if no alignment is done. Then it can happen that the show structure of the import and the destination do not match. In this case the automatic import will do a merge for every overlap of import and destination show. Therefore, a single import show might be imported into multiple destination shows and vice versa. (Details can be configured in BUS import settings dialog.)

5.1.7 Import DLLs

Import DLLs provide an interface to a specific system. Besides, they are responsible to convert all sort of special formats into the native BCS format. Each of these DLLs has a particular set of parameters. Each DLL together with its parameter set can be stored in DigAIRange below the "Import" key in the tree view or together with an import task in BUS. Each import DLL provides a settings dialog for the DLL specific parameters, see below.

5.1.8 Improvement DLL

Many systems do not provide all the necessary data to play the entries. The most important information is the filename, of course. But some systems as the old MusicMaster e.g. do not handle files at all. Therefore, they cannot provide the filename. However, every system uses some sort of ID to handle its elements. This is a fact we can use to find a filename for imported elements. If each element is stored in the DigaSystem (or perhaps any other database), we can use the ID to find the element in the database and retrieve the missing information. This is the task of the improvement DLL.

At the moment only the DigaSQL.DLL can perform this task for a DigaSystem database. The DLL has a special function, which expects a section string with information about an element, tries to find it in the database and returns a string with the completed information. You specify the tables to be scanned as a comma separated list (“DSN1\Tab1,DSN2\Tab2,…”) together with settings specific for the import DLL, see below. At the moment the DLL can search in the fields "Music_MusicID" (MUSICID), “Music_CDID” (MEDIUMNAME), "Advert_Motive" (MOTIVE), "Filename" or in "DBRef" (=<DSN>\<Table>\<Refno>). In newer versions of the DLL this search can be configured, see the next chapter.

As it is possible that each import-field exists twice, once in the import data and once in the database, there is the possibility to define for each field which of both sources is preferred (field selection option, see above).

For a story lookup the DigaSQL.dll can find out the member elements of a story. This allows importing stories without elements but with one of the ID fields above and the improvement will automatically insert the member elements. Hint: this is only done if the imported story is empty. If there are already elements in an imported story, no element lookup in the database is done. Text elements in a story are never improved. For them it is assumed that the specific text for a certain rundown position always superposes the "default" text in the database.

5.1.9 Improvement search expressions

Since version 3.13.3447.0 DigaSQL supports search expression in order to specify how the database lookup is being done. Like the tables, which are scanned, the search expression needs to be configured in the settings dialogs of the import modules. If a module has a support for entering a search expression, you will see a big text field with name “Search expression for database search” in the settings dialog. If this field is missing, you might need to upgrade the import module.

Improvement search expressions are similar to the where-clause of an SQL statement. You can query an individual field by a comparison like [MUSICID]=”ABC” and you can combine these queries with AND and OR operators and left and right brackets. Here are all the details:

Reference

- It is possible to split database lookup in multiple steps by separating multiple search expressions into a comma-separated list. In this case, database lookups will be done with the given expressions from the first one to the last one. Lookup stops if an element is found.
- If you specify multiple database tables for the lookup and multiple search steps, the search steps are iterated in the outer loop. As an example of the execution order: step1 in table1, step1 in table2, step2 in table1, step2 in table2, step3 in table1, step3 in table2, …
- If you use a star “*” as special search expression the old lookup logic will be applied (see previous chapter).
- The field names you can use must be enclosed in square brackets, e.g: [MOTIVE]. The field name corresponds to the field name as it is used in database section strings (=DBE) or in DBX format. Custom fields can be specified, they must have a slash in the name. For ambivalent fields, which are present as new field in the “Extended”, table and as a misused field in an old table, the new field is taken. Therefore, this works only if you have a DigaSystem database version V5. (Hint: in the future it might be possible to specify database column names by omitting the square brackets, but this has not been implemented yet.)
- A single query for one field must contain a field name, an operator and something to compare the content with, e.g: [MUSICID]=”ABC”. If you use a percent character, this means: compare the content of the database column with the metadata content from the import. Example: [MUSICID]=%. The last example can even be abbreviated more by simply writing the field name: [MUSICID]. (Hint: in the future it might be possible to specify any arbitrary metadata field of the import by writing %fieldname, but this has not been implemented yet.)
- Supported operators are: =, !=, <>, <, <=, >, >=. Hint: LIKE is not supported, it is only used internally sometimes. Case insensitive comparison is not supported.
- Supported data types are: string, integer, definition, ressort, class, state, flags. Not supported are: date, time, duration. The data type is deduced automatically from the field you specify.
  - String: This is used by default.
  - Integer: If a field is known to be an integer, a numerical comparison is done. Nevertheless, you have to write a string comparison in the search expression, e.g: [SPOTLENGTH]=”20”.
  - Definition: For all known definition fields you have to write the string. An automatic conversion to the corresponding numerical value is done. Example: [LANGUAGE]=”English”. Fields, which can be declared as “behave like a definition field” and “custom definition fields” are internally regular string fields. Therefore, the string comparisons you specify work correctly without the need to convert to an integer value first.
  - Ressort: Like for definitions write the ressort as a string. It will be converted to a numerical value automatically. Attention: a query for a sub-ressort is not supported so far.
  - Class: Specify the string value. A conversion to the integer value is done automatically. Example: [CLASS]=”Music”.
  - State: Specify the string value. A conversion to the integer value is done automatically. Example: [STATE]=”Existing”.
  - Flags: Specify the string value as it is used in section strings. There are individual sections for each flag. Possible values are 1 and 0. Example: [READY]=”1”.
- Search expressions can be built up by combining individual queries with the operators AND / OR. It is also possible to use left / right brackets. Example:
  ([CLASS]=”Music” OR [CLASS]=”Audio”) AND [MUSICID]=%
  If you omit the operator between to queries AND is assumed automatically. Nevertheless, this is not recommended to avoid a bad readability, except if you simply list fields to be checked against the imported metadata, e.g: [MUSICID] [CLASS].
- There is a parser in the DigaSQL.dll, which splits the search expressions. If a syntax error is encountered, an error message is printed to the log of type “DigaSQL-Improve”. Nevertheless, the parser does not fully check the specified expressions. Therefore, other errors might occur when the generated SQL statement

is sent to the SQL server. To find the corresponding message check the log destination “DigaSQL-Error” and to check the generated SQL statements see the log destination “DigaSQL-SQLCommand”.

Examples

Find an element of class Music with the correct MusicID:

```
[CLASS]=”Music” AND [MUSICID]=%
```

Find a commercial element, which has already been declared as Ready:

```
[CLASS]=”Commercial” and [MOTIVE]=% and [READY]=”1”
```

Perform a multi-step search for branded music elements. The branding is visible from the custom field “STATION/BRANDING”. If the correct branded version is not found the second step tries to find a non-branded version. The third step tries to find any element with the right MusicID only and the last step performs a search with the classic logic:

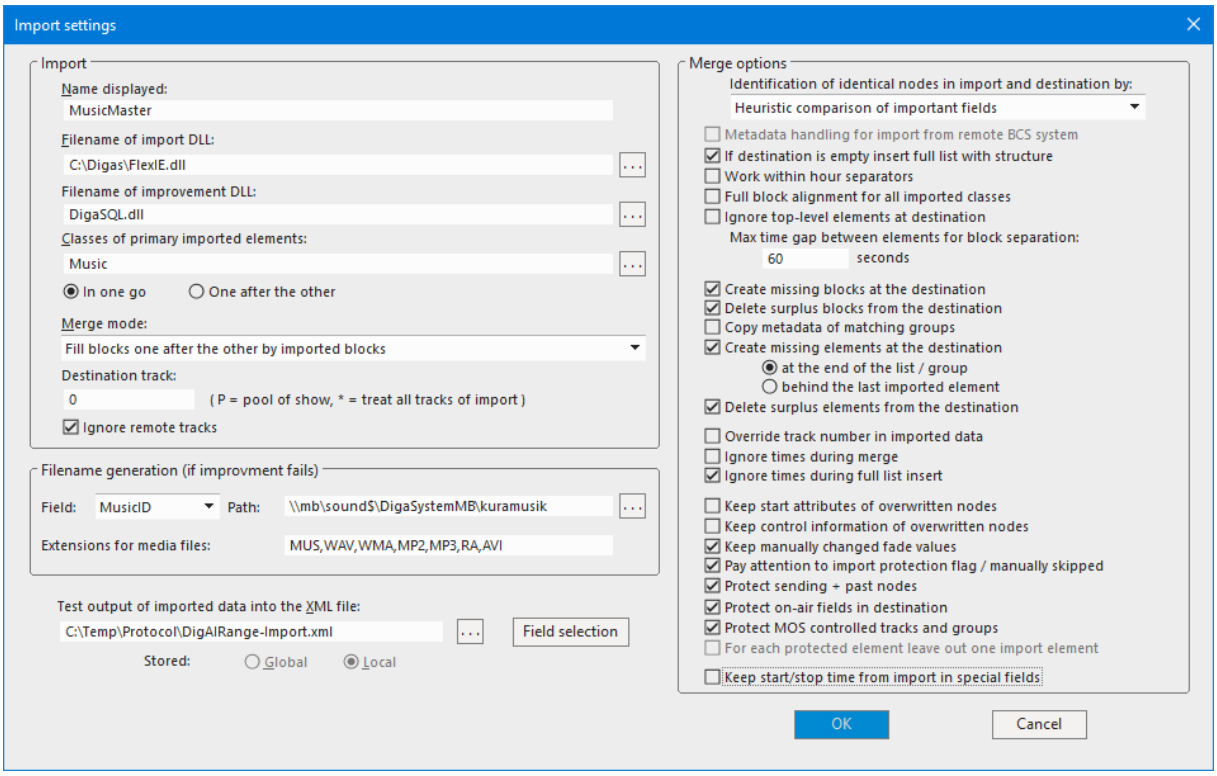
```
[MUSICID] AND [STATION/BRANDING],
[MUSICID] AND [STATION/BRANDING]=””,
[MUSICID],
*
```

5.1.10 Emergency filename generation

Normally, a filename for each element is either given from the imported data or it will be found by the improvement. If no filename is available, e.g. if the database server is not running, an emergency filename can be generated automatically from a defined path and one of the fields MusicID, CDID or Motive. With the automatically generated filename, the import task looks for the file. As the filename generation does not include the extension, you can list some media extension the task looks for. If it finds a file, it is stored in the field Filename0. If multiple files (with different extension) can be found it is undefined, which one will be used.

5.1.11 Import options

There are many additional options, which control the import and the merge process. We are going to discuss them now, but first here is a dialog in which you can see all options. It is the import settings dialog of DigAIRange but the options are also nearly identical to the settings in BUS. (Some of the fields of the dialog were already discussed above, others are discussed in chapter 7.4.about the settings of the import task of BUS.)



Classes of primary imported elements:

The idea of the import is to change only elements of the primary class and leave all other elements unchanged. But in the settings, you can also select multiple primary classes. If more than one class is defined you have to select whether all classes are merged in one run or each class is merged with an extra run. In the first case the process assumes an or-conjunction between the classes. E.g. if music and jingles are imported in one run, a music element can replace a music or jingle element at the destination and a jingle element can also replace music or jingle elements. This is not true if the classes are imported one after the other. The second option is like defining two separate import tasks.

Destination track:

By default, only the track 0 (the main scheduling track) is treated – meaning: this track is filled with import data. Imports do not have information for multiple tracks, normally. Therefore, you have to do multiple imports, one for each destination track, if you want to import into multiple tracks. If an import really yields multiple tracks you can use the special value * to treat all tracks which are listed in the import data. Of course, this implies that the track numbers in the import are present and correct. In special cases it might be desirable to import into show pools instead of tracks of the shows. This can be achieved with the special value P.

Ignore remote tracks:

In a system with multiple BroadcastServers tracks of shows might be linked to a remote BCS. If no import should be done into these remote tracks, you must check this option. The import into other tracks is not affected by this option.

Test output of imported data into the XML file:

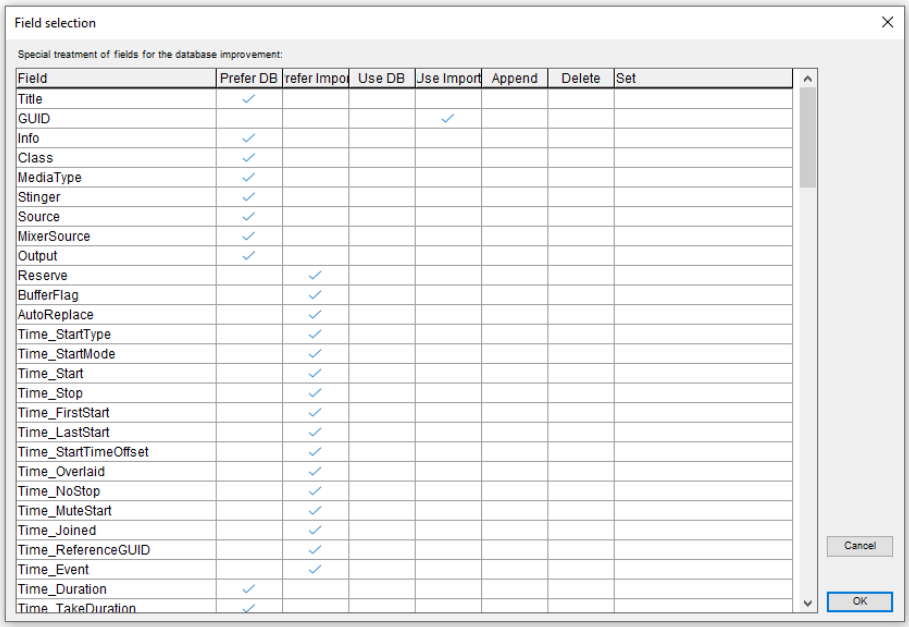
This field is intended for debugging. Whenever an import module is asked for import data, the result is written to the specified file. This can be helpful to see whether the import module yields correct information. For BUS, there is a "Debug" button, which opens a small dialog in which you can specify two output directories. This is used instead of a single filename because BUS will call the import module very often and with directories, you can get a file for each call.

Field selection:

This button opens a sub dialog, which allows you to define the treatment of each data field. This is useful if you perform an improvement by the DigaSQL.DLL. It is likely that some of the fields exist in both sources, in the import data and in the database (= improvement). By default, the data from the database will be preferred. But here you can define from which place the data is used / preferred. Possible values are:

- Prefer DB: The value from the database is used if it is present, otherwise the import data.
- Prefer Import: The value from the import is used if it is present, otherwise the database data.
- Use DB: Use the value from the database, even if it is empty.
- Use Import: Use the value from the import, even if it is empty.
- Append: Append the values from import and database.
- Delete: Do not import this field, remove it from the data.
- Set: Always set the field to a fixed string value, which can be specified.

Be aware that the dialog presents only some pre-defined fields. The last line is empty and allows adding any fields you need. You need to specify the tagnames as they are stored in BCS-XML.



Field	Prefer DB	refer Impor	Use DB	Use Import	Append	Delete	Set
Title	✓						
GUID				✓			
Info	✓						
Class	✓						
MediaType	✓						
Stinger	✓						
Source	✓						
MixerSource	✓						
Output	✓						
Reserve		✓					
BufferFlag		✓					
AutoReplace		✓					
Time_StartType		✓					
Time_StartMode		✓					
Time_Start		✓					
Time_Stop		✓					
Time_FirstStart		✓					
Time_LastStart		✓					
Time_StartTimeOffset		✓					
Time_Overlaid		✓					
Time_NoStop		✓					
Time_MuteStart		✓					
Time_Joined		✓					
Time_ReferenceGUID		✓					
Time_Event		✓					
Time_Duration	✓						
Time_TakeDuration	✓						

Identification of identical nodes in import and destination by:

The import routine can try to keep metadata of overwritten nodes. For example, this can be useful for the mirrored filenames. It would be a waste if the title of an element is updated and as a consequence all information about mirror files is lost and all these files have to be recreated. In order to keep metadata the import routine needs to check whether a node is meant to be the same one (=" identical node") in the import and in the destination. Depending on the merge mode the import routine can also use the methods defined here to find identical nodes if they were moved around. The most capable merge mode is "Synchronize destination tracks completely". Other merge modes cannot pick out old nodes from as many moved positions. The following identification methods are available:

Heuristic comparison of important fields	This is the most complicated method. Up to 25 fields are evaluated. For each field, which has the same content in import and destination a number is increased, weighted by the importance of the field. If a certain level is reached, the two nodes are treated as identical. Important fields are the ID fields (see below), the take duration or the database reference. This method can be used if it is unclear or if it varies which fields should be used best.
Comparison of title	The field "Title" is compared.
Comparison of hierarchical / enumerated title	The import routine constructs a virtual text from the title of the node and all of its parent nodes. In addition nodes with the same title on one level are enumerated. This could give a text like: "<Track0>|Music1|Happy hour (stinger)2" This method can be used if the title does not change and if there might be multiple nodes with the same title in one hierarchy level.

Comparison of Music-ID	The field “Music_MusicID” is compared.
Comparison of CDID	The field “Music_CDID” is compared.
Comparison of Motive	The field “Advert_Motive” is compared.
Comparison of Music-ID / CDID / Motive	All three ID fields are compared in the listed order. If the first ID in one of both nodes is not empty, this field will be used and compared. This method can help if your import contains nodes, which have their ID in different fields, e.g. MusicID for the music and the motive for the commercials.
Comparison of GUID	The field “GUID” is compared. This method can be useful for importing rundowns generated from another BCS system.
Comparison of database reference	The field “DBRef” is compared.
Comparison of filename 0	The field “File_Filename0” is compared.
Comparison of NodeID of import (is saved in destination)	The field “ImportedNodeID” is compared. It represents the ID as it is found in the import data. To be comparable in later imports, the content is stored as a second ID in the destination in field “ImportedNodeID”. This method can be useful for importing rundowns generated from another BCS system. Attention: the field is only generated when this method has been selected. Therefore, you cannot use this method to compare nodes which have been imported with a different method, the imported node ID will be missing in the data of the destination.
Comparison of “ImportID”	The field “ImportID” will be compared. This is not a standard BCS field but it might be possible to create such a field for/during the import. E.g. a VBS script could generate this field. Such an ID must identify each node unambiguously.
Comparison of start time	The field “Time_Start” will be compared. This option is intended for the very special case that the imported data contains only fixed starts for all elements and that the elements cannot be moved around - while getting a new start time - because this would make them unidentifiable in a re-import.

If destination is empty insert full list with structure:

Check this option in case you want to insert the imported list "as is" if the destination is empty (no elements, no blocks). This means: no merge as it is described above is done. Instead, the imported list (after improvement) will be written to the destination without any change of the structure. This is only useful if the imported data contains a complete structure you can use for your show and with placeholders for other elements. This feature is typically used if you make the structural planning in the music rotation and not in the BroadcastSystem. Then you can import your music plan and the structure will be imported too. Afterwards you can run additional imports to fill placeholders for commercials and so on.

Work within hour separators:

The option “Work within hour separators” is available for the block and element mode. This option can be useful if you are going to import hours, which are filled too much. E.g. a music import might contain so many elements that the last block starts in the next hour. Normally this block would be imported into the hour in which it starts, which is wrong in this case. To prevent this, you can create hour separators at each full hour in the destination show. If you activate the option, the import routine will first look for the interval between two separators (or show start/end) with the biggest time overlap with the imported hour. All operations will then be limited to the range between both separators.

Full block alignment for all imported classes:

This option is useful sometimes, if you want to import multiple classes one after the other. If the destination list differs from the desired end result, new blocks will be created at the end of the list. Example: let’s assume an import yields the blocks music, cart, music, music but the destination list contains only three music blocks. The missing cart block would be created behind the music blocks. To avoid this problem, you can force an extra alignment run by setting the flag. The extra run will create the missing blocks with the right classes as they appear in the import list. In the example, a cart block would be inserted in the second place. After this extra run the normal merging is performed, in the example two times for the two classes. But now the carts will be imported into the right place.

Ignore top-level elements at destination:

Check this option if you want to import exclusively into groups. As standard behaviour the merge function will see top-level elements with the right class as possible destination elements, too. If elements with the imported class exist on top level – e.g. as manually scheduled elements – and must not be overwritten, use this option.

Max time gap between elements for block separation:

Normally the merge function will see consecutive elements with the right class as a block (not a real group with elements in the group but treated like a group, see the import mode description above). Sometimes import data does not list these elements back-to-back (stop time of an element is identical to start time of next element) but instead there are gaps in between the elements. With this option you can define a limit for the gap size in seconds. If the gap is bigger the function will separate the elements into two blocks.

Create missing blocks at the destination:

If the import contains more blocks or groups than the destination, missing blocks will be created as a real group. Normally this is done at the end of the list or at the right time if the time is known. If you activate the "Full block alignment for all imported classes" option, it might also be in between of other blocks. This option is only available for some of the merge modes.

Delete surplus blocks from the destination:

If the import contains fewer blocks or groups than the destination, surplus blocks/groups will be deleted. Depending on the merge mode, this can be from the end of the list or in the middle of the list. This option is only available for some of the merge modes.

Copy metadata of matching groups:

The merge routine finds matching groups in the import and the destination and copies only the child elements. The metadata of the groups themselves is not copied, normally. You can change the behaviour by checking this option. Two hints: first, please note that this option says “groups” not “blocks” (like in the previous two options). The difference is that a block can be formed by a sequence of top-level elements of the imported class. In this case, there is no real group node present and therefore no metadata can be copied. Copying metadata is only possible for real group nodes. Second hint: the metadata of groups is not copied by default if a *merge* is done. If the import is inserted as a whole into the destination (e.g. if the option: “If destination is empty insert full list with structure” does apply) then the group metadata is also copied, of course.

Create missing element at the destination:

If the import contains fewer elements than the destination, missing elements will be created. You have the option to create these elements at the end of the list/group (list means a block of consecutive elements here) or behind the last imported element. Typically, both options will mean “at the end”. There is no option to evaluate the time and insert a missing element somewhere in the middle.

Delete surplus elements from the destination:

If the import contains more elements than the destination, surplus elements will be deleted.

Override track number in imported data:

In case the import module yields a track number (normally no one does) the track number might be wrong, meaning: it is not the number of the track into which you want to import. If this option is checked, the first track found in the import data is seen as the track to be imported. If the import contains multiple (wrong) tracks it is undefined which of the tracks will be used.

Ignore times during merge:

If checked, the start time of shows and blocks is ignored totally. Of course, this is only useful if you have a simple match of each source show to one destination show and if you want to replace the content of the destination show completely and from the beginning. Typically, this is useful when you have to import data with wrong times, e.g. a show of a different day. This option cannot be used together with any option, which evaluates the start times like “work within hour separators” or the match-by-time block mode, of course. For DigAIRange this option normally need not be set because if the user tries to import a show with non-matching times, he will be asked whether he wants to ignore the times (which is the same as this option). For BUS, the use of this option is very limited too, because BUS tries to find the right destination shows automatically and therefore needs correct time information. So pay attention when using this option.

Ignore times during full list insert:

If the destination list is empty it is possible to insert the full import list "as-is" into the destination. By default, there is still a selection of the groups and elements, which are taken from the import list: only groups and elements with a start time inside of the imported and the destination show are used. If you check this option all groups and elements of the import are used, even if their start time is totally off.

Keep start attributes of overwritten elements:

This option must be set if you have defined sensible start attributes for all placeholders at the destination and if you do not want to use the default start attributes which are generated by the import modules. (For an additional explanation see "Keep manually changed fade values" below.)

Keep control information of overwritten elements:

This is like before but here the control information for the timed macro commands of an element are being kept.

Keep manually changed fade values:

This option is needed for re-importing. Theoretically, it is possible to re-perform the import multiple times without losing important information at the destination. Of course, elements and their metadata are always kept for elements which do not have the primary class, or which lie on the outside of the separator elements, respectively. For the elements, which are treated by the merging routine it is more difficult. In all merging modes, the routine tries to identify identical elements in import and the destination. The “Identification method” mentioned above is applied to decide whether two elements are seen as being identical. When the function sees two elements as identical, it tries to keep some of the fields in the Broadcast System. E.g. the file information is always kept. This is done because otherwise BUS would start to re-copy/convert files it has already copied/converted. The whole fade part is kept if it has been changed at the destination and this option is set. If you say the fade-information from the import is more correct than any values in the BC system, you can clear this flag and the fade fields will always be taken from the import data. See also the import protection matrix below.

Pay attention to import protection flag / manually skipped

This option should be activated normally. If a user changes or adds an element manually, it might be overwritten or deleted the next time an import is performed. To protect such a manual element an import protection flag exists, which can be set with DigAIRange. But the flag is ignored unless you set this option. Since version 5.0 of the modules, it is also possible to set the protection flag for stories and for groups.
See also the import protection matrix below.
Since version 5.9 of the modules, elements, which were manually skipped by the user (not past elements, skipped by TurboPlayer when advancing) will be protected in the import, too.

Protect sending + past nodes:

In all merge modes, which do a real merge or synchronization the routine can try to keep sending or past nodes unchanged. Normally, it is a good idea to not import into shows, which are already being broadcasted. If this is done nevertheless, you should at least activate this option. Otherwise, it can happen that the import will overwrite sending or past elements, causing problems for TurboPlayer which might be audible on-air.
Hint: “past” does not refer to the times of a node. If a node has either send state “Sent” or if it has “Skipped” with a duration 0 (which normally means that TurboPlayer has skipped the node) it is defined to be a past node.
See also the import protection matrix below.

Protect on-air fields in the destination:

This setting is intended to be used if you import rundowns from another BCS system. Then it can happen that the imported elements are just playing or have already been played. Normally, fields, which signal these states (like the SendState), are copied to the destination like all other fields. If you don't want this behaviour, you can activate this setting. Then all on-air relevant fields from the source are filtered out and are not copied to the destination. This affects the SendState, the real times, the OnAirPlayed… fields and the very special fields "Time_IIRS" and "TurboPlayerGUID".

For the SendState there is a special treatment: the state "Skipped" is copied to the destination, if the skipping was done explicitly by a user on the source side. The SendStates "Placeholder", "Planned" and "Cleared" are copied, too.

Protect MOS controlled tracks and groups:

DigaMOS is the background program, which synchronizes content between a newsroom computer system (NCS) and a DigaBroadcastSystem via the Media Object Server (MOS) protocol. The synchronization can be done for tracks, show pools or groups. The content of a track/pool/group should either be filled by DigaMOS or by an import but never both ! Therefore, you should always check this protect option if you are using DigaMOS.

The protection of tracks and show pools works for all merge modes. You only need to be careful if you have a show structure alignment running because the show structure alignment might remove shows, which contain MOS controlled nodes. Regarding groups please be aware that the protection of MOS controlled groups works only for merge modes which pick destination groups for the import. If you select a merge mode like "Replace tracks at destination completely by imported tracks" which does not evaluate the group structure at the destination at all, the group protection cannot be applied.

Keep start/stop time from import in special fields:

After broadcasting, DigaSystem can report the "as-run-protocol" to other systems (reconciliation). For example, a music rotation wants to know which titles were played to adjust the future generation of the music schedule. In many cases, the receiving system can match the played elements to the planned ones with some ID field. But some systems require that the original planning time is visible somehow in the protocol (though the real play times might be totally different). This can be achieved with the help of this option. If checked the import routine will copy the original start and stop time into the special fields "Time_ImpStart" and "Time_ImpStop". As these fields are not used within BroadcastSystem, they remain unchanged and appear in the protocol. Now it is up to the used export module to convert these fields into the data as it is needed by the other system.

5.1.12 Import protection matrix

During a merge or synchronization there are multiple conditions, which make a node at the destination being protected. Depending on the type of condition and depending on the node hierarchy there can be multiple things, which can be protected. In detail:

Conditions:

- An element is just being sent or is assigned to the past (send state “Sent” or “Skipped” + real duration = 0),
- an explicit import protection flag was set or the element was manually skipped,
- a node was not imported by BUS.

All 3 conditions must be activated by the corresponding import setting. The last one is only available in BUS.

Which node fulfils the condition ?

- The checked node itself,
- one of the child nodes of the checked node (recursive),
- one of the parent nodes (recursive).

What can be protected ?

- The existence of a node is protected,
- the metadata of a node is protected,
- the position of a node is protected (meaning: it will not be moved out of its current hierarchy level),
- the full sub-hierarchy (meaning: all children) are completely protected.

Here is the import protection matrix, which defines the relation of condition, node and what is protected:

Condition | which node	Node itself	One of the children	One of the parents
Sending / past	– existence – metadata – position	– existence – position	
Import protection flag / manually skipped	– existence – metadata – sub-hierarchy	– existence	– existence – position – metadata – sub-hierarchy
Not imported by BUS	– existence – metadata	– existence	

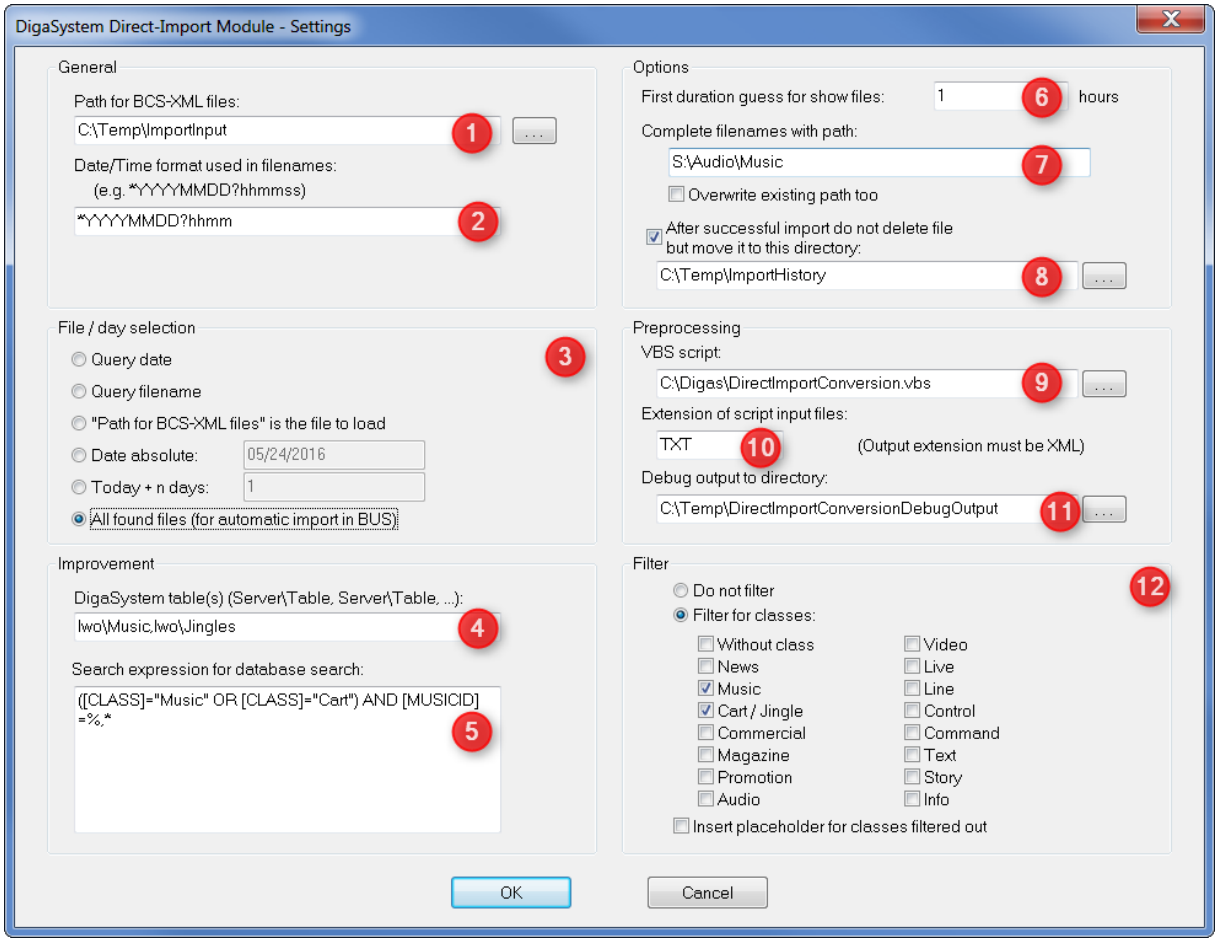
IMPORTANT: This matrix has been introduced with the merge mode “Synchronize destination tracks completely with imported tracks”. Therefore, the matrix is only fully available in this mode. Older merge modes support it only partially.

5.2 Import DLL specific Settings

Each import DLL has its specific settings. One DLL must know where to find the import files whereas the next must know how to connect to a server and so on. In this chapter, we want to discuss the configuration dialogs of some of the DLLs.

5.2.1 DirectImport.Dll

The DLL is used to import XML files in the format, which is internally used in DigaBroadcastSystem. In addition, it is possible to filter for element classes and to insert a VBS script into the data processing. Here is the configuration dialog:



1) The first edit box is the directory, which is scanned for input files. Depending on the settings in “File / day selection” (4), the user can still change to a different directory in manual mode, if “query filename” is set. Together with the setting "Path for BCS-XML files is the file to load" in (4), the field (1) specifies the file to be loaded directly, see below.

2) This is the date/time format as it is used within filenames. The DLL uses the format when it has to generate a filename when the user selects a date or the settings give a specific date. And it uses the format to extract a date and/or time from a given filename. This must be possible when using the DLL for an automatic import in BUS. In this case, the DLL must be able to determine the date/time of a file without opening it first. Depending on the type of the file, the date might be sufficient: for a file containing a full day, only the date is required. For a file containing a show, the filename must include the start time of the show, too.

   Here is a short explanation of the format specifier:

   - YYYY Year with 4 digits
   - YY Year with last 2 digits only
   - MM Month with 2 digits
   - DD Day with 2 digits
   - hh Hour with 2 digits
   - mm Minute with 2 digits
   - ss Second with 2 digits
   - fff Fraction of a second with 3 digits (=milliseconds)
   - ff Fraction of a second with 2 digits (=hundredth of a second)
   - f Fraction of a second with 1 digit (=tenth of a second)
   - * Placeholder for "any number of non-digits" - only when interpreting a filename. When a filename is generated, this format specifier does not yield any output characters.
   - ? Placeholder for "one char of non-digits" – only when interpreting a filename. When a filename is generated, this format specifier yields a space.
   - ... All other characters are interpreted/printed as they are.

   The standard format file naming for BCS-XML files (also used by DigAIRange) is “YYYYMMDD.XML” for day files and “YYYYMMDD-hhmm.XML” for show files where the date and time information is the start time.

3) This area is used to specify the way in which the import files are selected. These options are available:

   - Query date: During import, a dialog appears and the user has to select a date. The filename will be generated according to the date/time format.
   - Query filename: During import, a dialog appears and the user can select a file.
   - Path for BCS-XML files is the file to load: The field (1) does not contain a path but a filename. With every import the same file will be imported. This is useful to import always the same fixed exchange file.
   - Date absolute: A fixed date is given in the settings. The filename will be generated according to the date/time format. During automatic import, the imported files will be limited to the given date.
   - Today + n days: A date is computed as <today> + the given offset. The filename will be generated according to the date/time format. During automatic import, the imported files will be limited to the given date.
   - All found files Normally this option should be used with the automatic BUS import. All files, which are found in the specified import path will be imported.

   Pay attention not to use one of the "Query..." options for an automatic import because the popping up dialog will stop the running task. If you use "All found files" for the manual import it will be identical to "Query filename".

4) Here you can specify a list of DigaSystem database tables - specified like in the screenshot - for the database lookup (=improvement, see general part above).

5) Here you can specify a search expression for the database lookup. For details, please check chapter 5.1.8. Improvement search expressions.

6) The option “first duration guess for show files” is only needed when importing show files. Problem is that the duration of the show cannot be evaluated from the filename itself without opening the file. This is only important in the first part of the automatic import in which BUS polls the import module for known import items (=shows in this case). If you specify a wrong value here, it will not change the merge process, because this is done after the file has been opened and the correct duration is known. Only in case someone triggers a re-import, either from BUS or from DigAIRange, not enough files might be imported if the duration guess is wrong.

7) The parameter "Complete filenames with path" can be used if the imported data does not contain a path specification but only pure filenames. If the path information is existing but incorrect you can overwrite it with "Overwrite existing path too".

8) The DirectImport DLL can delete or move the imported files after a successful import. This option is triggered from the main application using the DLL, for example BUS. In the app there is only a general setting to remove the imported file. In the DLL configuration you can decide whether the file should really be deleted or whether it should only be moved to another directory (which must be different of the input directory, of course).

9) Here you can specify the filename of a script. This can be a VisualBasic script (*.vbs) or any script for the Windows Sripting Host. For every input file the DLL does import, the script is called with two command line parameters: first the input filename and second the output filename (a temporary name generated by the DLL). The script has to process the input file, convert or process the data and write the output file in BCS-XML format. The script must be finished within 60 seconds or the import will fail.

   Here is a simple VBS example, which only reads a file and writes it without any processing:

```
'*************************************************************
' Test VBS script for DirectImport.dll
' Dose not convert anything
' TristateTrue needed to work with Unicode files
'*************************************************************
Dim fso
Set fso = CreateObject ( "Scripting.FileSystemObject" )
Const ForReading = 1, ForWriting = 2, TristateTrue = -1, TristateFalse = 0

Set InputFile  = fso.OpenTextFile ( Wscript.Arguments.Item (0), ForReading, False,
                                    TristateTrue )
Set OutputFile = fso.OpenTextFile ( Wscript.Arguments.Item (1), ForWriting, True,
                                    TristateTrue )

Do While InputFile.AtEndOfStream <> True
    line = InputFile.ReadLine
    OutputFile.WriteLine line
Loop

InputFile.close
OutputFile.close
```

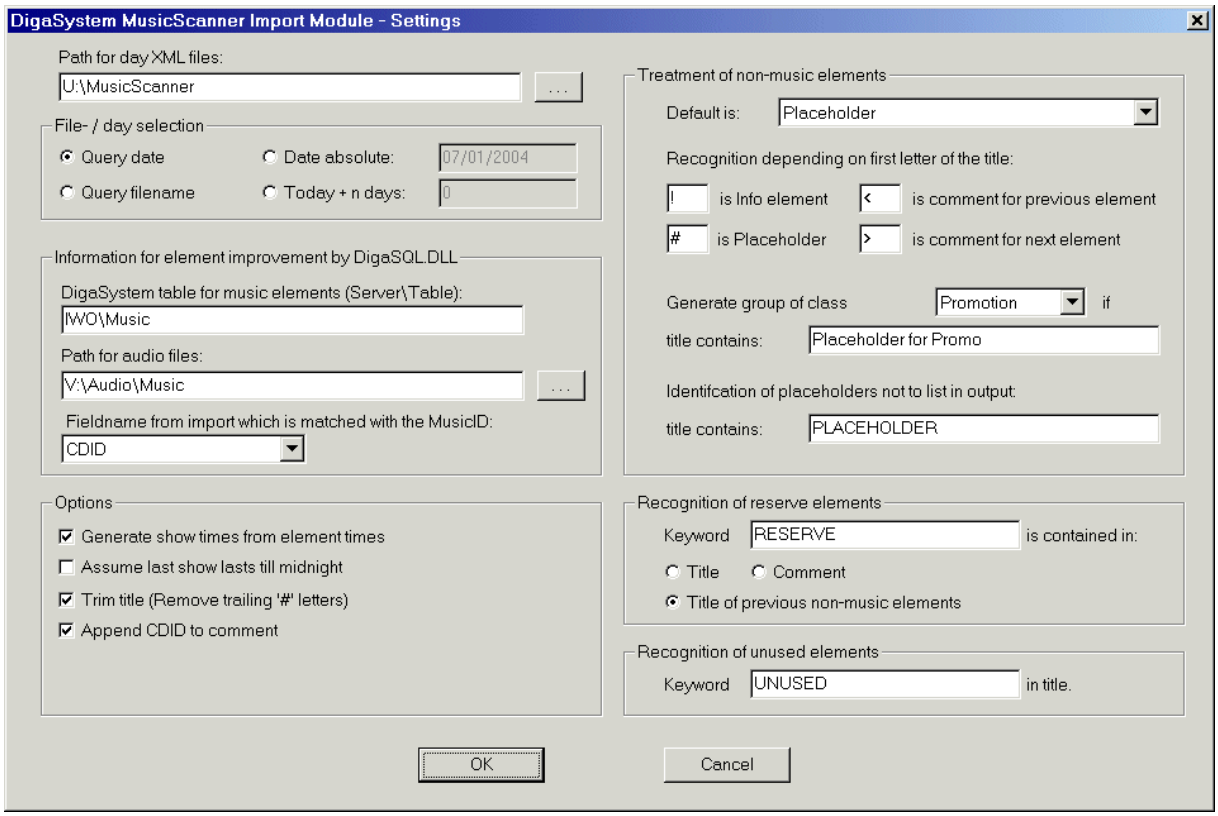
10) With the parameter "Extension of script input files", it is possible to scan for / read files, which do not have the standard extension "XML". This parameter is only valid if you specify a script too. It is the extension of the script input files – the output files must always be "*.XML" files.

11) Here you can specify a directory used for a debugging output. The DLL will copy the output file of every script call into this directory.

12) The filter options are needed, if you do not want to import a full rundown as it has been exported. Typically, this type of import is not used to import files, which come from DigaBroadcastSystem but from foreign systems, which are able to write the native BCS format. So you can use the class filters to produce the rundown you want to import. Normally you let pass only the class you are going to merge and let generate placeholders for all other classes.

5.2.2 MusicImport.Dll

The DLL is used to import files from the MusicScanner. This is a DigaSystem tool which converts MusicMaster text files into XML. Here is its dialog:



In the first line you enter the directory where to find the output files from the scanner. The group "File-/day selection" determines what happens when an import is executed. The DLL will either ask the user with a dialog for the date or the name of the file to import. Or it will automatically use a specific date.
The next box is information which is simply passed through to the improvement DLL. You can specify either the DigaSystem tables (can be many, comma-separated) or the corresponding media directory (only one !) where to find the files for the imported elements. You do not have to specify both fields, though it was done in the example above. There are two possible fields from MusicMaster which can be assigned to the DigaSystem field “MusicID”.
All the other options are present because of special customer requests. Most of them should be self-explanatory. Here is the explanation for some of them: In the big group on the right the treatment for non-music elements can be configured. The problem is how to distinguish between music, info and other elements because there is no intrinsic class information in the imported data. The solution used here works with the first letter of the title. If it is one of the letters entered in the four small edit fields, the DLL will produce an appropriate output. E.g. if the first letter is a '>' the line will be seen as comment for the next element (which is appendend to any existing comment). If none of the first letters matches, the default setting will be used.
The same problem arises for the recognition of reserve and unused elements. This can be done by writing some keyword as "Reserve" into the title or comment of an element.

5.2.3 FlexIE.Dll

FlexIE.dll is the successor of StandardIE.dll and MusicMasterIE.dll. It replaces the old modules which were designed for usage with MusicMaster only and provide more flexibility.
FlexIE.dll allows doing very flexible imports based on section strings and export based on “one string – one item” text formats.
The DLL is fully configurable and works with ASCII files for import and export workflows.

Important hint: As documented FlexIE.dll works with section strings. This means that sections are packaged in brackets. In case of importing values with contain brackets ‘[‘ or ‘]’ these brackets have to be replaced with two special ASCII characters.

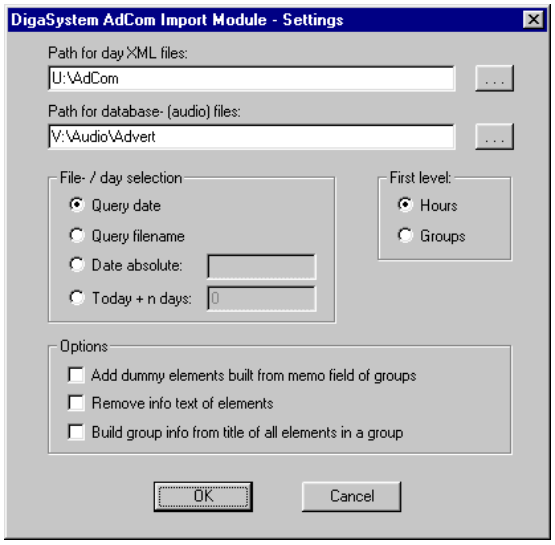
[→ ASCII code 02
] → ASCII code 03

In the import files these ASCII characters are displayed as because ASCII 02 and 03 are not printable but these two characters will be reverted to [and] during the import.

As the FlexIE.Dll is very complex there is an extra document describing all features and settings: FlexIE.doc.

5.2.4 AdComImport.Dll

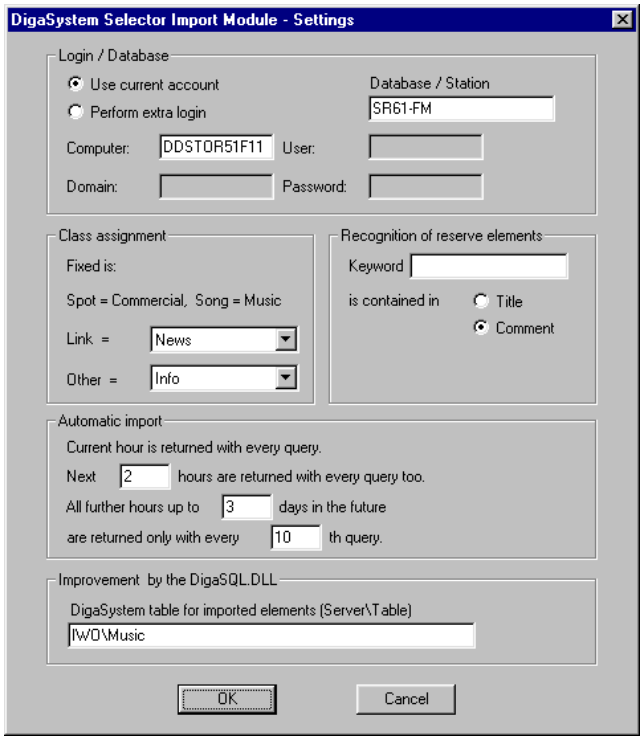
The DLL is used to import files from the AdComManager. This is a DigaSystem tool which converts AluP text files into XML (and allows changes to the list). Here is the dialog:



In the first line you enter the directory where to find the AdCom output files. The path for the audio files is the database directory where the corresponding files are located. This field is simply passed through to the improvement DLL.
The group "File-/day selection" determines what happens when an import is executed. The DLL will either ask the user with a dialog for the date or the name of the file to import. Or it will automatically use a specific date.
With the first level option you can choose between two values: hours or groups. The AdCom data is not separated into shows but the resulting XML data must have shows. Therefore, the DLL can generate these shows. If you choose "groups" every group in the input will yield a show in the output. If you choose "hours" the DLL will generate virtual shows, one per hour. Each show will contain groups on the next level. If you are not sure which of the possibilities to use, try "hours" first.
The last group box contains three options: (1) If an imported group contains a memo, the memo will be deleted and instead a dummy element (commercial, placeholder) will be generated as first element of the group. If you do not know why this could be useful, simply don't check the box. (2) This option deletes the info field of each element. This can be useful if the info text contains only information for the commercial system but not for the presenter. (3) If this option is set, an info for the group is generated which contains the title of all child elements. Therefore, the presenter will have an overview of the child elements without expanding such a commercial group.

5.2.5 SelectorImport.Dll

This DLL is used to import data from RCS Selector version 14 for Windows or newer. For these newer Selectors RCS provides an additional interface to read the schedules. It works via DCOM calls and transfers XML data. You have to install this interface before you can use the import DLL. Get the necessary files from RCS and install it on the server and all client computers which want to perform the import.
The settings dialog of the DLL looks like this:

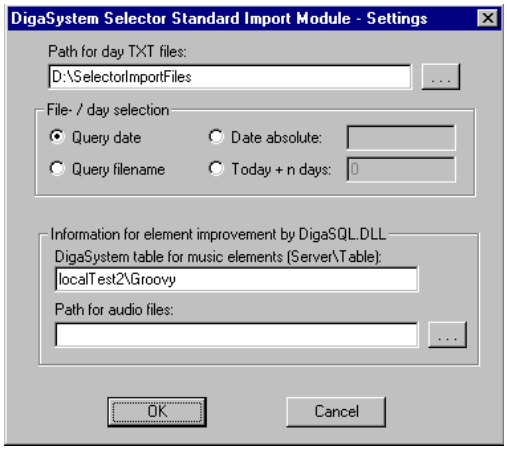


The first group contains data necessary to connect to the RCS server. The server name must be specified always. You have two options, you can either specify a login with username, password and domainname, or you can use an integrated login with the user currently logged on on the computer. All names wanted here are names of your Windows domain, the server name is its NetBIOS name. The database / station field is the name of the Selector / Linker database. If you do not know it, open the RCS license manager.
In the next group you can change the class assignments. Selector does not have a class like the DigaSytsem, but it knows four element types: Song, Spot, Link and Other. The first two are always mapped to the DigaSystem classes Music and Commercial respectively. The mappings for the remaining two element types can be choosen freely but we recommend to keep the default classes News and Info. Normally only music is imported from the Selector schedule. Therefore, non-music elements will be ignored anyway.
The recognition of reserve elements is done by finding a keyword (or a string) in either the title or the comment of an element. As Selector does not use the concept of reserve elements, this feature has no sensible application normally.
The group "Automatic import" is important. Selector does not know the concept of days / shows / groups. Instead it uses an endless list of elements. As it is not possible to import an infinite list, the list is limited virtually. When used with the automatic import, the DLL will be polled by BUS regularly. The DLL has to answer which import items it knows about and whether they have been changed. The Selector import dll reports the current and the next n hours as changed whenever BUS asks. You can specify a range more into the future where the hours are reported as changed only for every n-th query. Important: the DLL reports a change before really importing the data. Therfore it does not really know whether the data has been changed or not. This system simply means: together with the polling time of BUS you can make the next hours be imported regularly. The BUS polling time should be quite large, e.g. an hour, because the import is really time consuming.
The last group is the DigaSystem database and table where the audio files for the Selector elements are located. The information is simply passed through to the improvement DLL. This group is optional. If you do not need the improvement or if your Selector output does already contain filenames you do not have to specify a table.

5.2.6 SelectorStdImport.Dll

The Dll is used to import day schedule Textfiles produced by the RCS Selector Dos Version. Hint: despite its name the DLL does not import the standard selector format but a special sort of section strings.

Here is a snapshot of its Settings Dialog :



In the first line you enter the directory where to find the output files from the Selector. The group "File-/day selection" determines what happens when an import is executed. The DLL will either ask the user with a dialog for the date or the name of the file to import. Or it will automatically use a specific date.
The following boxes information is simply passed through to the improvement DLL. You can specify either the DigaSystem table or the corresponding media directory where to find the files for the imported elements. You do not have to specify both.

Note about the Format of the "Selector files" that are imported by the Dll :
The filename already determines the date of the playlist : "YYMMDD.txt". ("020328.txt" is the playlist produced for the 28th of March 2002)

The Playlist enumerates all the shows for the day, and all elements (music or place holder) within each show in the following syntax (extract of a file):

```
{[1]0700:00[2] 420[3]*** NEWS 7 MINUTES ***                          }
{[1]0707:00[2] 180[3]*** SPORT 3 MINUTES ***                            }
<[1]0710:00[2] 194[3]Stand by your man                                [4]CD
[5]Lyle Lovett
[6]12-118.427-03 01.01[7]     [8]     [9]
[A]CS[B]00.0[C]    >
{[1]0713:14[2]  15[3]--- PROGRAM ---                                   }
<[1]0713:29[2] 198[3]Toxic Girl                                     [4]CD
[5]Kings of Convenience
[6]12-135.285-00 01.01[7]     [8]     [9]
[A]F [B]07.0[C]    >
{[1]0716:47[2] 300[3]--- DAY HIGHLIGHTS ---                                          }
<[1]0721:47[2] 120[3]Barefoot again                        [4]CD
[5]RinkaLaOne
[6]12-048.346-05 01.10[7]     [8]     [9]
[A]CH[B]00.0[C]    >
```

Syntax Analysis :
Each line reports information about an element. Each element is either a **music element**(starting with “<[1]” and ending with “>”) or a **place-holder**(starting with “{[1]” and ending with “}”). A Show is marked by a place holder element (for News for example) and always starts on the hour (6h00, 7h00, etc..). A file always starts logically with the mark for the first show of the day (see example above, the file starts with the first show at 07h00).

“{“ indicates start of a place-holder element (end : “}”)
“<” indicates start of a music element (end : “>”)

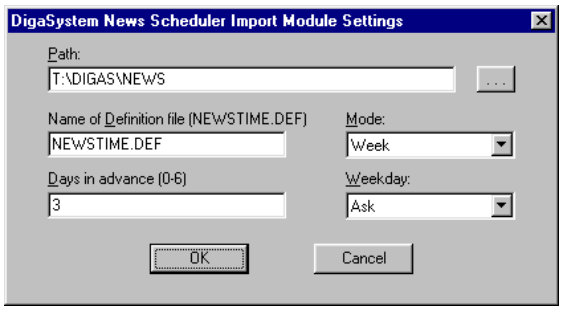
Common tags for **both** element types :
[1] : Start Time (format : **hhmm:ss**)
[2]: Duration in seconds
[3]: Title

Tags specific to **music-elements** only :
[4]: Source
[5]: Artist
[6]: Music ID
[7]: Mark In (format: hhmm:ss)
[8]: Mark Out (same format)
[9]: Comment (so far in a single line)
[A]: Ramp Out (text comment on ending)
[B]: Intro (format ss.x, where x : tenth of seconds)
[C]: Intro2 (same format)

Important Note : the Music ID will be used as Music ID in the DigaSystem to register the Music Items
Therefore, its length should never exceed 19 characters.

5.2.7 SeplImport.Dll

This DLL is used to import the SND files from the old 16 Bit News Scheduler or from the SePl.Exe. Here is the settings dialog:



The first field is the root path for the SND files. In day mode it is the path where the SND files are located. In week mode there are seven subdirectories for each weekday in the path. The second field is the name of the definition file which is "NEWSTIME.DEF" by default. Mode can be either "Week" or "Day".
The field "Days in advance" is used for the automatic import. The DLL has to generate schedules with a date and time for each show and element. But there are no dates in the SND entries. That means the date must be generated by the DLL. The concept is like this: for the current weekday always the current date is generated. For the next n weekdays a date in the future is generated, for all other weekdays a date in the past. E.g. if today is monday and you want 3 "days in advance", the weekdays tuesday to thursday are in the future and the weekdays friday to sunday are in the past. To avoid importing old schedules into the wrong destination you should choose a small value for the days in advance and you should check the option "Do not import past shows" in BUS.
The last field "Weekday" specifies which weekday is imported in manual mode. The DLL can either ask the user, or generate a list for every weekday including "today".

5.2.8 NewsScheduleXmlIE.dll

This DLL is used to import and/or export the XML files from the old NewsSchedulerX / NewsPlayerX system. The configuration is quite easy but nevertheless there is an extra documentation: "NewsScheduleXmlIE.doc".

5.2.9 DirectBcsImport.dll

The DLL is used to import data directly from another BCS without using the indirection of files. So far, the only relevant use case is the duplications of shows from a master location to a backup location as it is described in chapter 7.4.1 “Using Import Task for duplicating shows to a remote BCS system”.

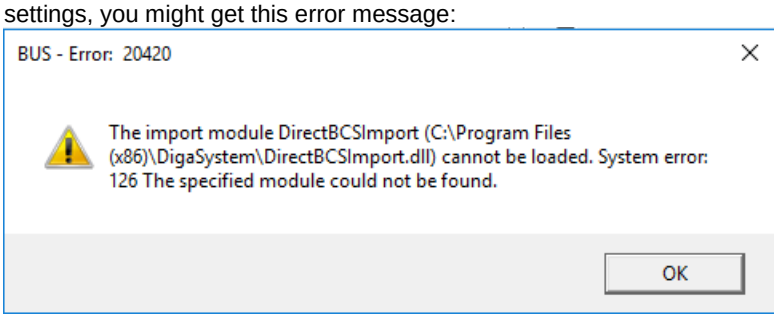
Installation

In contrast to most of the other modules of the DigaBroadcastSystem, this dll is linked dynamically to the C/C++/MFC runtimes. As a result, some additional dlls are needed, which are not needed by other modules.

The reason for linking dynamically is that linking statically to the needed libraries is not possible, because there is an incompatibility between the boost::thread library and the MFC library, if both are used within one dll. This problem is well known by the developer of the boost::thread library, but no action has been taken to solve the problem.

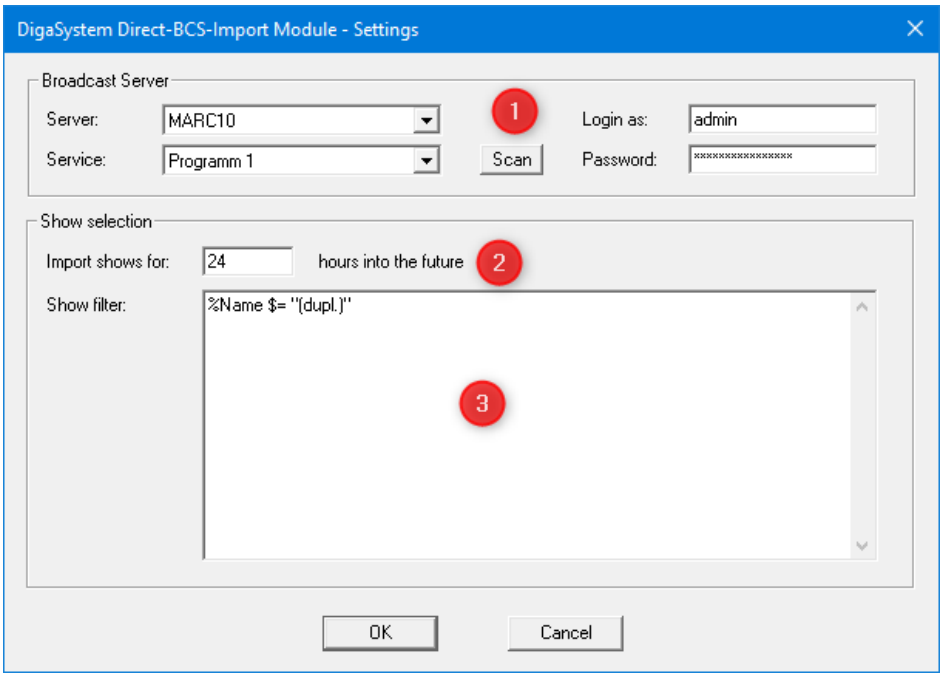
For installation, we need to distinguish three different parts of dlls:

1) The boost::thread dll is always needed. Please always copy it to the folder, where the DirectBCSImport.dll is placed (typically, the DigaSystem folder). The exact name differs, but one example is: boost_thread-vc142-mt-x32-1_74.dll.
2) The C runtime files are always needed, but they are not part of the delivery. You can install the latest version (which is also usable by older programs built by Visual Studio) by downloading it from the Microsoft page: https://support.microsoft.com/en-ca/help/2977003/the-latest-supported-visual-c-downloads
   Please use the “Permalink for latest supported x86 version”.
3) The C++ runtime files and the MFC runtime files might or might not be needed. It depends on whether they are already present on your computer. In most cases this is already true, because Microsoft programs use these libraries, too, and the files typically can be found in a folder below C:\Windows.
   We recommend to not copy the extra dlls, which are provided together with the DirectBCSImport.dll and first try to configure the dll in a new BUS import task. If you then click onto the “Configure” button in the import task settings, you might get this error message:



   This error report is coming from the Windows loader, which is responsible for loading modules and dlls. Obviously, the error report is wrong, because it appears too, if the dll mentioned in the error text can be loaded, but one of the further dlls, which are needed by the specified dll cannot be found. This is a well-known bug of the Windows loader and Microsoft does not fix it (e.g. by correctly saying, which dll is missing). So, you can only guess, which dll is missing. Or you can use ProcessMonitor of the SysInternals suite to monitor the API calls when the process tries to load this dll and try to find out, which additional module fails to load.
   The recommendation is to copy all the other dlls, which are part of the delivery to the folder, in which BUS and the DirectBCSImport.dll are located. As this can create incompatibilities with other programs, which are installed in the DigaSystem folder, we recommend to create an extra subfolder and install BUS, the DirectBCSImport.dll and any other needed dll into this subfolder. This keeps the C++/MFC dlls separated from any other program. You can then run an extra BUS instance from this folder, which only performs the job to synchronize rundowns.

Here is the configuration dialog of the DirectBCSImport.dll:



1) Here you configure the parameters for the source BCS. “Server” lists the known BroadcastServers as they are configured in the registry. Select one. Next you should enter valid credentials. Then you can press the “Scan” button so that the combo box “Service” is filled and then you can select a service.

2) This option is the number of hours into the future to specify the interval, in which the module should look for shows. This is done in interval <now,future hours>. Every show, which overlaps (even slightly) with this interval will be reported to the application as an import item. The module requests notifications for these shows and monitors them continuously. Whenever a change occurs, this is reported to the application (BUS).

3) With the optional show filter, you can restrict the shows, which are handled by the module. The filter is written in the macro language as it is explained in chapter 7.1.10 “Macros and filter expressions”. You can either write a simple expression (like it is visible in the screenshot, in which the filter checks for shows, which have the text “(dupl.)” somewhere in the name of the show). Or you can write macro functions. In this case the last function must have the name “filter” and it must either return true in order to handle a show or false in order to ignore the show.
In both cases 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. 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.
Here is an example:

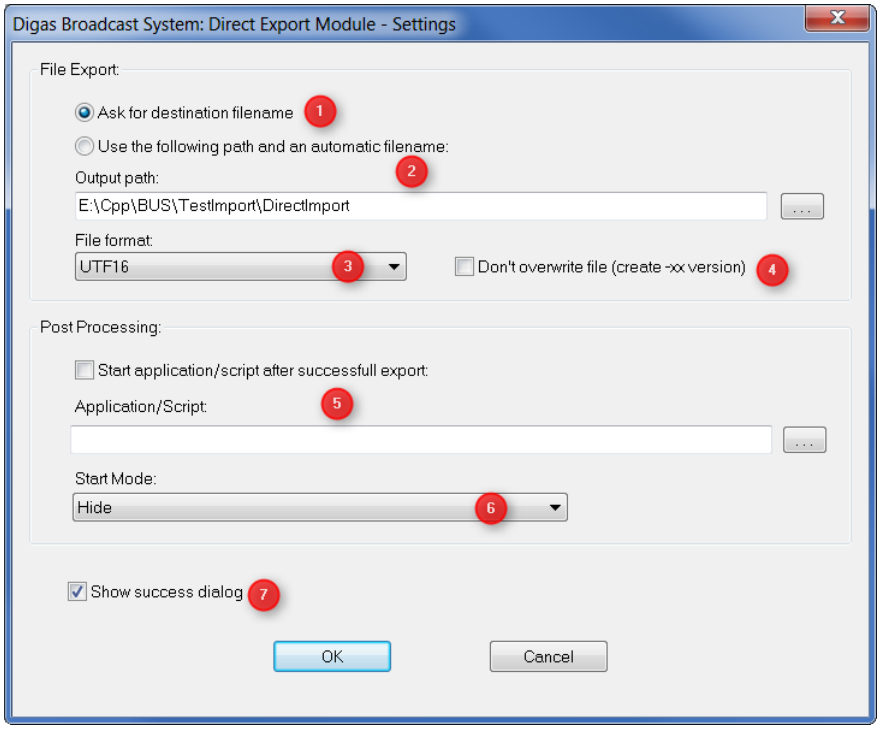
```
function filter ( )
{
    if ( DataOfShow ( "Name" ) $= "(dupl.)" )
    {
        return true
    }
    else
    {
        return false
    }
}
```

5.3 Export Dll specific Settings

Here is the explanation of the settings of some selected export modules.

5.3.1 DirectExport.Dll

The DirectExport.dll exports the BCS-XML as it is, no transformation is done. Therefore, the DLL is mainly used for a backup or to transfer the XML data from one system to another one. Nevertheless, a transformation of the data can be achieved, because the DLL can call a script or application after it has written the data to a file. This allows any usage of the data when there is a script or application, which can do something with the data. Here is the settings dialog:

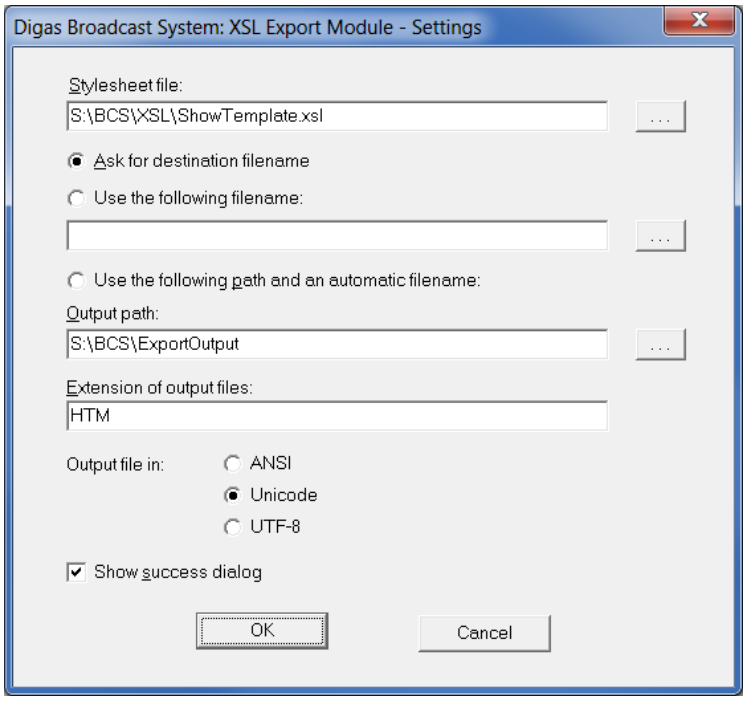


1) If checked the DLL will ask the user with a file-save-dialog to which file the data should be stored. This option is only useful for manual operation, of course.
2) This option is used for an automatic export. You specify the output directory and the DLL automatically creates a filename. This is:
   - "YYYYMMDD.XML" for exported days
   - "YYYYMMDD-HHMM.XML" for exported shows
   - "YYYYMMDD-HHMM-HHMM.XML" for a range of shows
   - "<Name>-<ID>.XML" for a day template
   - "Pool_<Name>.XML" for a pool
   - "PreProd_<Name>.XML" for a pre-production
   - "JingleGroup_<Name>.XML" for a jingle group
3) Here you can select the text format of the output file: ANSI, UTF-8 and UTF-16 with or without a byte-order-mark (=BOM) for UTF.
4) Normally the DLL overwrites an existing file. If this is not desired you can check this option. Then the DLL will create the first export file like described above. Afterwards every export will create a new file with the filename "<old_filename>-<counter>.<old_extension>". The counter will be in a 36-number system.
5) Here you can specify either a script or an application to be started after the export file was written. The filename will be specified as first parameter of the command line.
6) The start mode is an option for the Windows function starting the application. It is intended to give the application a hint how to start (normal, invisible, maximized, ...). Unfortunately, it is up to the application to evalute this parameter, which many programs don't do at all.

7) A success dialog can be shown after a successfull export. Again, this is only useful in manual operation. the box is shown when the file was written and the application of (5) was started but the DLL does not wait until the application has finished to do whatever it does.

5.3.2 XSLExport.Dll

This DLL is used to export schedules into files. The internal XML data can be combined with XSLT-stylesheets before the result is written to a file. That allows to select only some of the elements or only some of the XML data and to generate HTML output. Here is the settings dialog:



In the first field, specify the filename of the stylesheet file to use. With the next option, you can choose how the output filename is generated. The DLL can ask the user with a dialog, it can use a fixed filename or it can use an automatically generated filename. Typically, the generated name contains an identification like the service, the show, the date or the time of the show. The field "Output path" should be filled in both cases. If the user is asked, it is only a default. The extension of the output file must be given too.

The next option is to define the character encoding used for writing the file: ANSI, Unicode or UTF-8. Pay attention: in case you write XML files the encoding appears in 3 places: the real character encoding, a byte order mark (BOM) in the first two or three bytes of the file and in an encoding attribute within the introduction text. The DLL will write a correct BOM for Unicode and UTF-8. In general, it is up to you to specify the correct encoding attribute in the input XSL file. But internally the DLL uses the MSXML parser for the transformation and possibly this parser does not use the input encoding attribute but instead provides a wrong own one. The DLL (since build 13) is able to correct some of the possibly wrong encoding attributes for an XML file. But this cannot be handled correctly in all possible cases – especially if such an encoding attribute is needed for any unknown file format.

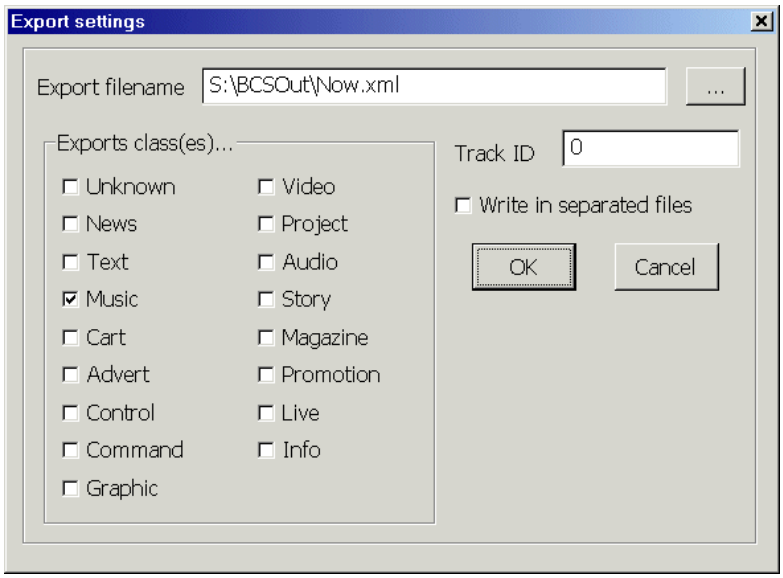
The last option "Show success dialog" is only useful in manual mode. Without it, the user does not get a positive feedback after the export. In automatic mode, this option can be fatal.

5.3.3 FullExport.Dll

This export module is like the DirectExport.Dll. It exports the full XML data as it is. In addition to the XML file, it exports all media files (audio+text) too. This is useful if you want to send a whole show to a different location. Simply send the whole directory. The receiver can use the function “Import from file” in DigAIRange to import the data with all media files. The settings dialog is as simple as the one above.

5.3.4 EasyXMLExport.Dll

This module writes a simple XML file with some information about the current on-air element, the previous and the next element. If one of these elements is not defined, it is omitted. The main purpose of the file is to support displaying the current “playlist” in the internet or to provide information for an RDS system. Here is the settings dialog:



You can define the name of the export file. It is a single file, which is updated with every change in the BCS data. The track ID is the BCS track within a show, which is exported. Default is 0 which is the schedule track. Other tracks should not be used. Below you can select which classes should appear in the output file. If the option “Write in separated files” is selected, each time an export is performed a new file is being written. In this case the specified filename is no longer used. Instead, each time a filename is generated with the current timestamp.
Attention: This export DLL should be used with BUS in the mode “Write after every OnAir state change”. This is a special mode which will hand over two shows to the DLL, if necessary, so the DLL has always the information of the last and the next element, even if they are within the last or the next show.
Here is an example of the XML text generated:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<wddxPacket version="1.0">
    <item sequence="past">
        <author> … </author>
        <artist> … </artist>
        <title> … </title>
        <time>MM:SS</time>
    </item>
    <item sequence="present">
        <author> … </author>
        <artist> … </artist>
        <title> … </title>
        <time>MM:SS</time>
    </item>
    <item sequence="future">
        <author> … </author>
        <artist> … </artist>
        <title> … </title>
        <time>MM:SS</time>
    </item>
</wddxPacket>
```

5.3.5 FlexIE.Dll

FlexIE.dll is the successor of StandardIE.dll and MusicMasterIE.dll. It replaces the old modules which were designed for usage with MusicMaster only and provide more flexibility.
FlexIE.dll allows doing very flexible imports based on section strings and export based on “one string – one item” text formats.
The DLL is fully configurable and works with ASCII files for import and export workflows.

Important hint: As documented FlexIE.dll works with section strings. This means that sections are packaged in brackets. In case of importing values with brackets ‘[‘ or ‘]’ these brackets have to be delivered as ASCII descriptor.
ASCII code 02 → [
ASCII code 03 →]
In the export files these ASCII characters are displayed as because ASCII 02 and 03 are not printable, but these two special characters have to be replaced by [and] by the receiving module.

As the FlexIE.Dll is very complex there is an extra document describing all features and settings: FlexIE.doc.

5.3.6 FlexXML.Dll

The FlexXML.dll can work in two modes:

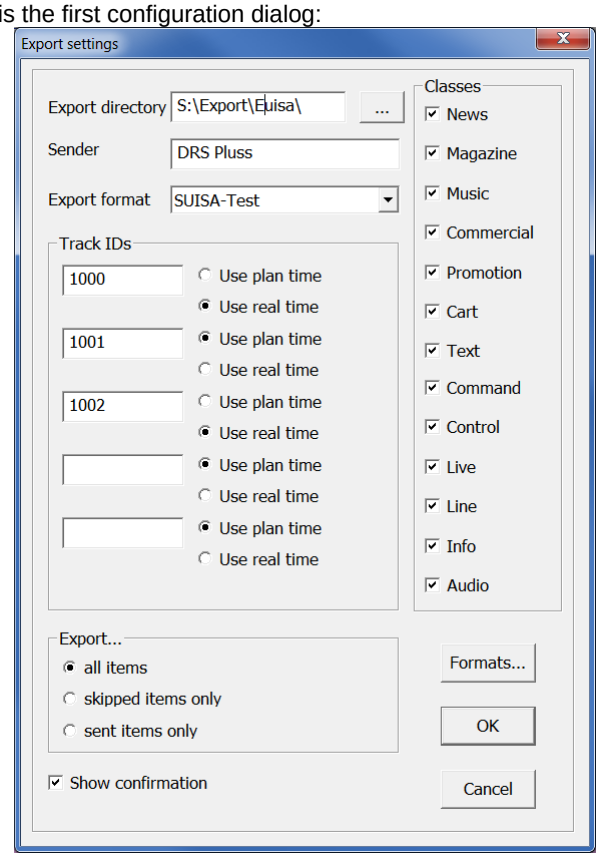
- The FlexXML.dll generates real time information about the last played item, the current item and the upcoming item. This information is used for e.g. displaying the title information on a website.
- Export any items passed through user defined conditions in XML format

The FlexXML.dll can be fully configured to work with dedicated classes, titles, types, etc. to leave out unnecessary information like commands or controls.

As the FlexXML.Dll is very complex there is an extra document describing all features and settings: FlexXML.doc.

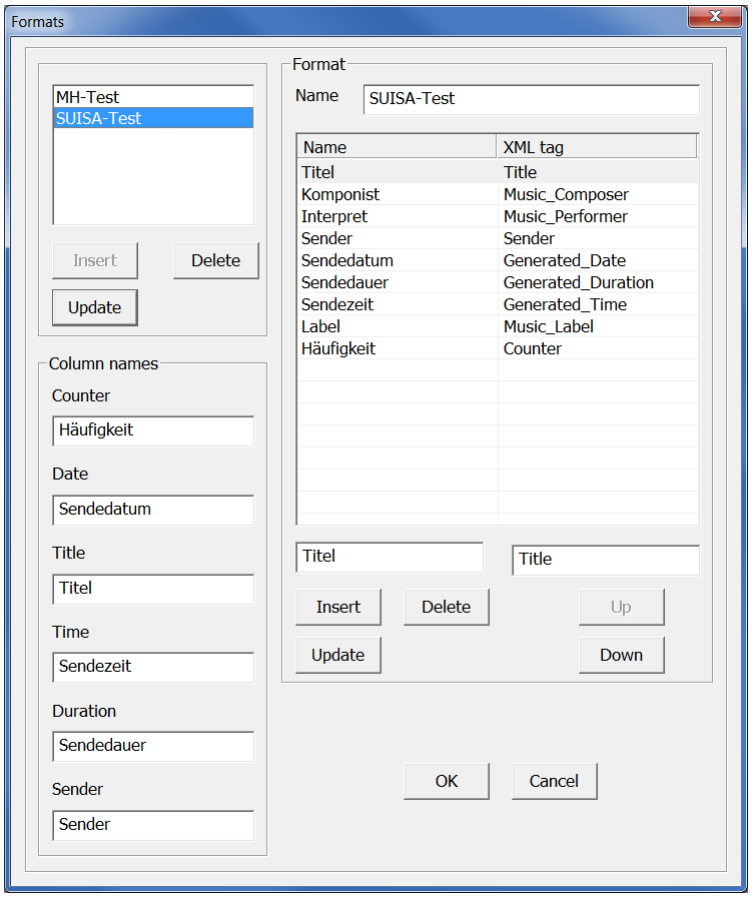
5.3.7 SUISAExport.dll

Despite its name, this export dll is quite useful, because it is widely configurable. The DLL creates pure text files, one file per month. Each line represents an entry. The columns printed can be configured and are tab-separated, so they can be read from Excel, e.g.
A general explanation in advance: You can assign each XML tagname which can be found in the BCS data to an exported column. In addition, there are some columns which have a special meaning. First there are columns which must be evaluated from the dll, like the “Counter”. This is a column which counts how many times an entry was being sent, the first time it appears in the file the counter is 1, in the second line with the same entry the counter is 2, and so on. The dll calculates the counter value by analyzing the file already written and therefore it must know which column contains the counter. Second there are some “virtual” tagnames which are generated by the dll. This are the tagnames “Generated_Time”, “Generated_Date” and “Generated_Duration”. It can be either the planned or the real start date / time / duration, depending on the settings for the corresponding track.
Here is the first configuration dialog:



- **Export directory:** Directory in which the files will be exported.
- **Sender:** Broadcasting station, yields a virtual tagname, which can be used as an exported column.
- **Export format:** Selects the export format used, the formats are defined with the second configuration dialog which appears if you press the button “Formats…”
- **Track IDs:** The dll can add up to 5 tracks to the export. Here you can specify the track numbers and whether real or planned times are used (for generating the special tagnames “Generated_Date”…) for this track. Hint: the result will be time-sorted but the dll does not check whether there are multiple elements for the same time on different tracks, it simply prints the sum of all elements.
- **Classes**: The classes to be exported.
- **Formats**…**(button):** Displays the second configuration dialog to define export formats.
- **Show confirmation**: Show success or error message box after export. Do not use with BUS !

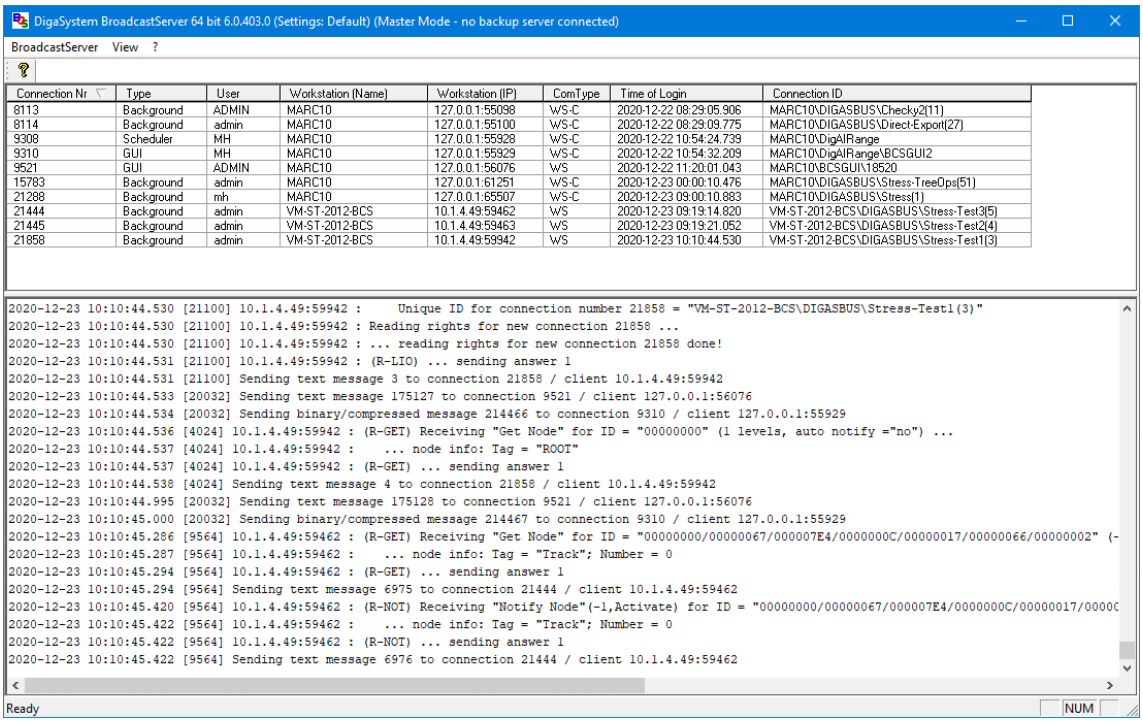
Here is the second configuration dialog:



The top-left listbox shows the available formats. (All formats are stored in the local DigaSystem registry.) You can select one of them for editing. On the right hand side you see the name of the format you are editing. Below the name is the list of the exported columns. On the left hand side of the list, you specify the export name of the column. This is the name, which appears in the first output line. On the right hand side, you specify the XML tagname for the column. This can be every tagname used in the BCS-XML or one of the generated tagnames, see above. If you want to export one of the special columns defined in the group “Column names”, e.g. the counter, specify the name you have given the column as export name. For the columns “Counter” and “Sender” the XML tagname does not matter and can be left free.
On the left hand side, you have the group “Column names”. Here you must enter the column names (the export name, not the XML tagname !) of the special columns which are evaluated by the dll.

6 BROADCAST SERVER

6.1 General information



Since version 5.4 BroadcastServer can run in three different ways: as a desktop application, as a sys tray application and as a Windows service. The installation steps for running BCS as a desktop application must be done in every case. For running in sys tray or as a Windows service, additional steps are necessary.

Since version 6.x BroadcastServer is available as a 64-bit application, too. There is an extra chapter below, which describes special things you need to do for upgrading to and using the 64-bit version.

It is strongly recommended to always start BCS as a desktop application first until installation has been fully completed and all your tests have been finished. Only afterwards, you should change the way BCS operates.

Requirements

BCS needs a Win32 operation system. A server version is not necessary, but a 64-bit system is strongly recommended.

Minimum hardware requirement:

2 GByte RAM, singe core CPU

Recommended hardware:

>= 16 GByte RAM, Xeon, >= 16 cores

Be aware that the minimum requirement is only sufficient for a very small system with a few clients. If you have many clients, a full program with many elements or many remote links you must spend as much memory and cpu power as possible. In general a full-time service normally needs 4 GBytes.

NOTE: For the 32-bit version of BCS we do neither recommend nor support multiple full-time services on a single BroadcastServer! For the 64-bit version this limitation does not apply. Hint: if you want to host multiple services, consider using the parameter "BroadcastServer\<Config>\General\UseAdvancedTreeLock".

NOTE: Under a 32-bit operating system, a single process cannot use more than 2 GB of memory, regardless how much physical memory is installed in the computer. But if BCS.exe is running under an x64 64-bit operating system, the BCS process can use up to about 3.8 GB of memory. Therefore, 4 GB RAM or more are recommended if BCS is running under a 64-bit version of Windows. The 64-bit version does not have this limitation. In this case it helps to have much more memory. A minimum of 16 GByte is recommended.

NOTE: You should also know that we can only recommend to run BCS on a physical computer with professional server hardware and integrated hard disks, either mirrored or as RAID array. This is especially important if BCS should run in a cluster with a master/buddy pair. Here there is even one more requirement: both computers of the cluster must be connected by a single switch, which handles all client connections. Otherwise, it cannot be guaranteed that the synchronization between the two BCS servers of a cluster will work correctly. Especially in a combination of virtual machines and the usage of a NAS instead of local hard disks, customers have encountered problems with the synchronization. In such an installation multiple problems with network caching or incompatibilites between Windows and a Linux-based NAS can arise.

6.2 Installation

- **DigaSQL.DLL has to be installed**, and a local DigaSystem parameter file must be created with the DigaSystem Administrator (Admin.exe) tool. Version 1.018 or higher of DigaSQL.DLL is required, but the most current version is generally recommended! If you want to run the 64-bit variant of BCS you need DigaSQL64.dll.
- A new directory for the following files can be created or you can put the following files into the DigaSystem(64) folder:
  - BCS.exe or BCS64.exe
  - Restart2.exe
  - libtcmalloc_minimal.dll (only needed for the 32-bit version of BCS)

  It is strongly recommended to use a separate directory for the 64-bit modules of Diga(Broadcast)System! Therefore, put the 64-bit BCS into a folder like: C:\Program Files\DigaSystem64.

  By default, BroadcastServer will store its data in a subdirectory of this directory. However, this can be changed (see Settings).
- Open the Admin tool. Go the local registry and create the top-level key “BroadcastServer”. Within this key create the value: Installed=1 as integer value. Hint: this value is a protection against BCS starting with an invalid or corrupt registry, which could result in data loss.
- Start BCS. After starting BCS should write default settings into the DigaSystem registry in the local key “BroadcastServer\Default”. You can terminate BCS again.
- Use the Admin tool and this document to scroll through the parameters to see whether the default for each value is suitable for you. If not, please adjust the parameters accordingly.
- If you want to run BCS as application, which can hide into the sys tray of Windows you can set the parameter “General\HideToSysTray” to TRUE. If BCS is running with a window, you can also use the main menu command: “View / Hide window to sys tray after startup”.
- If you want to run BCS as a Windows service, you need to install the executable first as new service. This can be done by opening a command line – which must be started with “Run as administrator” ! Navigate to the location of BCS.exe and run the command “BCS.exe –install”. This will register BCS with the Service Control Manager of Windows. There are additional command line options you can use, please see below. Afterwards you can start BCS via the Service Control Manager or let BCS be started automatically.

  Please pay attention to 3 topics. The first one is the account with which BCS runs. This is explained below with the command line options. The second topic is about the automatic restart. If running as a desktop application BCS can make use of Restart2.exe to start itself if a downgrade: master -> buddy must be performed or if an unrecoverable error is being detected. This does not work if BCS is running as a service because the Service Control Manager must start a service. You should check the third property page of the service, which lists actions in case of a failure of the service. The actions for all failures should be to automatically restart the service. Hint: this is automatically configured by BCS when it is installing itself. The 3. topic is about logging – see the extra chapter below.

6.3 Command line options

BCS 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
-key -k	location	BroadcastServer supports the use of a different DigaSystem parameter file set than the default one. This feature can be used to run several instances of BCS.exe on one computer, with all using e.g. different DigaSystem users. The command line parameter –key:location can be used to say that the paths of the parameter file set are stored in the Windows Registry key *Location*. *Location* should begin with either "HKLM" or "HKCU".
-config -c	configuration	BroadcastServer stores its settings in the local DigaSystem registry. All settings are stored below the DigaSystem registry key BroadcastServer/*Configuration*. *Configuration* is the name of the parameter set. This can optionally be passed to BCS.exe with the –config:configuration command line parameter. If this is omitted, the parameter set "Default" is used.
-b -bm	Filename -	When BCS starts as a Windows service, it is running in a different "ring" and probably with a different account than programs running on the regular desktop. As a consequence, mapped drive letters might not be available like for programs on the desktop. This can affect any area, but it is a special problem with the registry / parameter files of the DigaSQL.dll. If the location of these files is specified with drive letters, then the startup of BCS will fail. The command line parameters -b and -bm allow to execute a batch file before BCS itself starts. In such a batch file drive letters can be mapped as needed. BCS will try to execute the specified batch file and will wait until it terminates before continuing. With the parameter -b you can (and must) specify the filename of the batch file. This can either be a fully specified filename with a path or only the filename. If you omit the path, then the path from where BCS.EXE is being executed is used. The parameter -bm is only an abbreviation for: -b:"Mapping.bat". Some hints: you cannot use both, -bm and -b at the same time. You cannot execute an arbitrary executable. It must be a batch file. Have a look into the log of BCS. Immediately after the startup line there is a line, which reports about the started batch and whether it was a success of whether an error occurred.
-install -i	-	Can be specified if BCS is executed from a command line which must be opened with "Run as administrator" ! BCS 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: -key, -config, -account, -password, -service. The new service will be called "BroadcastServer" if you do not specify a key option and no config option (meaning the default is used). If you specify a key, it is appended to the service name after an underscore, 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: BCS –i –key"MyKey" –config"MyConfig" The created Windows service will use this name: "BroadcastServer_MyKey-MyConfig".
-uninstall -u	-	Uninstalls BCS as a Windows service. In order to uninstall a service, which has been installed with a key and/or config name you must specify the options –key and/or –config, too.

-account	username	If you install BCS 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 for BCS. BCS needs to have access to files like: its own XML files for all metadata, 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 BCS. 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).
-password	password	
-service	-	This option is automatically used if BCS installs itself. This option signals BCS that it is started as a service and therefore should never be used otherwise. The impact of this option is very limited. BCS will detect that it is 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.
-startdelay -sd	time[seconds]	This parameter is intended as help for starting two BCS programs within a buddy cluster. There it is quite important that both programs do not start at the same time, because otherwise it can happen that both declare themselves as being in master mode. If it can happen due to organizational or technical reasons that both programs start at the same time, one of both programs (typically the intended slave/backup server) should be started with a delay. You can use other means to achieve this, but the start delay parameter might help. When running BCS as a desktop program you will see a box with a countdown before BCS continues with the start process. In this case you can use big times, too. The parameter works when running BCS as a Windows service, too. You only have to specify the -sd parameter when installing BCS as a service. But in this case the Windows service manager will complain that BCS could not be started if the delay exceeds something around 2 minutes. BCS will start nevertheless, but the message of the service manager might be disturbing.

6.4 64-bit BCS and WebSocket communication

This chapter describes two topics, the 64-bit version and the WebSocket communication, which are in parts independent but also related somehow. In addition, upgrading an existing installation to the usage of a 64-bit version and/or WebSockets is described.

Attention: 64-bit variants of all modules should NOT be installed in the regular Digas folder with all other 32-bit modules, but instead in a corresponding 64-bit folder. Nowadays this is typically: C:\Program Files\DigaSystem (in contrast to C:\Program Files (x86)\DigaSystem).

6.4.1 WebSockets

For the longest time, between 2000 when the development of DigaBroadcastSystem was started and 2020, the communication between BCS and its clients was done via "SeplProtocol". This protocol is based on TCP/IP and is a proprietary protocol. It uses a mixture of binary and text data. Optionally, both parts can be compressed. A typical message from BCS to a client or vice versa is separated into two parts. The first part is transferred as binary data. This is the data, which forms something like a remote-procedure-call – aka the function name and the parameters. The second part is transferred as BCS-XML-formatted UTF-8 strings. This is the "payload" data representing all the metadata, BCS can store and handle.

The problem with the old code, which handled the SeplProtocol, was that it used a very old programming model, Microsoft has only introduced for TCP/IP clients (!) and for compatibility with Windows 95. Therefore, it had serious limitations. It would have been a lot of work to convert this old code to 64-bit – if possible at all – when starting with a 64-bit variant. Therefore, it was decided to put the work into a modernized way of communication: WebSockets.

WebSockets (according to RFC 6455) are an invention by Google and are supported by all modern browsers. The standard defines a way how to transfer data in a message-oriented way. Splitting long messages into individual blocks (called frames) and setting them together is part of the standard. Text and binary messages are defined and for text UTF-8 is prescribed. Compression is supported, too, but the standard "per-message-deflate" is not supported so far, because it came too late, when a proprietary solution based on the zLib library had already been implemented. (This might change in the future). Making the communication secure via TLS/SSL is also supported. BCS uses the OpenSSL library for that task.

6.4.2 WebSocket message format

In WebSocket communication individual messages are exchanged between BCS and its clients. For a plain text (non compressed and non-TLS) communication you could see this clearly by monitoring BCS with a tool like WireShark. As messages are pure text, it is also possible to include them into the log output of BCS – in protocol level 5 - though BCS will print only the first 16k characters (and 256 Bytes of a binary message) in order to avoid exploding log files. Here is an example of such a message:

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateNodeRequest>
    <TransactionID>ABC123</TransactionID>
    <UpdateDataOnly>true</UpdateDataOnly>
    <UseOnlySpecifiedData>true</UseOnlySpecifiedData>
</UpdateNodeRequest>
<<<<DATA>>>>
<Element>
    <ID>00000000/00000067/00000004/00000064/0000006A</ID>
    <Title>Changed title!</Title>
</Element>
```

It is a message a client sends to BCS to update the title of the specified element. You will surely notice the strange format of the message. It is not a single XML but it is divided into two parts, which are separated by a line with the text "<<<<DATA>>>>". This special format was chosen for the backward compatibility with the SeplProtocol. As mentioned above, data is transferred in two parts - which is also true for the internal handling of these parts within BCS. It is by far the easiest and quickest way to simply concatenate the two parts in a way like this. Not all messages have a second part. The first part represents the remote-procedure-call with its function parameters and is always present.

If you are interested in more details: all available messages are described in BCS Programmers Manual.

6.4.3 64-bit BCS / Advanced tree lock

Since version 6.0 BCS is available as a 64-bit variant. The 32-bit variant is still available and especially needed for upgrading, see below. The 32-bit variant of BCS can use a maximum of 3 GBytes of memory. It is possible to reach that limit in many situations. That was the reason, why we did neither recommend nor support hosting multiple 24/7 services on a single BCS. One disadvantage of the old SeplProtocol code was also that beginning with 50 clients problems could occur. With 64-bit and/or WebSockets these problems are gone. In detail:

- With 64-bit the available memory for BCS is much bigger and there is no longer the limitation to host only a single service. BCS also automatically loads more data in memory during startup (e.g. more templates or pools). In addition, it is possible to configure the parameters "Future/OldDaysToKeepInMemory" in key "Memory" more generously.
- By using a new implementation for WebSockets (based on boost::asio) there is no more a limit of 50 clients before problems might begin. BCS does now start as many threads for handling client requests as there are logical CPU cores on the machine and uses IO completion ports as it is recommended by Microsoft for servers. Therefore, there is no more a known limit for the number of connected clients. A limit is only reached, when either the network throughput is not sufficient or when the available computing power is not sufficient to serve all clients with an acceptable performance.
- As it is now possible to hold more data in memory and to serve more clients an additional improvement was made. In older versions than V6 (and also in the 32-bit variant of V6) there is only a single internal lock, which prevents simultaneous access by more than one client. This is also true for a pure read access and it means that only a single thread can work with the BCS tree at a given time.
  Now the 64-bit variant offers a better tree lock, which has two advantages: First, it is a multiple-reader/single-writer lock, meaning: only a write lock is exclusive but multiple reading clients/threads at the same time are possible. Second, the whole BCS tree can be locked on multiple levels. Not all existing levels are possible, but some important ones, e.g. for the root, a program/service, a jingle group or a day. This allows that even multiple writing clients/threads can access the BCS data tree at the same time if they are operating on different areas of the tree.
  This feature needs to be activated with the parameter:
  BroadcastServer\<Config>\General\UseAdvancedTreeLock = yes
  If this parameter is set to "no" or is not present, BCS falls back to the single/global lock for its data tree. Therefore, in case you encounter any problems, you can still go back to the old behaviour. You should also know that the advanced lock is a little bit more time consuming than the simple lock. Therefore, it is recommended to use it only if it brings an advantage. It is not possible to give a hard limit here, but in order to give some hints: if you have many clients (more than 30 or 40), if you have a lot of clients, which only want to have read access, or if you want to serve multiple services on a BCS.

6.4.4 Upgrading an installation

As described above, the old code handling the SeplProtocol was not ported to 64-bit. Therefore, the 64-bit version of BCS (BCS64.exe) does not support the old SeplProtocol. This makes it necessary to follow a specific upgrade path until using the 64-bit variant. This path is:

1) As a first step it is recommended to edit the settings of BCS so that BCS directly offers a WebSocket communication after the upgrade in step (2). This allows you as administrator to play around with the new communication, while all existing clients are still using SeplProtocol. You need to configure BCS to open one of the following ports:
   - BroadcastServer\<Config>\IP\WSPort
   - BroadcastServer\<Config>\IP\WSSPort

   The first one is for a plain-text communication, the second one for a secure TLS/SSL communication.
2) Upgrade existing BCS servers to version 6.x in the 32-bit variant. Run them for some time to make sure that there is no problem with this version (independent of the 64-bit implementation or WebSockets).
3) Special attention needs to be payed to multiple BCS servers communicating with each other. This are master/buddy clusters or servers with remote links. In principle, the 6.x version is backward compatible when using the old SeplProtocol. Nevertheless, it was always recommended to have all BCS servers on the same version number – or at least not very different ones. This is also true in this case. Therefore: don't run V6 and older BCS servers together for a longer time.

   One special problem remains: due to flaws in the old implementation it was not possible to make generic broadcast messages fully backward compatible. This type of messages is used when two BCS clients wants to communicate to each other but do not have a direct connection. BCS will relay these messages to all connected clients and the designated receiver will react on it. This is used e.g. for triggering BUS tasks from DigAIRange or to send macro commands between TurboPlayers. In case these generic broadcast messages must be transferred to a remote BCS this only works reliable if all affected BCS servers have version 6.x. Otherwise, it can happen that a message is not transmitted at all or multiple

times. Here it is strongly recommended to upgrade all affected BCS servers to a version 6.x as quick as possible. Master/buddy clusters are not affected by this special problem with generic messages.

4) Next you can upgrade **all** clients to a WebSocket-compatible version. Mainly, this are 6.x versions (but IBCS e.g. does not follow the main version number). This can take as much time as you like. You can run them for some time to make sure that no problems with the new versions occur.

5) Then you can change the general communication settings of the clients to use WebSockets. In the registry this are the parameters:

- Digas\PlanServer\<Config>\WSPort
- Digas\PlanServer\<Config>\WSSPort
- Digas\PlanServer\<Config>\BuddyWSPort
- Digas\PlanServer\<Config>\BuddyWSSPort

This is for master/buddy and for plain-text and TLS/SSL. You can also use the dialog "Server settings" of DigAIRange, see below. Have in mind that there is a hard-coded order in which the parameters are used. Secure WebSockets are preferred against plain-text WebSockets, which are preferred against SeplProtocol. When connecting to a BCS, clients can automatically fall back to a buddy connection, but they do not change from a WSS to a WS to a SP connection. On one hand it would take too long to try out up to 6 different options, on the other hand it is a simple configuration error if the client tries to use a protocol, which is not configured for BCS. Therefore, there is no fallback for the protocols

Important: if your clients run on computers with a different DigaSystem registry and connect to one BCS nevertheless, you might need to adjust the login method used by the clients. Reason is that when performing a login via WebSockets, the clients use an HMAC method, which requires the same registry on both sides of the connection (or at least the same Admin password). For more information please see: Login to BCS and BUS

When the clients reconnect for the next time, they will use the new protocol. In the client view of BCS there is a new column "ComType", which informs you about the protocol. Possible values are:

- SP — SeplProtocol, uncompressed
- WS — WebSockets, plain-text and uncompressed
- WSS — Secure WebSockets via TLS/SSL, uncompressed
- SP-C — SeplProtocol, compressed
- WS-C — WebSockets, plain-text and compressed
- WSS-C — Secure WebSockets via TLS/SSL, compressed

As long as you have clients using SeplProtocol, you cannot switch to a 64-bit variant of BCS. One possible "client" is the BCS to itself, if you have configured a regular "ServerSocketSelfCheck". Once again, run the system for some time to watch for any problems.

6) Special attention is necessary for remote links. If you open the properties of a show- or track-link in DigAIRange you will notice the new combo box on top of the dialog "Type of connection". For old links it will be "SeplProtocol". This needs to be changed to "WebSocket" or "Secure WebSocket". In addition, you might need to adjust the used port number accordingly. If you are using a different port number you should also enter new login credentials for the remote links in the registry in key "BroadcastServer\<Config>\ RemoteLoginData" and adjust master/buddy settings in "BroadcastServer\<Config>\ RemoteBuddyData". Don't forget existing remote links in your templates - in BCS and link templates for DigAIRange in registry! If you don't want to or cannot change all existing remote links at once, you can make use of the registry value: BroadcastSystem\<config>\RemoteLoginData\<address>:<port>\UseInstead. It allows a translation of old data using SeplProtocol into new data using WebSockets. For more information, please see the description of this parameter.

7) Last, you can upgrade your BCS servers to the 64-bit variant BCS64.EXE (including DigaSQL64.dll). Unlike in steps (2)/(3) it is not as critical to have all servers at the same time up to 64-bit. As long as WebSockets are used for the communication, both variants are compatible and a mixture of 32-bit and 64-bit BCS servers is supported. This is also true for the XML files on harddisk. In both variants of BCS they are stored as UTF-8 text file. Therefore, it is always possible to go back to a 32-bit variant in case of problems. Nevertheless, the long-time goal should be to have all servers running as 64-bit variant.
If you are using DigaMOS, please see: DigaMOS and BCS 64 Bit

A limitation is that the usage of a Plugin-DLL (see extra chapter below) is not supported so far in the 64-bit variant. If you need it, please contact DAVID Systems GmbH. There is also no support for DigaReportServer, but this product is deprecated anyhow.

6.4.5 Configuring Secure WebSockets via TLS

If you want to use secure WebSockets via TLS/SSL you first need to get a certificate for your BCS computer. You can get a certificate from a certificate authority, e.g. VeriSign, DigiCert or Let's Encrypt. It is also possible to create a self-signed certificate. This is possible, because at the moment the clients do not verify the certificate of BCS.

TLS is only used as transport encryption but not to make sure that a client is connected to the right server. (With the common in-house usage of BroadcastSystem this should not be an issue, anyhow.) In the future this might change and the clients will then check the certificate of the server. Then you will need to work with official certificates and/or install the required root certificate for the self-signed certificate on each client computer.

In order to activate TLS you need to configure the usage of the WSS ports on server and on client side. A configured TLS port will be preferred against a plain-text WebSocket port. On the server side you must copy the certificate files to a folder of your choice. BCS needs the certificate in form of two separate files in PEM format. PEM format means: a plain text format which begins with something like "-----BEGIN CERTIFICATE-----" or "-----BEGIN PRIVATE KEY-----". The actual file extension does not matter. Usually it is ".pem", ".crt" or ".cer" for the certificate file and ".key" for the private key file. If you only got a single combined file (e.g. a .pfx file) you can use the OpenSSL application to extract the two individual files with calls like:

```
openssl pkcs12 -in filename.pfx -nocerts -out key.pem
openssl pkcs12 -in filename.pfx -clcerts -nokeys -out cert.pem
```

See the documentation of OpenSSL or many examples in the internet for more information.

The two files must be specified in the BCS settings in key "BroadcastServer\<Config>\IP" in value "TLSCertificateFile" and "TLSPrivateKeyFile ". If the private key is protected with a passphrase you need to specify the passphrase in value "TLSKeyPassphrase". This last value must not be entered in plain text, but you need to call the dialog "Change crypted value" in Admin.exe and select the crypting method "Broadcast Server".

BCS will open an additional port for the secure WebSocket communication and will offer TLSv1.2 and TLSv1.3 to its clients.

6.4.6 BCS and OpenID Connect / JSON Web Tokens

OpenID Connect (OIDC) is a suite of definitions and protocols, which is used by modern web applications for authentication. WebDigAIRange can make use of it, too. (The classical desktop and backend applications do not use it. They use an HMAC method or – if older – a password encryption with the RC5 algorithm.)

There are a lot of details, but in principle the idea is that a web application lets an "authentication server" handle all the work to identify and authenticate a user. After successfull authentication, the web application gets a token back, which can then be used for accessing so-called "resource servers". BCS would be such a resource server. A resource server is able to verify the token by internal operations. There is no need to contact the authentication server for the verification.

The token format used for OIDC is called: JSON Web Token (JWT). It is simply a long string of characters. The string is separated into 2 or 3 parts, separated by dots and each part is encoded with the Base64Url algorithm. The first and the second part are human-readable JSON strings, containing information about the user name, the issuer of the token, an expiration time, the algorithms used for signing the token and optional other information. The third part is a signature of the first two parts. This signature allows to verify the token. The third part is optional, but if it is not present, the token cannot be verified. BCS does not accept unsigned tokens. If you are interested in more details, a good starting point is the web page: https://jwt.io

For the signature asymmetric cryptography is used, normally. The most common method is the classical RSA algorithm (Rivest–Shamir–Adleman, see: https://en.wikipedia.org/wiki/RSA_(cryptosystem)) with private/public keys, based on factoring large prime numbers. In case of OIDC, the private key must be kept secret and is only known on the authentication server. On contrast, the public key must be known to everyone who wants to verify a token. Therefore, authentication servers have a standard interface, which allows to get their public key. This is an http(s) URL for downloading the key. OIDC also specifes a standard interface called ".well-known/openid-configuration". This is the primary source of information for an authentication server. There you get a JSON file, which lists the endpoints/URLs for the different services. Retrieving the public key is only one of them.

OIDC defines a format for the keys, too. It is called JSON Web Key(s). It can be a single key or multiple keys in one document. For multiple keys the format defines a JSON array of multiple individual JSON Web Keys. Why multiple keys? Authentication servers can make use of multiple ones for different purposes or can iterate among them as they like. This makes it necessary to make the keys identifiable. Therefore, each key and each token has a property called "kid" (key identifier). This allows to pick the right key when a token should be verified. Authentication servers can also generate new keys from time to time. In order to stay up-to-date, BCS automatically re-queries all keys once per day (in the first 10 minutes after midnight).

Normally, you do not need to know the format of JSON Web Keys because you do not come into contact with them. As long as the RSA method is used for signing the tokens, BCS can query the keys automatically from the authentication server. There is one (so far theoretical) exception: Instead of the RSA algorithm, it is also possible to use a symmetric algorithm for signing the tokens. In this case a common secret must be known to the authentication and the resource server. In order to let BCS know the common secret, you have to define a JSON Web Key manually. This a simple JSON text file, which would look like this:

```
{
    "kty": "oct",
    "alg": "HS256",
    "kid": "123456",
    "k":   "U2VjcmV0"
}
```

They key type (kty) must always be "oct". The algorithm (alg) can be: "HS256", "HS384" or "HS512". The key identifier (kid) can be an arbitrary value, but it must be unique among all used keys. The common secret must be given in "k" and it must be encoded with Base64Url, which can be done with multiple web pages in the Internet. (In the example it is "Secret".) The same information must be known to the authentication server, too.

BCS supports the verification of OIDC tokens since version 6.0 – but only for clients, which already use the new WebSockets communication. As authentication server all OIDC-confirming servers should work. So far, we have tested only with our own DigaSystem Identity Server (part of DPE) and with ADFS (Active Directory Federation Services of Microsoft). If the authentification server is ADFS, a login with an Active Directory user will be done. This requires that these users are synchronized with the DigaSystem users (which is done by tool ADSync). These users have a long name "Domain\User" in their properties and you will also see this format with a domain name as the logged-in user in BCS.

Configuration:

The only thing that needs to be configured to allow BCS to verify the tokens is the locations of the JSON Web Key(s). This must be done in the subkey "JWK" of BCS (e.g: "BroadcastServer\Default\JWK"). Create a new value for each location. The name of the value does not matter, but do not use a preceding underscore, because this will be filtered out by BCS in order to allow temporary disabling of values. The data itself can either start with http: or https:. In this case BCS will try to download the public keys from the specified URL. You can either specify the URL for the openid-configuration or directly the URL for retrieving the keys (the "jwks_uri"). Alternatively, you can specify the name of a single file with a JSON Web Key or a directory, containing one or multiple JSON Web Keys in files with extension ".jwk".

6.4.7 BCS and DPE Tokens

Since version 6.1.430.0 BCS supports the login with a token of DPE (DigaSystem platform evolution). So far, DPE uses two different types of tokens, the classical ones and secure ones. The classical tokens contain the name of the DigaSystem user and the specified password. BCS can convert such a token directly into the username and the password and will use DigaSQL.dll for verifying the authentication. Therefore, no external verification is necessary.

This is different for secure DPE tokens. They do not contain any password, but only a secure hash value as a signature. The verification of such a token is done by DPE itself. Therefore, BCS has to make a call to a DPE rights service in order to verify a secure DPE token. You need to configure the API BCS can use for this verification. This is done in the registry parameter:

BroadcastServer\...\General\DPETokenUrl

The minimum required content is the protocol (http or https) and the host name. E.g. "https://mydpehost". Optionally, you can specify a port number and/or the path for the call. Default for the port numbers are 80 for http and 443 for https. Default for the path is "/DpeWebApplication/api/token". BCS will use the POST method on this API.

In order to work, the specified DPE service must be running and must be reachable by BCS, otherwise a login with a secure DPE token is not possible. So far, this type of login is only used by web clients. The desktop/backend programs do not make use of it.

6.5 Logging

6.5.1 Version smaller than 5.4

BroadcastServer writes all log information to the directory, which is specified in the parameter “Paths\WorkPath”. The log files have the name “SSRVMMDD.LOG” whereas MM is the month and DD the day of logging.

All different type of information is written to one type of log file. Nevertheless, you can increase or decrease the amount of information with the parameters “General/ProtocolLevel” and “General/ScreenProtocolLevel”.

If a log file reaches a certain size, a new log file is started which has the same filename but a continuous number appended. The size can be specified with the parameter “General/MaxLogfileSizeKB”. After n days log files are deleted. n is specified in the parameter “General/ProtocolMaxAge”.

6.5.2 Version 5.4 or higher

The old logging mechanism is no longer used. Instead, the logging is now done with the help of DigaSQL.dll. Therefore, all features of the dll can be used, like: specifying a max size per file, cyclic writing to files, automatic deletion after some days. For more information, please see the description of the “Protocol” key in the document “DigParam.rtf”.

BroadcastServer uses the log destination “BCS-Log” which must be configured as value in the “Protocol” key. At least, this is true if you run the standard registry set and the configuration “Default”. If you start BCS with a different registry set (-key command line option) or a different registry configuration (-config command line option) the name scheme is somewhat different: a registry set is appended to “BCS” with an underscore, a registry configuration is appended to “BCS” with a minus. Example: The registry set “MyKey” would use the destination “BCS_MyKey-Log”. The registry configuration “MyConfig” would use the destination “BCS-MyConfig-Log”.

The parameters “General/ProtocolLevel” and “General/ScreenProtocolLevel” are still valid. Logging to the screen will completely be suppressed if BCS runs as a Windows service and no remote GUI is connected.

If you run BCS as a Windows service, the event log of Windows (type application) will be used in addition. Most of the information is still written to the specific log files. But in case of startup or shutdown problems the event log is used. This is especially necessary if a problem is encountered before the DigaSQL logging has been initialized. Therefore, you should always check both, the log files and the event log, in case of startup/shutdown problems.

Special hint: you might encounter the problem that the log file of BCS is empty or contains only some lines of the startup but nothing else. This problem can occur if you have defined login credentials in key "DigaSQL", parameters "DigaSqlUser" and "DigaSqlPassword", which are invalid. In most cases you do not have any disadvantages from the wrong credentials, because BCS requires a DigaSQL login in very few cases. Therefore, the wrong credentials remain unnotified, normally, but they can interfere with the logging.

6.6 Remote GUI

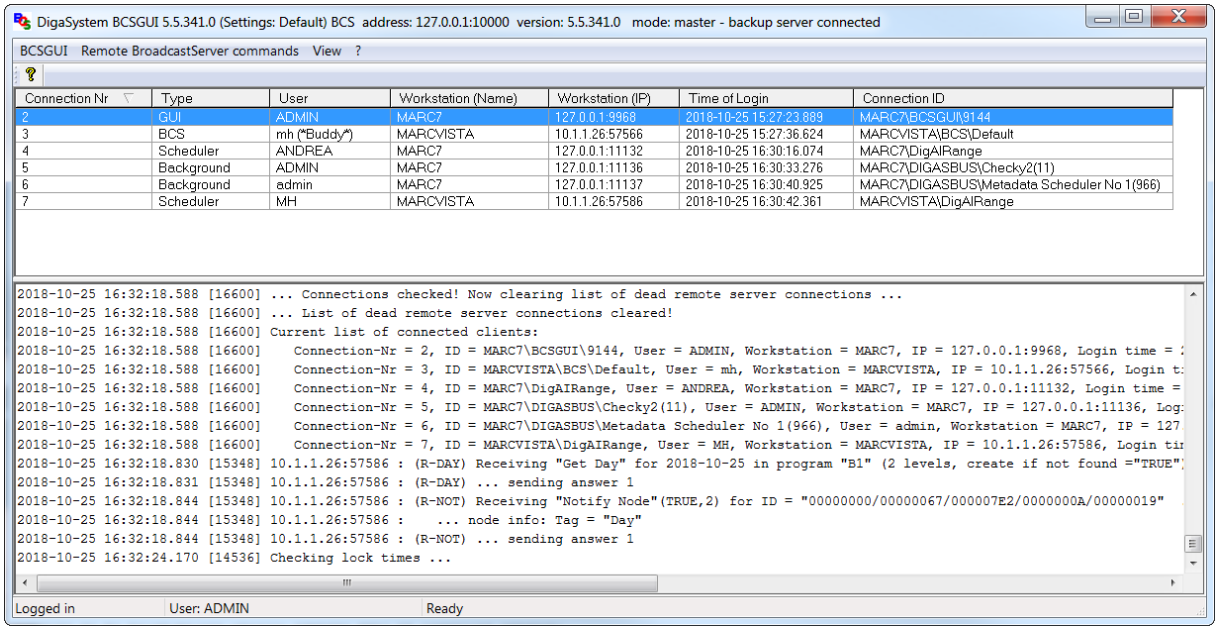
If you run BCS as a desktop or sys tray application a GUI is available to monitor and control BCS. This is not the case if you run BCS as a Windows service. Two modules are available which provide a remote-GUI for a BCS – one is BCSGUI and the other one is a dialog of DigAIRange. Hint: the remote GUIs are intended to be used with a BCS running as service, but they work for a desktop/systray BCS, too.

6.6.1 BCSGUI.EXE

BCSGUI is intended to be used on the same computer on which a BCS is running as a Windows service. There are two reasons for this limitation: first, BCSGUI can connect to a single BCS only. For connecting to multiple BCS servers you can use DigAIRange, see below. Second, the connection parameters are read from the local key “BroadcastServer\<config>\IP” and “…\Buddy”. These are the same parameters as BCS itself reads. The command line parameters “-config” and “-key” can be used, too. Switching between a master and a buddy server (if you have a buddy configured) can be done with a menu command. The login account is queried during startup and cannot be changed later. The user must either be “ADMIN” or the user must have the manage right for the BCS.

If you really want to connect the GUI to a remote BCS, you can use the command line parameter "/BCS". It has the format: "/BCS=<protocol>:<address>:<port number>". Example: "/BCS=WS:OurServer:10001". If you omit the protocol, the 64-bit version uses "WS" by default, the 32-bit version uses "SP" by default. If you omit the port number, the default for "SP" is 10000 and for "WS" it is 10001. The specification of this command line parameter overrides the settings from the registry. You can only connect to the master or to the buddy at startup, but it is no longer possible to switch between the two of them during runtime.

BCSGUI looks mostly like BCS. You will see only minor differences:



In the header, you can see “BCSGUI” instead of “Broadcast Server” and at the end a hint is printed if there is no connection or if the connection is disturbed. If the GUI is connected and logged in, the address, port number, version number and the current mode of BCS is printed.

The menu “BCSGUI” can be used to terminate the program or to change the connection between master and buddy server (if available). Most menu commands of BCS can be executed remotely with the menu “Remote BroadcastServer commands”. The menu “View” can mainly be used to control the protocol view:

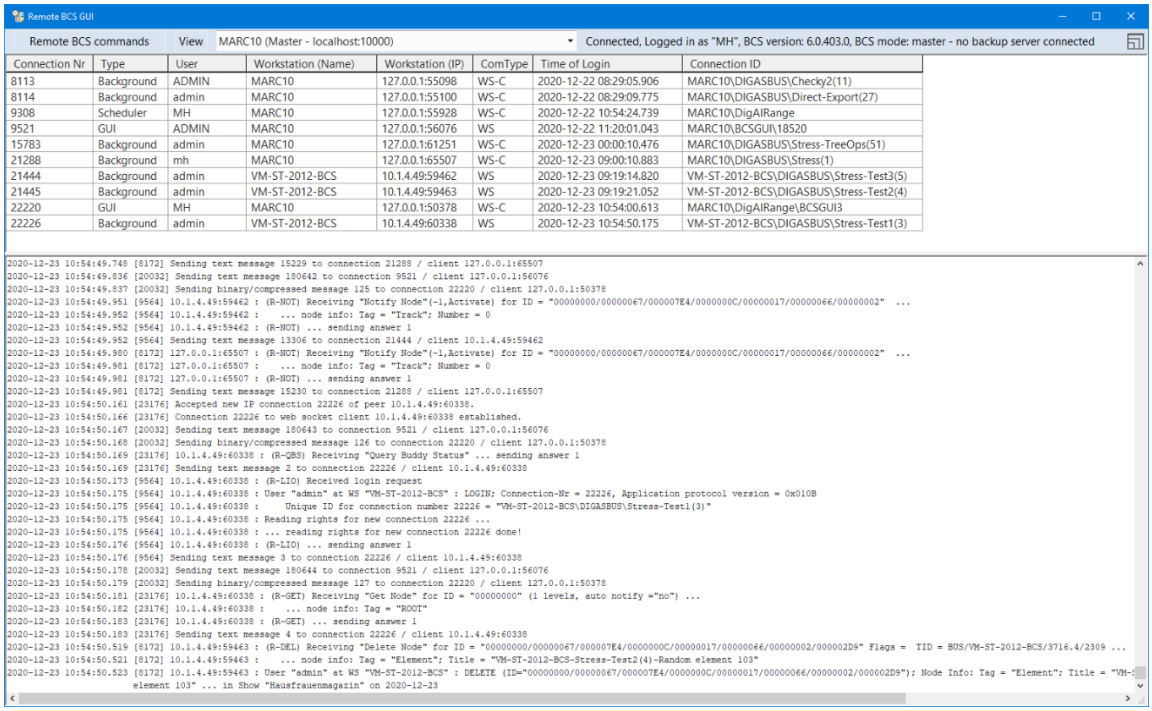
- “Lock protocol view”: log messages are still received but are buffered internally. If the view is unlocked, these buffered lines are printed. A maximum of 10000 lines are buffered. If the buffer size is exceeded, log messages are lost.
- “Transfer of protocol from BCS is active”: this menu item can be used to tell BCS to no longer send any log messages. Using this option will generate gaps in the log output, of course. Nevertheless, this option is very useful in order to reduce the network traffic of BCS. Receiving all log output might be nice but it does generate much network traffic, which can affect the other clients. Therefore, it is recommended to switch logging off whenever you do not need it. You can also consider watching the log indirectly by opening the log files of BCS with a log viewer.

The list with all connected clients is identical to BCS. You also have the possibility to disconnect a client by using the context menu command “Disconnect”. It is possible to disconnect the connection of the own GUI but as the GUI will try to reconnect automatically in the background you might not notice a visible reaction.

Since version 6.0 BCSGUI is also available as a 64-bit variant: BCSGUI64.EXE.

6.6.2 BCSGUI in DigAIRange

In order to have a remote GUI for multiple BCS servers in a network a special window is available in DigAIRange since version 5.4.701.0. You need to be logged in as user “ADMIN” or as a user who has the manage right for a BCS; otherwise, you cannot open a remote GUI. In the tree view click onto a server symbol and call the context menu commands “BCS remote GUI for master” / “… for buddy”. It is not necessary that the connection to the BCS was already established. The new window needs to open a new connection, anyhow. If the selected server is online, a window like this should appear:



The window is a non-modal window. You can open multiple ones and you can move them around on the screen or use the window menu button on the right side to position the window. The position of these windows is not saved. If you log out or log in as a different user, the windows are closed.

The combo box on the top displays the current connection. If you open the box, a tree appears which allows you to select any of the configured BCS servers. The servers, which appear, are identical to the tree view of the main window but in contrast to the main window, you can distinguish between master and buddy server. If no buddy server is configured, you will see only a single entry.

The status text on the right side of the combo box shows the current connection state. If a connection is active the version of the remote BCS is displayed and the current mode – whether the BCS is running as master or as backup server and whether a backup/buddy BCS is connected.

With the button “Remote BCS commands”, you can execute some of the menu commands a BCS provides, too. With the “View” button, you can limit the log output. See “BCSGUI.EXE” above for an explanation of the options. If you right-click on one or multiple of the connections in the connection list, it is possible to disconnect clients.

In case the connection to the BCS is lost, the remote GUI will automatically try to reestablish it. Please be aware that the remote GUI always uses its own connection but not the connection DigAIRange would regularly use. This is necessary because the regular connection is for the cluster of a master/buddy pair only. With such a connection, it would not be possible to select a dedicated server within the cluster.

6.7 Settings

6.7.1 Parameter description

The following parameters can be configured. Settings marked with an (*) can be changed at runtime (see the chapter “Operation” below).

Parameter (Key/Value)	**Meaning**
Paths/WorkPath	Directory, which is used by the BCS for protocol (only up to version 5.3) and temporary files. **Default:** directory of BCS.exe
Paths/SeplTreeRootDir	Base directory for the storage of the complete broadcast data tree. **Default:** subdirectory "SeplServTree" below *WorkPath*
General/HideToSysTray (*)	Set to TRUE if you run BCS as a desktop application and want it to hide itself to the Windows SysTray. **Default:** FALSE
General/ServerNameForGlobalRights	In the rights system you have right objects of type "Showplan", which do not have the name of a service in their first part, but a server (BCS) name. If you have a single BCS server (no cluster) this name can typically be the computername or hostname of the BCS computer. But if you run BCS in a cluster of two buddies, you would have to create the right objects two times, once with every computername. To avoid that, we have this parameter. Here you can give a name, which is used by BCS on both computers of a cluster. By configuring the same name you have to create the right objects only once, with this name. The name you should use is not the computer- or hostname of one of the two computers. You should use the same name as the BCS cluster is listed for its clients in the registry in key "Digas\PlanServer". Reason: right objects of type "BCS" are verified by both, BCS and clients. BCS uses the configured "ServerNameForGlobalRights", whileas clients use the name of the "PlanServer" in the registry. At the moment, the following right objects use the server name from this parameter: Type Showplan: <ServerNameForGlobalRights>\GeneralPools <ServerNameForGlobalRights>\GlobalJingles Type BCS: <ServerNameForGlobalRights> **Default:** computer name of the server
General/RegKeyLocationDisplayName	Display name of the DigaSystem registry location. **Default:** (none)
General/ProtocolLevel (*)	Level of the protocol as written to the log file: 1=normal, 3=debugging, 4/5=extended debugging; for "normal" operations, levels >= 4 are not recommended (performance problems!) **Default:** 1
General/ScreenProtocolLevel (*)	Level of the protocol as displayed on screen: 1=normal, 3=debugging, 4/5=extended debugging; for "normal" operations, levels >= 4 are not recommended (performance problems!). **Default:** 1
General/ProtocolMaxAge (*)	Age in days, at which protocol files will be deleted. **Default:** 30
General/MaxLogfileSizeKB (*)	Maximum size of a single log file in kilobytes. If this size is exceeded, a new file is started (at the end of each day, a new file is started anyway). A value of 0 means that a new file is only started at the end of each day. **Default:** 0
General/DayReloadingThreadPriority (*)	Relative priority of thread which reloads “day nodes” from disk into memory (-6 (lowest) … +7 (highest)). **Default:** 0

General/WriteCleanScreenProtocol (*)	If this is TRUE, UTF8 character sequences in the on-screen log are converted to the local code page. This makes a better output in the presence of special characters, but slightly lowers the performance. The protocol file is written as UTF8 anyway, and is correctly displayed in all current text editors, including Notepad.exe and DigaSystem's LogView.exe. **Default:** FALSE
General/AutoRestartAfterCrash (*)	If this is TRUE, the BCS will try to restart itself in case of a fatal crash. Depending on the exact nature of the crash, this may or may not work. **Default:** FALSE
General/ConfirmExitRequest (*)	Defines whether BCS shows an exit confirmation box or not. **Default:** TRUE
General/FieldsToLogOnInsert (*)	A comma-separated list of BCS field names (technical names, i.e. the names as they show up in the XML tags), which are logged in the BCS protocol when a client inserts a new node. **Default:** (empty)
General/UseAdvancedTreeLock	By default, BCS uses a single and exclusive lock for the whole data tree. In the 64-bit variant of BCS it is possible to activate an advanced lock with this parameter. It provides multiple-reader/single-writer locking and multiple individual locks on different levels of the data tree. Therefore, multiple clients can read and/or operate on the data tree at the same time. It is recommended to activate the advanced lock if you have many clients (more than 30-40) and/or are hosting multiple services on one BCS. **Default:** FALSE
General/DPETokenUrl	When a client tries to perform a login with a secure DPE token, BCS does not verify the token itself. BCS has to connect to DPE rights service in order to verify the token. Therefore, you have to configure the corresponding URL in this parameter. The minimum required content is the protocol (http or https) and the host name. E.g. "https://mydpehost". Optionally, you can specify a port number and/or the path for the call. Default for the port numbers are 80 for http and 443 for https. Default for the path is "/DpeWebApplication/api/token". BCS will use the POST method on this API in order to verify DPE tokens. If this parameter is not configured, a login with a secure DPE token is not possible (but a login with a classic DPE token will still work). **Default:** <empty>
General/ WriteMemoryDumpBeforeRestart	If this parameter is set to TRUE, a full memory dump of the BCS process is written to the current working directory before BCS needs to do a restart because a serious internal error was detected. (In case BCS runs as a Windows service, the dump is written to the Windows system directory.) To do this, a version 2.6.0.0 of Restart2.exe or newer and ProcDump.exe (from the SysInternals suite of Microsoft) is required. For BCS64.exe use ProcDump64.exe to create the dump files. These programs must reside in the working directory, in which BCS is located. **Default:** False
General/Version	For internal BCS use – do not change!

IP/ServerPort	TCP/IP listen port of the BroadcastServer. Not supported in 64-bit BCS! **Default:** 10000
IP/WSPort	The port number for the listen port for plain-text WebSocket communication with clients. **Default:** 10001 (only 64-bit BCS)
IP/WSSPort	The port number for the listen port for secure WebSocket communication via TLS/SSL with clients. **Default:** 0
IP/TLSCertificateFile	If you want BCS to open a secure WebSocket port via TLS you need to specify a certificate file in PEM format with this value. PEM format means: a plain text, which starts with something like "-----BEGIN CERTIFICATE-----". Typical extension are .pem, .cer or .crt **Default:** <empty>
IP/TLSPrivateKeyFile	If you want BCS to open a secure WebSocket port via TLS you need to specify a private key file in PEM format with this value. PEM format means: a plain text, which starts with something like "-----BEGIN PRIVATE KEY-----". A typical extension is .key **Default:** <empty>
IP/TLSKeyPassphrase	If you want BCS to open a secure WebSocket port via TLS and you have specified a private key file with the parameter TLSPrivateKeyFile and if this private key has a passphrase you need to specify this passphrase here. Do not enter a plain text! Use Admin.exe, open the dialog "Change crpyted value" and select the crypting method "Broadcast Server". **Default:** <empty>
IP/ServerAddress	Attention: this parameter is only used by BCSGUI, not by BCS itself ! This parameter can be used to define an alternative IP address to which BCSGUI connects. **Default:** localhost / 127.0.0.1
IP/RemoteServerConnectionTimeout (*)	Timeout (in milliseconds) for connection attempts to remote BCS servers. This should not be too high, because during a connection attempt to a remote server, the BCS generally blocks any other client operations. **Default:** 5000
IP/RemoteServerCommunicationTimeout (*)	Timeout (in milliseconds) for communication with remote BCS servers. This can be higher than *RemoteServerConnectionTimeout*, because during normal communications, the BCS does not block other client operations (except for those operating on the same remote server, of course). **Default:** 20000
IP/DeadRemoteServers (*)	The parameter can be used to define a list of remote BCS servers (format “IP1:Port1;IP2:Port2;…”) which are known to be unavailable for a longer period (e.g. because of network shutdown). The BCS will never attempt to establish a connection to such a “dead” server, preventing unnecessary timeouts when clients access a remote node linked to a “dead” server. **NOTE:** The entries refer to single servers, and not Buddy Systems. Therefore, if a remote host is a Buddy System, and both servers are “dead”, both have to be entered into the list. **Default:** (none)

IP/KnownRemoteServers (*)	The parameter can be used to define a list of remote BCS servers (format “IP1:Port1;IP2:Port2;…”) which are known to exist. The BCS will try to establish and keep open a permanent “dummy” connection to any “known” server, and will therefore immediately know when such a server goes down or becomes unavailable. A server, which has become unavailable, will be temporarily treated just like a “dead” server as defined by *DeadRemoteServers*. However, the BCS tries to reconnect to the unavailable server every 5 seconds (unless that server is explicitly included in *DeadRemoteServers*). If successful, the remote server is again treated as “normally available”. **NOTE:** The entries refer to single servers, and not Buddy Systems. Therefore, if a remote host is a Buddy System, both servers must be entered into the list. **Default:** (none)
IP/CompressXmlOnRemoteConnections (*)	If TRUE, compressed XML data will be transferred between any BCS-to-BCS connection, which this server initiates. **Default:** FALSE
IP/ServerSocketSelfCheckInterval	Since build 265 BCS can start a background worker thread which performs a regular self connect via TCP/IP in order to check that the IP stack is working correctly. If the connect fails, BCS will restart itself. For self-restarting the helper “Restart2.exe” is necessary. NOTE: Such a self-restart will make the BCS unavailable for some minutes. Therefore, this feature is primarily intended for a usage in a master/buddy system. The Interval parameter is the time between two connect attempts in seconds. Default: 0 – meaning: the self-connect is not performed. The Address is given in parameter "ServerSocketSelfCheckAddress". Please check also the parameters "ServerSocketSelfCheckMaxDuration" and "ServerSocketSelfCheckMaxFails". **Default: 0**
IP/ServerSocketSelfCheckAddress	See parameter "ServerSocketSelfCheckInterval". Here the TCP/IP address to connect to is specified. Default is 127.0.0.1 which is also the only sensible value, except your BCS is multi-homed, aka has multiple network interfaces. The port number and the protocol BCS uses cannot be configured. If multiple listen ports are open, BCS has the usual priority: Secure WebSocket (TLS); if not configured then plain-text WebSocket; if not configured then SeplProtocol. **Default: 127.0.0.1**
IP/ServerSocketSelfCheckMaxDuration	Since build 269 the time needed for running a self-connect check is measured. If it exceeds the time given here (in milliseconds) a restart of BCS will happen, but only if this occurs more than n times, which is configured in parameter ServerSocketSelfCheckMaxFails. It is necessary to activate a self-connect check with the parameters ServerSocketSelfCheckAddress and ServerSocketSelfCheckInterval. If the value is set to 0 the time a self-connect needs is not evaluated. A reasonable time would be some seconds, e.g. 5000. **Default: 5000**
IP/ServerSocketSelfCheckMaxFails	Since build 269 it is possible to configure the number of failed self-connect tries (in sequence) until a self-restart is executed with this parameter. (It is necessary to generally activate a self-connect check with the parameters ServerSocketSelfCheckAddress and ServerSocketSelfCheckInterval.) If the value is set to 0 the self-check can fail endlessly and no self-restart will occur. **Default: 3**

IP/LoginSelfCheckUser	Since build 267 BCS can perform a regular dummy DigaSQL login in order to check whether the rights system of the DigaSQL.dll did not become corrupted. If a login fails for 3 times BCS is restarted. NOTE: This check is only an add-on to the server socket self check. Therefore this check must be configured first ! Both checks have the same interval. Pay attention to specify a correct user and password, otherwise BCS will permanently restart itself. The User parameter specifies the DigaSystem user for the dummy login. Default is: <empty>, meaning: no login check is done. The password is given in parameter "LoginSelfCheckPassword". **Default: <empty>**
IP/LoginSelfCheckPassword	See parameter "LoginSelfCheckUser". The Password parameter specifies the corresponding password. It must be entered with Admin.exe with the registry parameter editor “Registry string editor (crypted)” with the crypting method “BroadcastServer”. **Default: <empty>**
JWK/...	One or multiple values can be created for each location, where BCS can get JSON Web Keys for the validation of OIDC / JSON Web tokens. The name of the value does not matter, but do not use a preceding underscore, because this will be filtered out by BCS in order to allow temporary disabling of values. The data itself can either start with http:// or https://. In this case BCS will try to download the public keys from the specified URL. You can either specify the URL for the openid-configuration or directly the URL for retrieving the keys (the "jwks_uri"). Alternatively, you can specify the name of a single file with a JSON Web Key or a directory, containing one or multiple JSON Web Keys in files with extension ".jwk". **Default: <empty>**

Buddy/Address	TCP/IP address (or resolvable name) of the buddy server. **Default:** (none)
Buddy/Port	The communication port used between BCS and Buddy BCS if SeplProtocol should be used. Pay attention that both WebSocket port have priority. **Default:** 10000
Buddy/WSPort	The port number for the connection to the buddy BCS if plain-text WebSocket communication should be used. Pay attention that a configured secure WebSocket port has a priority. **Default:** 0
Budy/WSSPort	The port number for the connection to the buddy BCS if secure WebSocket communication via TLS/SSL should be used. This has a priority against plain-text WebSockets and against the old SeplProtocol. **Default:** 0
Buddy/TreePath	Path to the base directory for the storage of the broadcast data tree of the buddy server (only read access is necessary!) **Default:** (none)
Buddy/User	If you are using a Buddy BCS you have to establish a communication between both BCS. With this value you specify the user which should be used for the connection. If you do not specify a value, the user "ADMIN" is being used by default. Each other user you specify must have the manage right for the corresponding BCS rights object. **Default: ADMIN**
Buddy/Password	If you are using a Buddy BCS you have to establish a communication between both BCS. Enter the password of the user you specify with the value "User". You need to enter a crypted value with ADMIN.EXE and use the crypting method "BroadcastServer". **Default:** (none)
Buddy/IAmPreferredMaster	If this is TRUE, the BCS will switch itself to master mode if it's connected to a buddy server. **Default:** FALSE
Buddy/ReconnectRetries	Number of times a BCS running as a slave tries to reconnect to the master before declaring itself as master if the connection is lost. Please have in mind that these reconnect retries can heavily increase the time until a BCS becomes a master and accepts any clients. **Default:** 0
Timer/TreeSavingInterval (*)	Interval (in seconds) for updating of the BCS data on disk. **Default:** 60
Timer/BuddyFileCheckInterval (*)	This parameter controls a safety feature to prevent data loss in case of master/slave-switches of a BCS cluster. In case a running slave BCS cannot be synchronized when new days or shows are created on the master, the corresponding day directories or show XML files will be missing on the harddisk of the slave BCS. When making the slave the new master and then switching back to the original master, the synchronization on file level will delete the day directories/show XML files on the master, too - because they were never present on the slave. As workaround, a BCS, which is running as slave, can regularly check, whether the XML data tree of the master contains more dirs/files than its own XML data tree. If a discrepancy is detected and remains for some minutes, the slave triggers a self-restart. During startup a full synchronization on file level will then take place. This parameter specifies the interval in minutes of these regular checks. The value should be a multiple of the parameter TreeSavingInterval (converted to minutes, default for the tree saving interval is 1 minute). Setting the parameter to 0 switches the feature off. If no buddy is configured, the feature is not active. **Default:** 30

Timer/MaxNodeAgeInMemory (*)	Time (in seconds) after which unaccessed tree data will be removed from memory (for fine tuning, see Memory settings). **Default:** 300
Timer/TreeCompactingInterval (*)	Interval (in seconds) for removing data nodes from memory, whose last access was more than *MaxNodeAgeInMemory* seconds ago (for fine tuning, see Memory settings). **Default:** 600
Timer/ShortLockWaitTime (*)	If a client requests a tree operation and there are node locks set, the operation normally fails. In case the locks are short-time locks, which were not set by a user but by another client for protecting some quick operations, BCS can delay the tree operation request instead of immediately sending the answer that a node is locked. The maximum wait time until the sum of blocking short-time locks get released can be configured with this parameter. Pay attention to use a time which is significantly smaller than the timeout value for clients. Otherwise clients can easily run into timeouts. If you use the value 0 waiting for short-time locks to become release is disabled. **Default:** 5000
DigaReport/KeepAliveInterval	Interval (in milliseconds) for the Keep-Alive message sent to the DigaReport dispatcher. **Default:** 10000
DigaReport/MessageTimeout	Internal timeout (in milliseconds) when sending messages to the DigaReport dispatcher. **Default:** 5000
Memory/CriticalMemoryPercentage (*)	If the percentage of used physical memory exceeds this value, the parameters *FutureDaysToKeepInMemory*, *OldDaysToKeepInMemory* and *KeepDayTemplatesInMemory* are ignored, and as much old nodes as possible are removed from memory. **Default:** 95
Memory/MinimumFreeVirtualMemoryMB (*)	If less than the given amount of (in megabytes) of virtual memory is available, the parameters *FutureDaysToKeepInMemory*, *OldDaysToKeepInMemory* and *KeepDayTemplatesInMemory* are ignored, and as much old nodes as possible are removed from memory. **Default:** 128
Memory/FutureDaysToKeepInMemory (*)	Number of days in the future, which are never removed from memory (optimizing performance when planning for the near future). The current day is not counted, e.g. “1” means that the current day and tomorrow is permanently held in memory. **Default:** 1
Memory/OldDaysToKeepInMemory (*)	Number of days in the past, which are never removed from memory (optimizing performance when browsing past shows). The current day is counted, e.g. “2” means that the current day and yesterday are permanently held in memory. **Default:** 0
Memory/KeepDayTemplatesInMemory (*)	If TRUE, all day templates are permanently kept in memory. **Default:** FALSE
Settings/TextDurationMode	Defines a default for calculating the total duration of elements: 0: total duration = audio duration 1: total duration = audio duration + text duration 2: the value is read from the service (program) node in the tree. It can be set with the manage dialog of DigAIRange. If the value is not set in the service 1 is assumed. It is possible to change the calculation mode for each element; this value is only the default mode if nothing is defined for an element. **Default:** 0
Settings/KeepDayPool (*)	If TRUE, the contents of the DayPool of a day are preserved, when the shows of the day are automatically inserted from the day template **Default:** FALSE

Settings/AddDayTemplatePool (*)	If TRUE, the DayPool of a day template is also copied to a day, when the shows of the day are automatically inserted from the day template. If FALSE, the DayPool of a day template is not used. Note: If both *KeepDayPool* and *AddDayTemplatePool* are TRUE, the resulting DayPool will contain the data of both pools (from the day and the day template). **Default:** TRUE
Settings/FixedTimeInShowTolerance (*)	Fixed Elements (mode “StartOnTime”, “EndOnTime”) will be rejected when their times fall outside the limits of the show they are in. *FixedTimeInShowTolerance* is the number of seconds, by which the times are allowed to fall outside the show limits. E.g., with a setting of “10”, a show ending on 9:00:00 may have a last “EndOnTime” element with a stop time of 9:00:10 **Default:** 0
Settings/CurrentShowTolerance (*)	Some rights in the BCS system are defined to operate on the “current show”. This parameter defines an interval in seconds, which extends the logical duration of the current show at both ends. E.g. with a setting of “30”, the show from at 9:00:00 to 10:00:00 is considered the “current show” between 8:59:30 and 10:00:30. **Default:** 60
Settings/ReserveElementMaxGap (*)	Reserve elements are “activated” whenever a gap longer than *ReserveElementMaxGap* would occur. **Default:** 0
Settings/AllowRemoteShowWriteAccess (*)	If set to FALSE, no write operations on any remote nodes are allowed. **Default:** FALSE
Settings/CheckMOSControlledRundownOps (*)	In a rundown which is synchronized by DigaMOS some operations are not possible because the MOS protocol does not have the capability to reflect these changes back to the newsroom system. At the moment there is only one such operation: the insertion of an element directly on top-level of a MOS-synchronized track. If the parameter is TRUE the operation is denied. **Default:** FALSE
Settings/DayPoolsHaveNoTimeAdjustmentByDefault (*)	If TRUE, Day Pools don’t use automatic time adjustment by default. **Default:** FALSE
Settings/ShowPoolsHaveNoTimeAdjustmentBy Default (*)	If TRUE, Show Pools don’t use automatic time adjustment by default. **Default:** FALSE
Settings/ GroupClassGrantsAccessRightToAll Elements (*)	If this option is TRUE, a user with write rights for class X can manipulate (create, update or delete) elements with arbitrary classes in a group of class X. If set to FALSE, such a user can only manipulate elements of those classes for which he has write rights, regardless of the class of the elements’ parent group. **Default:** TRUE
Settings/MaxAgeForChangeableDays (*)	In days, which are older than this value (unit: days), no changes are allowed. These days are effectively read-only nodes. Also, such “old” days are not automatically created nor are templates automatically filled in. A value of 0 means that there is no maximum age for changeable days. **Default:** 0
Settings/MaxAgeForConsistencyCheckResults (*)	BCS can store the results of ConsistencyChecks as reported by BUS tasks in its tree. The deletion of old reports is done by BCS in the background. In this key you can specify the time after which old reports are deleted. The value must be given in minutes and if refers to the end of each reported show. If you specify the word "INFINITE", old reports never gets deleted. **Default:** 10
Settings/MaxRetriesForNotifications (*)	If the transmission of a change notification to a client fails, the BCS drops this notification. If the parameter is > 0, the BCS will retry to transmit the notification that many times. **Default:** 0

Settings/MaxErrorCountForNotifications (*)	After this number of consecutively failed notification transmissions to a single client, the BCS will forcible close the connection of this client. **Default:** 50
Settings/StartAttributeInheritance	This parameter decides how BCS handles the inheritance of the start mode and the start type form parent nodes (=groups, stories, endpoint-stories) to the children. The inheritance is done depending on the current value. E.g. a fixed start is inherited to the first child, whereas a fixed end is inherited to the last child. Other children are normally not affected (with the exception of the fixed end, which makes all previous children be backward-floating). The inheritance is done by using the special values "Time_OnAirStartType" and "Time_OnAirStartMode". This means that the original start type/mode is never overwritten and if the element is used in a different place it can get its old start attributes back. Older versions of BCS inherited only the start types: Fixed, FixedStart, FixedEnd, Variable, Relative and BackwardFloating, but not Floating. And the start modes: StartOnTime and EndOnTime. This old behaviour can be configured by setting this value to "Legacy". Since version 6.1 of BCS additional start attributes can be inherited. This is the start type Floating and the start modes Manual and Sequenced. (All of them are inherited to the first child.) This new behaviour can be configured by setting this value to "Regular". Hint: at the moment (January 2023) the value "Legacy" is still the default and it is recommended to use the new "Regular" only for testing. In the future it is intended to change the default to "Regular". **Default:** till version 6.4: Legacy, since version 6.5: Regular
Plugin/DLL	Path to a DLL which is loaded as a plug-in to check and modify newly inserted nodes. **Default:** (none)
Plugin/Parameters	A string which is passed to the plug-in DLL at startup. For the current “standard” DLL, this is the name of the settings subkey. **Default:** (none)
DigaSQL/DigaSQLUser	Some plug-in DLLs require a DigaSQL database connection. If this is the case, the user name for this connection can be configured here. **Default:** (none)
DigaSQL/DigaSQLPassword	The password for the user defined by “DigaSQL/DigaSQLUser”. **Default:** (none)

RemoteLoginData/*Server:Port*	If you use remote servers (see section Regionalization and Remoting), you must specify login parameters for each remote server. For each server, a subkey named “server:port” below key “RemoteLoginData” must be created. For “server”, either an IP address or a resolvable DNS name can be used (do NOT use a BCS name as it is used in the registry in key Digas\Planserver, this key is not evaluated by BCS). However, the name or IP must be in exactly the same format as used for the configuration of the remote links! In the subkey, the values “**User**” and “Password” must be created. To enter the password use ADMIN.EXE and enter the value with context menu command "Change crypted value" and select crypting method "BroadcastServer". If you want to connect to a remote server, which opens multiple ports (for SeplProtocol and WebSockets) you should create multiple entries here, one for each used port.
RemoteLoginData/*Server:Port*/UseInstead	This parameter is only intended for a temporary usage in order to better support an upgrade of BCS v5 -> v6! Here a problem might be that there are plenty of configured remote links on show or track level, which use the old SeplProtocol. If you then configure the usage of WebSockets for newer shows/tracks, you will end up with two BCS connections to the remote server - one using SeplProtocol and one using WebSockets. To avoid this situation, you can configure a replacement of the old connection info in this value. The info to be replaced is given by the key name in which this value is located (which has the format: <remote server address>:<remote server port>). The new connection info to be used instead is given in the data of this value in format: <protocol>:<remote server address>:<remote server port> For the protocol use one of: SP, WS, WSS. Using this value will not change the remote link info in the metadata. But whenever BCS wants to make use of a remote link, the data is first exchanged by the info given in this value and only then BCS tries to establish the remote connection. If you are using a master and a buddy server you have to configure both of them in two separate keys below RemoteLoginData.
RemoteBuddyData	For every remote server, a buddy can be configured, which enables transparent connections to remote hosts with a Buddy System. An arbitrary number of entries can be made inside this subkey. The IP address and port (Format "IP:port") of the primary host (which is also used to refer to the host system e.g. in DigAIRange when configuring a remote node) must be the name of a entry, while the IP and port of the Buddy must be its value (e.g. "10.1.1.24:10000"="10.1.1.20:10000"). NOTE: Log messages referring to remote connections to a buddy system will always quote the primary server (which is the means of identification of any specific remote system) even if the actual current connection is switched to the buddy server. If you want to connect to a remote server cluster, which opens multiple ports (for SeplProtocol and WebSockets) you should create multiple entries here, one for each used port.

6.7.2 Multiple BCS Instances on One Computer

NOTE: Multiple instances of BCS.exe running on a single system are not recommended in production environments. Please use such a setup for test and evaluation only!

By using multiple parameter sets, it is possible to launch several BroadcastServer instances on one computer. However, it is the responsibility of the administrator to make sure that the following settings are different for all parameter sets:
Paths/*: All paths must be different!
Buddy/Address and **Buddy/TreePath**: Two BCS must *never* use the same buddy!
IP/ServerPort: Each BCS instance must listen on a different port.

6.8 Operation

In the top half of the BroadcastServer window, a list of connected clients is displayed, while the lower half shows the protocol. By selecting one or more lines in the client list, and right-clicking the selection, a popup-menu appears with the option to force a logout of the selected clients.
The "BroadcastServer" menu has currently the following options:
Reload Settings: This reloads all settings, which can be changed at runtime (see list of parameters in chapter “Settings” above). Since the optimized handling of DigaSystem rights and objects with version 5.7.365.0 it might be necessary to execute this command to make BCS read the right objects of a newly created service.
Switch to Master/Backup Mode: If a buddy server is configured and connected, the master/backup status can be switched (for details, see chapter “Buddy Server” below). This option requires the Restart2.exe application!
Update full tree on disk: This command forces the BCS to update the XML files in the disk storages with the current data. Normally, such an update is done in the background every few minutes (and of course before shutdown), but for testing and debugging purposes it can be necessary to view the XML data immediately after some operations by a client.
Convert disk files to UTF-8: The first release of BCS stored XML files in UTF-16 format. To reduce storage space, this was changed to UTF-8 later. Files are automatically converted whenever they are accessed by the BCS, but this may never happen for XML files in past days. The menu command can be used to explicitly convert all the XML files. This will run in the background so that normal BCS usage can continue.

6.9 Buddy Server

6.9.1 Setup

The BroadcastServer uses the concept of "Buddy Servers" for increased reliability. To install a Buddy System, you have to set up two BroadcastServers (on two separate machines). For each BCS, access data for the *other* BCS (the "buddy") must be configured in the "Buddy" subkey of the parameter settings. It is mandatory that each BCS has read access to its buddy's disk storage tree.
IMPORTANT: If a single-server system is to be upgraded to a Buddy System or if changes are made to the Buddy System configuration, always make a **backup of the complete storage tree** on disk!! In case of a configuration error, it is possible that data can be deleted!
Steps to set up a Buddy Server (assuming a stand-alone server is up and running)
ATTENTION: It is very important to follow these steps in this exact order - otherwise, the setup might not work, or data loss might occur!

1) Configure the Buddy Server data in the DigaSystem registry file for the running server. Do **not yet run** the intended **Buddy!** If you have already done so, you must shut down the Buddy, and **remove the "LastSaveTime.txt"** file from the Buddy's storage tree directory!!

2) The setting "General/ServerNameForGlobalRights" should be the same for both buddies, so that they use identical rights settings from the DigaSystem registry.

3) If you are sure that the Buddy isn't running, and there is no "LastSaveTime.txt" file on the Buddy, shutdown and restart the first (master) server. It is now ready to accept a Buddy login.

4) Configure the Buddy server and start it. It will then align its disk storage tree with the Master's (this can take some time, if the Master's tree is already big), and continue to run in Backup mode.

5) In case the Master is intended to run as Backup, simply use the "Switch to ...." option on either of the two servers. You should try this switching at least once in both directions to make sure the setup works.

6) You can optionally set the setting “Buddy/IAmPreferredMaster” to TRUE, so that this particular BCS will run always as master if possible.
Notes:
1) If this option is set, a manual switch to Backup Mode will be followed immediately by an automatic switch to Master Mode! You have to change the setting to FALSE first.
2) Make sure that at most one of the two Buddy Servers has this flag set to TRUE!

Principles of Operation

When a buddy server is configured at BCS startup, the BCS tries to log into the buddy BCS. If this fails, it assumes the buddy isn't running, and will run itself in Master Mode. If login at the buddy is successful, it goes into Backup Mode.
The first action in Backup Mode is to align its own disk storage tree with the master's. As soon as this is done, the backup BCS will receive notifications about changes just like any other ordinary client, and will update its own disk tree accordingly. **Note:** During tree alignment, the master BCS won't update its disk storage. However, with the exception of the very first backup server login, this alignment will take only a few minutes at most, because only changed files have to be copied. Normal operation of the master server is unaffected during tree alignment.
The buddy BCS does continuously check its connection to the master. As soon as this connection breaks, it assumes the master is down and switches to Master Mode itself. When the buddy BCS is restarted, it will now enter Backup Mode, so the roles have been switched.
The master/backup modes can also be switched manually by using the option in the "BroadcastServer" menu. This works by forcing the current master BCS to shutdown and restart. Because the backup BCS keeps running, it enters master mode as soon as the (former) master has shut down. After restart, the former master can log into the new master and become backup BCS.

6.9.2 Limitations

IMPORTANT: Don’t try to start the two BroadcastServers in a Buddy system simultaneously! Otherwise, it can happen that both servers are not ready to accept connections, when the other one is checking for a running master. This way, you end up with two BroadcastServers in Master Mode! If the BCS is auto-started after PC boot-up, don't boot both servers at the same time. If this can happen (e.g. after recovering from a power failure), make sure one of the BCS is started via a script which includes a delay. As a fail-safe measure, a BCS in master mode, which has a Buddy server configured, but not logged in, will continually check whether the Buddy is also running in Master Mode. If so, the two Masters negotiate, which one of them will switch to backup mode.
It is not possible to use more than two servers in a Buddy System. It is currently the responsibility of the BCS administrator to make sure that the "Buddy" settings (especially the path to the storage tree of the buddy) are all correct.
Network Trouble
The Buddy system is somewhat vulnerable to *partial* network failure. If the IP route between the two buddies is broken, both buddies will continue to run in Master Mode. As long as one of the buddies is completely unreachable from any client (which is the usual case, if that buddy's network hardware fails locally), this is no problem, because all clients will connect to the sole reachable BCS.
However, there are two critical situations:
1. Some clients can reach only one BCS, some clients only the other. This will immediately result in discrepancies between the data sets of the two BCS.
2. The network connection of the "failed" BCS comes back unexpectedly, so that it is again reachable. Now there are two reachable masters, with potentially the same problems as in case 1.
As soon as the two BCS can see each other again, and find out that they are both running as Master, the one which has logged the lower number of write access to its data will switch to Backup mode, and continue to use the other server’s data. This ensures that the two Buddies again use identical data sets, while minimizing the amount of “lost work”.
If possible, it should be ensured that a failure of the network connection between the two buddies automatically means that at least one Buddy is completely unreachable. This can be guaranteed, if there is not more than one router, switch or hub between the two BCS servers.

6.10 Plug-In DLL

The BCS can optionally call a plug-in DLL whenever a new node is inserted. Such a DLL can be custom-designed for each customer of a BCS system. Settings for the DLL are always stored in the DigaSystem registry under the key “BCSPluginDLL*Configuration*”. The key name *Configuration* is passed via the Plugin/Parameters setting of the BCS. If this setting is empty, the key “Default” is used.

6.10.1 Standard Plug-In DLL

There is a standard plug-in DLL (named “BCSPlugin.DLL”), which offers the following services.

6.10.1.1 Assignment of Default Values

The DLL can assign default values to fields which are empty in newly inserted elements. These fields must be defined in the key “BCSPluginDLL*Configuration*\Common\DefaultValues”. The name of each entry must be the name of the BCS field, and the value is the default value, which will be written to this field if it is empty.

6.10.1.2 Adjustment of Deletion Date in Database Entries

If an element from the DigaSystem database is inserted into the BCS, and this element has a deletion date defined, the DLL can ensure that the deletion date is after the last date of use of this entry in the show plan. If an element is inserted into a show, the deletion date is modified to be not earlier than two days after the date of the show. If the element is inserted into a day template or show template, the deletion date is moved to 2099-12-31, because templates can potentially be valid forever. Note that if the database entry does not have a deletion date defined in the first place, it will never be modified.

6.11 Error-Codes

Here is a list of possible error codes / return values from BroadcastServer and a short explanation for each code:

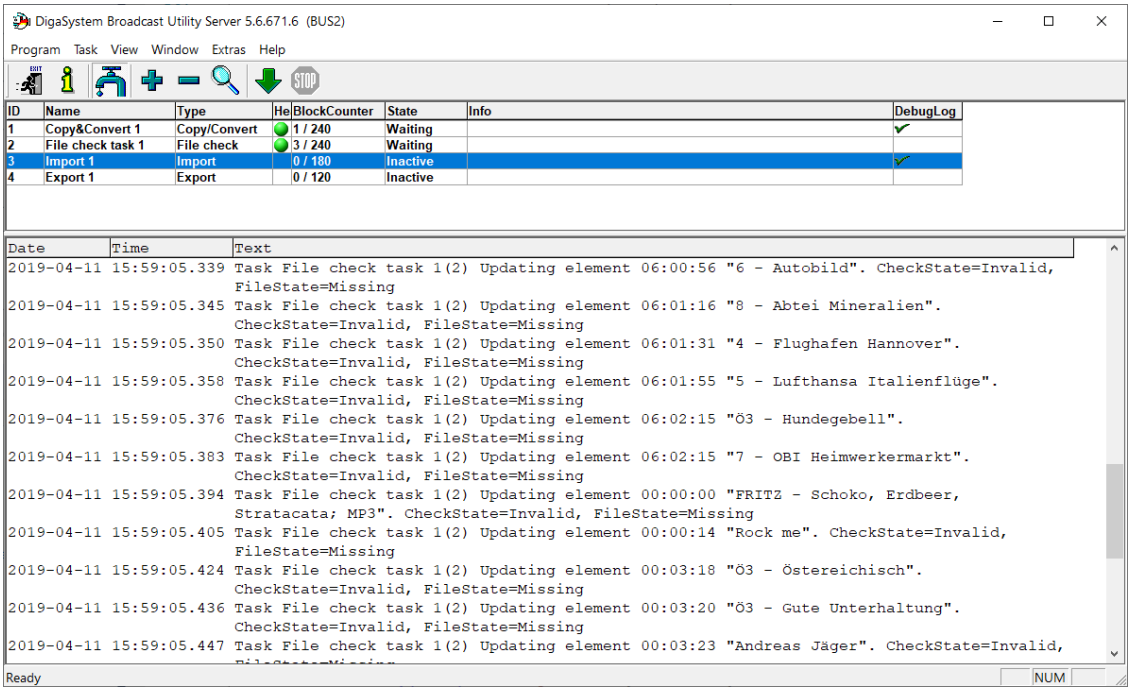
Code	Ret value	Explanation
Windows error codes:		
10022 (0x00002726)	-	An invalid argument was supplied.
10024 (0x00002728)	-	Too many open sockets.
10037 (0x00002735)	-	An operation was attempted on a non-blocking socket that already had an operation in progress.
10038 (0x00002736)	-	An operation was attempted on something that is not a socket.
10042 (0x0000273A)	-	An unknown, invalid, or unsupported option or level was specified in a getsockopt or setsockopt call.
10043 (0x0000273B)	-	The requested protocol has not been configured into the system, or no implementation for it exists.
10044 (0x0000273C)	-	The support for the specified socket type does not exist in this address family.
10046 (0x0000273E)	-	The protocol family has not been configured into the system or no implementation for it exists.
10047 (0x0000273F)	-	An address incompatible with the requested protocol was used.
10048 (0x00002740)	-	Only one usage of each socket address (protocol/network address/port) is normally permitted.
10049 (0x00002741)	-	The requested address is not valid in its context.
10050 (0x00002742)	-	A socket operation encountered a dead network.
10051 (0x00002743)	-	A socket operation was attempted to an unreachable network.
10052 (0x00002744)	-	The connection has been broken due to keep-alive activity detecting a failure while the operation was in progress.
10053 (0x00002745)	-	An established connection was aborted by the software in your host machine.
10054 (0x00002746)	-	An existing connection was forcibly closed by the remote host.
10055 (0x00002747)	-	An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.
10056 (0x00002748)	-	A connect request was made on an already connected socket.
10057 (0x00002749)	-	A request to send or receive data was disallowed because the socket is not connected and (when sending on a datagram socket using a sendto call) no address was supplied.
10058 (0x0000274A)	-	A request to send or receive data was disallowed because the socket had already been shut down in that direction with a previous shutdown call.
10060 (0x0000274C)	-	A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.
10061 (0x0000274D)	-	No connection could be made because the target machine actively refused it.
10065 (0x00002751)	-	A socket operation was attempted to an unreachable host.
10091 (0x0000276B)	-	WSAStartup cannot function at this time because the underlying system it uses to provide network services is currently unavailable.
10092 (0x0000276C)	-	The Windows Sockets version requested is not supported.
10093 (0x0000276D)	-	Either the application has not called WSAStartup, or WSAStartup failed.
10103 (0x00002777)	-	A call to WSALookupServiceEnd was made while this call was still processing. The call has been canceled.
11001 (0x00002AF9)	-	No such host is known.
536872064 (0x20000480)	-	WinSock 2 is not available
536872065 (0x20000481)	-	Cannot enumerate NetBIOS LanA's
536872066 (0x20000482)	-	Cannot resolve IPX-server name
536872067 (0x20000483)	-	Socket already opened

536872068 (0x20000484)	-	Socket not opened
536872192 (0x20000500)	-	Packet size is already defined
536875008 (0x20001000)	-	No data available
536875009 (0x20001001)	-	Packet is too big
536875010 (0x20001002)	-	Cannot read from packet-FIFO
536875011 (0x20001003)	-	Packet operation timed out
536872198 (0x20000506)	-	Packet size is not defined
536872199 (0x20000507)	-	Cannot open packet-FIFO
536872200 (0x20000508)	-	Two simultaneous send operations on same socket attempted
536872201 (0x20000509)	-	Invalid answer received (server does not use the PacketSocket protocol)
536872202 (0x2000050A)	-	Packet transmission is in progress
536872320 (0x20000580)	-	No answer received from connected socket
536872321 (0x20000581)	-	Internal \"PAM\" table is full
536872322 (0x20000582)	-	File error at the other side
536872323 (0x20000583)	-	Data transmission was aborted by peer
536872325 (0x20000585)	-	Receiver buffer too small
536872326 (0x20000586)	-	Wrong or unexpected answer from peer
536872327 (0x20000587)	-	The peer does not allow file deletion
536872328 (0x20000588)	-	Operation not supported by peer
BCS error codes:		
536873984 (0x20000C00)	-	Buffer allocation error at receiver side
536873985 (0x20000C01)	-1	Invalid user
536873986 (0x20000C02)	-2	Invalid password
536873987 (0x20000C03)	-	No user logged in
536873988 (0x20000C04)	-2	Node not found
536873991 (0x20000C07)	-	Invalid date
536873992 (0x20000C08)	-3	Error while creating a node
536873993 (0x20000C09)	-5	It is not allowed to insert the node at the specified location
536873994 (0x20000C0A)	-6	The node cannot be deleted
536873995 (0x20000C0B)	-9 / -10	You do not have the rights to perform this operation
536873989 (0x20000C05)	-11	The node is locked by another user
536873997 (0x20000C0D)	-12	The specified class is not allowed at this place
536873998 (0x20000C0E)	-5	BroadcastServer and client are incompatible. Please upgrade your software!
536873999 (0x20000C0F)	-	Internal DigaSQL error
536874000 (0x20000C10)	-13	Invalid time data in show
536874001 (0x20000C11)	-14	Invalid time in non-moving element or group
536874002 (0x20000C12)	-15	Too many levels were requested
536874003 (0x20000C13)	-16	Invalid node for this operation
536874004 (0x20000C14)	-17	Connection to remote server failed
536874005 (0x20000C15)	-18	Invalid destination node for block move operation
536874006 (0x20000C16)	-6 / -19	Backup connection to backup server not allowed
536874007 (0x20000C17)	-7	Second backup connection not possible
536874008 (0x20000C18)	-	Buddy notification was ignored
536874009 (0x20000C19)	-	Only read operations are allowed on backup server
536874010 (0x20000C1A)	-21	The node cannot be moved to this destination
536874011 (0x20000C1B)	-	No running master server found
536874012 (0x20000C1C)	-22	These nodes cannot be updated in a single block operation
536874013 (0x20000C1D)	-23	Node not found - parent does not exist
536874014 (0x20000C1E)	-24	Maximum number of jingles in jingle group is reached
536874015 (0x20000C1F)	-25	This operation is not allowed
536874016 (0x20000C20)	-26	One or more nodes are locked by other users

536874017 (0x20000C21)	-27	An invalid search request was sent to the server
536874018 (0x20000C22)	-28	The node is currently in use
536874019 (0x20000C23)	-29	The node ID is already in use
536874021 (0x20000C25)	-	Error while parsing a client-side XML string
536874022 (0x20000C26)	-30	There is only read access to this remote show
536874023 (0x20000C27)	-31	Unexpected error during access to remote show
536874024 (0x20000C28)	-32	Error during operation on referencing node
536874025 (0x20000C29)	-33	The selected remote node is itself linked to a remote node
536874026 (0x20000C2A)	-34	The ID of the remote track doesn't match the ID of the local track
536874027 (0x20000C2B)	-35	There is no connection to the remote server
536874028 (0x20000C2C)	-36	No answer received from remote server
536874029 (0x20000C2D)	-37	There are several shows with the same start time on the remote server
536874030 (0x20000C2E)	-38	No matching remote node was found
536874031 (0x20000C2F)	-39	There is already a track with the specified number
536874032 (0x20000C30)	-40	The remote server is busy connecting itself to another remote server
536874033 (0x20000C31)	-41	Error while reloading the node data from disk
536874034 (0x20000C32)	-42	Operation is temporarily not possible (may succeed at a later time)
536874035 (0x20000C33)	-43	Day is too old, write operations are no longer allowed
536874036 (0x20000C34)	-44	Move from one remote server to another remote server
536874037 (0x20000C35)	-45	The functionality is not implemented in BCS LITE
536874038 (0x20000C36)	-8	Maximum number of connections is reached (BCS LITE only)
536874039 (0x20000C37)	-46	Invalid start type of node in insert or update operation
536874040 (0x20000C38)	-47	An invalid XML string was received
536874041 (0x20000C39)	-48	An internal BCS error did occur
536874042 (0x20000C3A)	-	BCS did send a result which is unknown to the client
536874043 (0x20000C3B)	-50	An operation was rejected because a parent node is MOS-controlled and the changes cannot be synchronized.
536874048 (0x20000C40)	-49	An operation on a mixture of local and remote nodes was tried which is not allowed.
536874049 (0x20000C41)	-51	A node was deleted while waiting for a short-time lock to get released.
536874050 (0x20000C42)	-52	A regular node lock was requested which would affect a MOS controlled node. This is not allowed because MOS controlled nodes must be changeable by MOS at any time.
536874051 (0x20000C43)	-10	No rights object to check the rights has been defined.
536874052 (0x20000C44)	-	This is a generic error number for a WebSocket error on client side. See the logfiles. They might contain more detailed information.
536874053 (0x20000C45)	-	Invalid authentication data for a login received.
536874054 (0x20000C46)	-	The WebSocket connection was closed.
536874055 (0x20000C47)	-	Timeout while waiting for an answer.
536874056 (0x20000C48)	-	SeplProtocol is not available for 64-bit architecture. You have tried to configure the usage of the old SeplProtocol but are using a 64-bit variant of the application. Only WebSockets are supported.
536874057 (0x20000C49)	-	The client tried to login with an authentication type, which is not supported.
536874058 (0x20000C4A)	-	OIDC-Token is not valid for the current time (e.g. has expired).
536874059 (0x20000C4B)	-	The URL for DPE rights service was not configured, the specified secure DPE token cannot be checked. Configure the parameter: BroadcastServer\...\General\DPETokenUrl.
536874060 (0x20000C4C)	-	The configured DPE rights service for checking a token is not reachable or it does not respond.
536874061 (0x20000C4D)	-	An unexpected error was reported from DPE when trying to verify a secure DPE token. Please check the BCS log for more information, or – if nothing is present there – increase the log level up to 5 max.
536874062 (0x20000C4E)	-	A client attempted to do a BCS request, which is not supported with SeplProtocol. You should use a WebSocket connection so that the type of request is supported.

536874063 (0x20000C4F)	-53	The specified file cannot be found. This error is only returned when a client tries to get the content of an XML file with the request GetTreeXMLFile.
536874064 (0x20000C50)	-54	There is an access problem for the specified file. This error is only returned when a client tries to get the content of an XML file with the request GetTreeXMLFile.
536874065 (0x20000C51)	-55	The specified file is too big (bigger than 128 Mbyte). This is not supported. This error is only returned when a client tries to get the content of an XML file with the request GetTreeXMLFile.
536874240 (0x20000D00)	-	This is an internal error. BCS tried to open a BCS connection for a remote link via the wrong connection to another BCS.
536874241 (0x20000D01)	-	A remote link seems to be defect – it does not point to a show / track.

7 BROADCAST UTILITY SERVER



7.1 Bus in General

7.1.1 Introduction

The Broadcast Utility Server, or in short “BUS”, is part of the DigaBroadcastSystem. It should run on a dedicated Windows computer to perform some background tasks. The main reason for creating an extra program was the reduction of workload for the actual broadcast server. BUS has been designed for good scalability. You can run all tasks on the same computer, or you can distribute them on multiple computers if the load gets too high. The following tasks are handled by BUS:

- Checking of media files
- Copying or converting of media files (including copying files from remote locations)
- Deleting of old media files
- Automatic day structure import
- Automatic import or export of schedules
- Automatic copying of shows to the future to repeat these shows
- Automatic mirroring of shows for safety
- Automatic rebroadcast of sent shows
- Automatic insertion of database elements to some specific places
- Automatic replacement of BCS entries by DigaSystem database entries
- Support of de-archiving workflows
- Copying information from the BC system to the database (protocol data, send counter)
- Synchronize definitions and ressorts with a DigaSystem database
- Stress test for BroadcastServer
- Synchronization of shows between two BCS
- A permanent check of many problems which can occur in a rundown (consistency check)
- Automatic Radio AD-Server import
- Index creation for full-text search
- Metadata scheduler

All these tasks will be described below.

7.1.2 Requirements

BUS needs a Windows operation system. A server version is not necessary, but a 64-bit system is strongly recommended. The hardware depends on the number and type of tasks you want to run: for Deleting, Aligning and Stress test a very old hardware is sufficient. Checking, Import and Export already need a quite powerful computer and if you are going to use Conversion the computer cannot be fast enough. As a lead you should use a multicore-CPU with 3 GHz and 4 GByte RAM if BUS (as 32-bit variant) has to treat a single service. For the 64-bit variant of BUS you can add much more memory. For multiple services it is better to use multiple computers than a single fast one. In general, a multiple processor system is sensible because BUS makes heavy use of independent internal threads which can be divided onto different processors. The memory should be quite big because BUS stores a lot of data in RAM whereas the harddisk can be small because it is only needed as a cache for import/export data. Unfortunately, the 32-bit variant of BUS cannot make use of more than 2 GByte of RAM. The network connection (to the BroadcastServer, to the file servers and to the on-air clients) should be as fast as possible.

7.1.3 64-bit BUS

Since version 6.2 BUS exists as a 64-bit variant: BUS64.EXE. Important: a 64-bit process can only load 64-bit dlls. Therefore, all required dlls must be converted to 64-bit, too. Especially for some exotic import or export dlls this will only be done on special request by a customer. So please contact DAVID Systems if you think that you need a 64-bit dll, which is being used by BUS!

A general hint: all 64-bit modules of DAVID Systems must be installed in a 64-bit Digas folder. This is not the same as the regular Digas folder, which is normally located nowadays here: C:\Program Files (x86)\DigaSystem. You should make use of a folder for 64-bit modules: C:\Program Files\DigaSystem.

If you upgrade from the 32-bit variant or if you want to switch between the 32-bit and 64-bit variant, you need to pay attention to the configured filenames of dlls: the reference to these dlls is stored in different registry values. For example, the import module (dll) for the import task is stored in value “ImportModule” for 32-bit and in “ImportModule64” for 64-bit. This allows to start both variants of BUS without the need to reconfigure all these modules. But when you start the 64-bit variant for the first time, you must check all the settings and enter the correct filename of the 64-bit dlls. BUS will try to automatically derive a filename from a given 32-bit filename, but the result need not be correct.

7.1.4 Installation

Since version 5.6 BUS can run in three different ways: as a desktop application, as a sys tray application and as a Windows service. The installation steps for running BUS as a desktop application must be done in every case. For running in sys tray or as a Windows service, additional steps are necessary.

It is strongly recommended to always start BUS as a desktop application first until installation has been fully completed and all your tests have been finished. Only afterwards, you should change the way BUS operates.

The installation is simple, if you have a DigaSystem registry available on the computer. If not, you should copy the DigaSQL.DLL to the computer and start the Admin program at least once to configure the location of the parameter files. In addition to the program file itself (BUS.EXE/BUS64.EXE) the following components are needed:

- The Audio32.dll / Audio64.dll is required for audio processing. Depending on the used formats, this dll uses many other coders/decoders, like: MP2.SYS, MSACM32.DLL or the lame_enc.dll. Typically, you will always have to include the libmpg123.dll. This is a codec for MP3 files, and it is linked statically to the Audio32/64.dll, which means: you always need this dll even if you do not process MP3 files. Attention: the libmpg123.dll exists in a 32-bit and in a 64-bit variant, but both variants have exactly the same filename! This can be confusing. Therefore, pay attention to put all 32-bit modules into a DigaSystem folder and all 64-bit modules into a DigaSystem64 folder!
- For loudness analysis, the LoudnessDll.dll / LoudnessDLL_x64.dll is required.
- Licenses dlls are needed to activate some of the tasks or features: BcsLoudness.dll, BcsConsistencyCheck.dll, RadioADServiceLicense.dll, BCS_MDPLicense.dll, LoopImporterLicense.dll, MetadataSchedulerLicense.dll.
- TCP/IP support (install this protocol if missing)
- XML support: at least MSXML3.DLL, MSXML3A.DLL, MSXML3R.DLL, the first one must be registered on the system (use REGSVR32.EXE) Hint: this was only necessary on old Windows versions, like Windows 2000. For current Windows versions like Windows 10 these dlls are part of the regular installation.
- ODBC support (install MS Data Access components if missing). This is only needed for some of the tasks – for all which have to access a DigaSystem database. Hint: for current versions of Windows ODBC is always included and you do not need to install anything manually anymore.

If you want to run BUS as application, which can hide into the sys tray of Windows you go to the general settings dialog (see below) and set the parameter “Hide window to systray after startup”.
If you want to run BUS as a Windows service, you need to install the executable first as new service. This can be done by opening a command line – which must be started with “Run as administrator” ! Navigate to the location of

BUS.exe and run the command “BUS.exe –install”. This will register BUS with the Service Control Manager of Windows. There are additional command line options you can use, please see below. Afterwards you can start BUS via the Service Control Manager or let BUS be started automatically.

Please pay attention to 3 topics. The first one is the account with which BUS runs. This is explained below with the command line options. The second topic is about the automatic restart. If running as a desktop application BUS can make use of Restart2.exe to perform a restart on request of a user or if an unrecoverable error is being detected. This does not work if BUS is running as a service because the Service Control Manager must start a service. You should check the third property page of the service, which lists actions in case of a failure of the service. The actions for all failures should be to automatically restart the service. Hint: this is automatically configured by BUS when it is installing itself. The 3. topic is about logging – see the extra chapter below.

7.1.5 Command line options

BCS 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 -prf	configuration	BUS stores its settings in the local DigaSystem registry. As default, all settings are stored below the DigaSystem registry key “BUS”. Alternatively, other top-level keys, like e.g. “BUS2”, can be defined in the registry. The name of such an alternative key must be specified as “configuration”. This allows running multiple BUS instances on a computer with a different set of parameters and tasks. If this option is not specified, “BUS” is used as configuration name.
-install -i	-	Can be specified if BUS is executed from a command line which must be opened with “Run as administrator” ! BUS 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. The new service will be called “BUS” 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 an underscore. For example, if you install the service with this command line: BUS –i –config”MyConfig” The created Windows service will use this name: “BUS_MyConfig”.
-uninstall -u	-	Uninstalls BUS 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.
-account	username	If you install BUS 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 for BUS. BUS 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 BUS. 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).
-password	password	
-service	-	This option is automatically used if BUS installs itself. This option signals BUS that it is started as a service and therefore should never be used otherwise. The impact of this option is very limited. BUS will detect that it is 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.

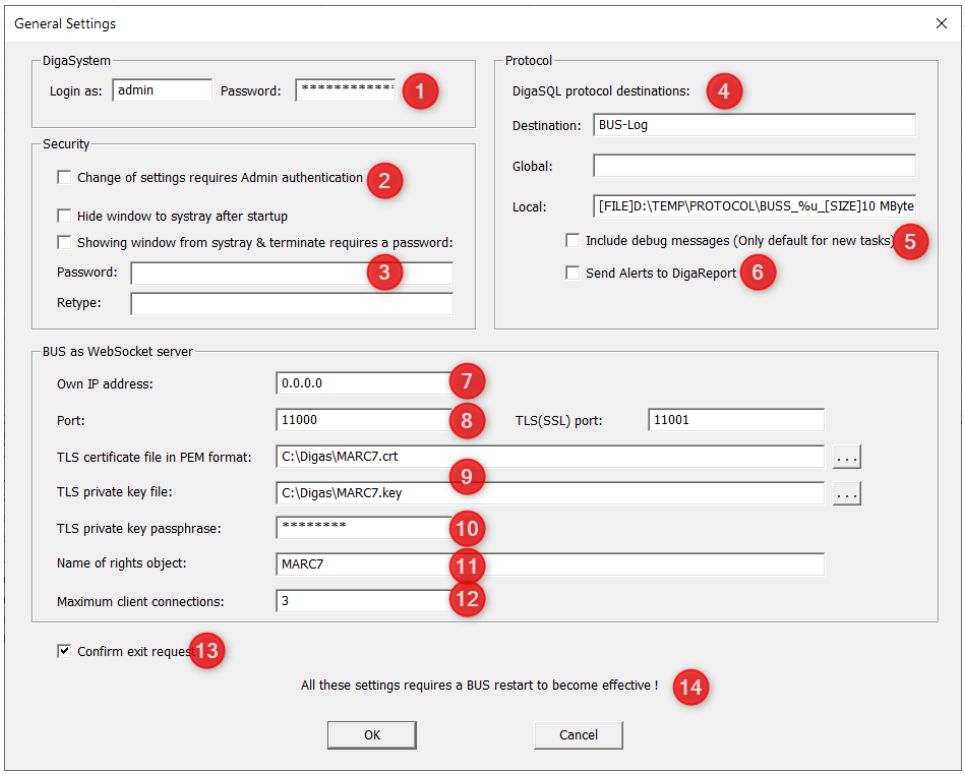
-b -bm	Filename -	When BUS starts as a Windows service, it is running in a different “ring” and probably with a different account than programs running on the regular desktop. As a consequence, mapped drive letters might not be available like for programs on the desktop. This can affect any area, but it is a special problem with the registry / parameter files of the DigaSQL.dll. If the location of these files is specified with drive letters, then the startup of BUS will fail. The command line parameters -b and -bm allow to execute a batch file before BUS itself starts. In such a batch file drive letters can be mapped as needed. BUS will try to execute the specified batch file and will wait until it terminates before continuing. With the parameter -b you can (and must) specify the filename of the batch file. This can either be a fully specified filename with a path or only the filename. If you omit the path, then the path from where BUS.EXE is being executed is used. The parameter -bm is only an abbreviation for: -b:”Mapping.bat”. Some hints: you cannot use both, -bm and -b at the same time. You cannot execute an arbitrary executable. It must be a batch file. Have a look into the log of BUS. Immediately after the startup line there is a line, which reports about the started batch and whether it was a success of whether an error occurred.

For the BUSGUI.exe (see below) the following command line options are available:

Option	**Data**	**Meaning**
-config -c -prf	configuration	Like for BUS above.
-BUS	name	Directly connect to BUS engine “name” after start. “Name” must be one of the keys in the DigaSystem registry below key “Digas\BUS”.

7.1.6 Program settings

A general hint: Do not change any settings in the DigaSystem registry itself. All possible settings are available from program dialogs. The following dialog is available for configuring the general settings of BUS:



1) The login name and password are used for the general DigaSystem login of BUS itself. The name specified here is independent of any BroadcastServer- or database login described later.
2) The next option can be set if you want to limit the access to the settings dialogs. If checked, BUS requests an Admin authentication whenever you want to change settings. To ease the administration an authentication stays valid until you do not change anything for some minutes.
3) BUS can hide itself to the systray while running. With a context menu of the systray icon, BUS can either be terminated or made visible again. These operations can be protected by an extra password.
4) The three protocol fields correspond to the log destination BUS uses. The default destination is “BUS-Log”. In the registry you can add this value to the top-level key "Protocol" and assign it to the files into which the protocol will be written. The second and the third field show these registry values. Normally you specify a directory and the beginning of the log filename. But as you can see in the third field, you can also use a section-string syntax to specify more options. (See the parameter documentation for more information about the DigaSystem logging mechanism.) The destination need only be changed if you run multiple instances of BUS (e.g. "BUS-Log2", "BUS-Log3"...), because otherwise all BUS instances will write to the same destination (=file). If the global and the local value are empty, the protocol is only available in the lower window, but no file will be written.
5) The checkbox for debug messages should only be activated if you are looking for problems or errors. Normally this will create too much output. This setting is only a default for tasks created anew. Each task has its own flag for enabling debug reports, which can also be changed with the context menu of the task list while the task is already running. The DigaReport flag is valid for all tasks, but they must all be restarted to take effect.
6) DigaReport is an old product, which is no longer supported. Possibly, it will be replaced sometime by an SNMP support.
7) The own IP address is needed if BUS should act as a WebSocket server for a remote GUI. Set this address to “0.0.0.0” if you want BUS to listen on all network adapters. If BUS should listen on a single network adapter only, enter the IP address for this adapter.
8) You can specify two listen ports. One is for plaintext communication, the other one with TLS. If you leave one of the port settings empty, the corresponding listen port is not opened.
9) In case you want BUS to use TLS for communication, it is necessary to provide a certificate file in PEM format. These files contain a certificate in Base64 encoding and typically have one of the file extensions *.pem, *.crt, *.cer, *.p7b or *p7s. Such a file can be opened with a text editor and the first line will contain something like: “-----BEGIN CERTIFICATE-----“. Such a certificate file comes in conjunction with a private key file, which typically

has the file extension *.key and which is also a Base64 encoded text file with a first line like e.g. “-----BEGIN RSA PRIVATE KEY-----“. If you have a *.pfx file, this is a bundle of certificate and private key. It can be split up into different files, e.g. with the OpenSSL executable. Normally, you need to request these files from a certification authority. In case of BUS and as long as you are going to use only DigAIRange and BUSGUI as remote GUIs, you can also create a self-signed certificate, because these two clients will not validate the certificate. This means that these two programs do not check whether they are connected to the right server, but the communication is encoded, nevertheless.

10) Normally, a passphrase is required to make use of the private key. In case you create a self-signed certificate, you can also specify that no passphrase is required and then you can leave this field empty.
11) In order to connect to a BUS as a remote GUI, the user logged in at the GUI program must either be “ADMIN” or he/she must have the manage right to a DigaSystem rights object of type “BUS” and with the name you specify in this field. Pay attention: if you leave this field empty, the manage right is not checked and all DigaSystem user will be allowed to connect!
12) Optionally, you can limit the maximum number of client connections from remote GUIs. If you leave this field empty, the number of connections is not limited.
13) Check this box, if BUS should display a confirmation box before closing. This applies only if BUS is running as a desktop program.
14) Please have in mind that you need to restart BUS whenever you make changes in this dialog, otherwise the changes will not become effective.

7.1.7 Special test commands

The main menu of BUS has the “Extra” item on it. It is intended only for testing / debugging purposes and none of its commands should be executed during normal operation! The menu item has these commands:

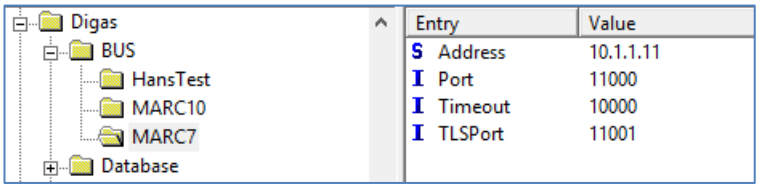
Test out-of-memory situation	This command will eat up all available memory in the next 1-2 minutes. Afterwards any operation in BUS, which requires memory, can fail. It depends on the running tasks and operations, but mainly the location is given randomly. Be aware that this command will cause a serious crash of BUS! It is only intended to check whether a self-restart with the help of the tool “Restart2.exe” works correctly under the most extreme condition. Hint: you can also use a Stress test task with mode "Simulate out-of-memory" for the same purpose.
Do not log at all	Log output is completely suppressed.

7.1.8 Remote GUIs

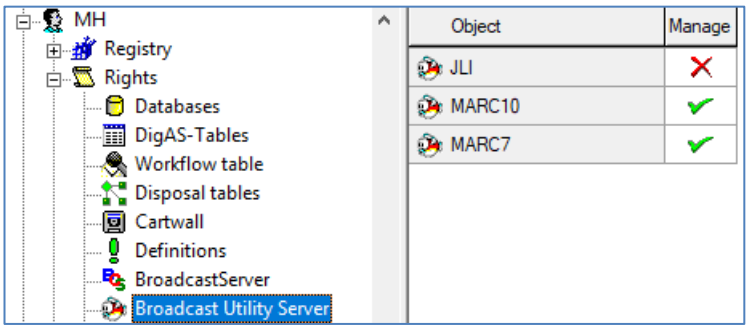
Two different remote BUS GUIs are available: one in DigAIRange, which has only limited functionality, and another one with the program BUSGUI.EXE. The remote GUI of DigAIRange allows only to have a view of the BUS main window (with the task list and the protocol window), to monitor tasks, to start and stop tasks, to set a few settings and to terminate or restart BUS. This GUI is not intended for configuration. Therefore, you have to use BUSGUI.EXE which allows full remote configuration.

7.1.8.1 DigAIRange

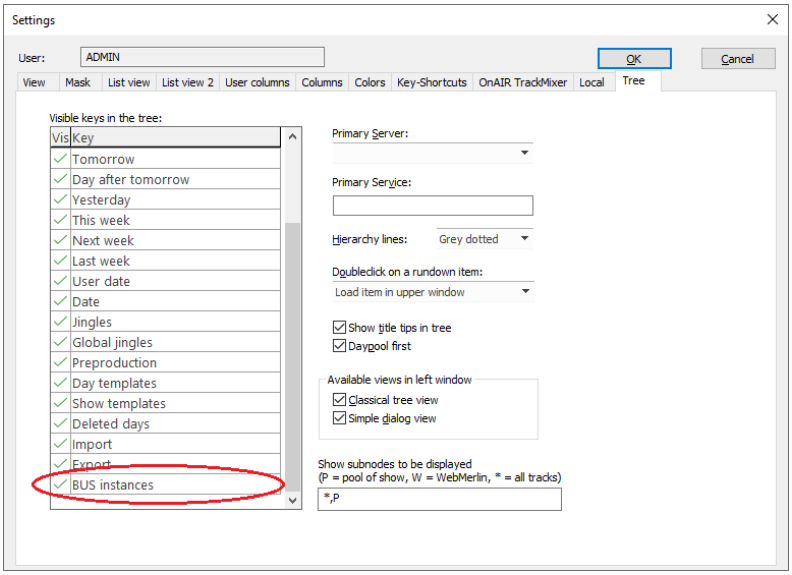
In order to have the remote BUS GUI window available in DigAIRange, you first need to configure the available BUS instances in the DigaSystem registry. Use the key “Digas\BUS” to create sub keys, one for each instance. The name of the sub key is the name as it is displayed to the user. Possible parameters are: Address, Port, TLSPort and Timeout. For more information, see the parameter description in DigParam.rtf.



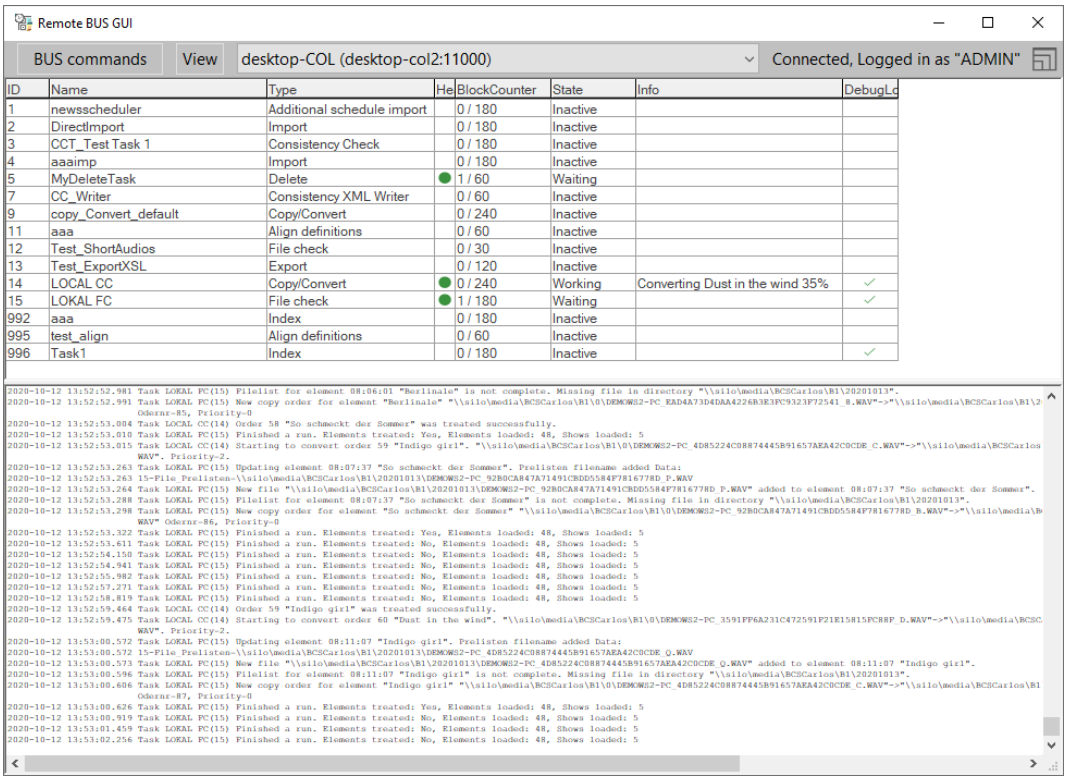
Second, you need to grant manage right on the BUS instances for each user, which should have access. Therefore, create new right objects of type “Broadcast Utility Server”, one for each instance. The name of the right object is independent and can be configured in the general settings dialog of BUS. Nevertheless, it is a good idea to make the name of the right object identical to the name of the sub key in the registry. Afterwards, you can grant manage right for these objects.



Third, you need to activate the display of BUS instances in the tree view of DigAIRange. This can be done in the settings dialog:



Now you should see the configured BUS instances in the tree view. If you double click on one of them, DigAIRange opens a new non-modal top-level window, which looks like this:



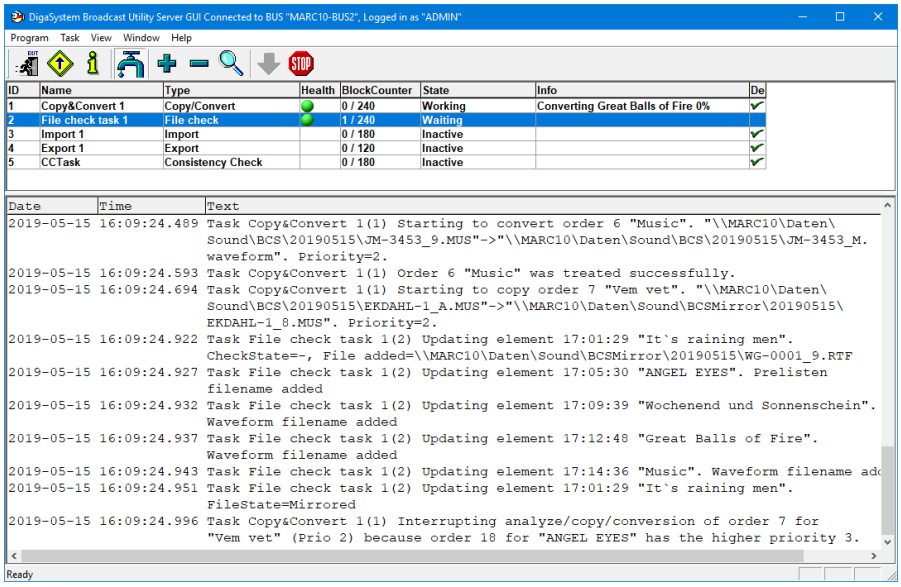
ID	Name	Type	He	BlockCounter	State	Info	DebugLo
1	newsscheduler	Additional schedule import		0 / 180	Inactive		
2	DirectImport	Import		0 / 180	Inactive		
3	CCT_Test Task 1	Consistency Check		0 / 180	Inactive		
4	aaaimp	Import		0 / 180	Inactive		
5	MyDeleteTask	Delete	●	1 / 60	Waiting		
7	CC_Writer	Consistency XML Writer		0 / 60	Inactive		
9	copy_Convert_default	Copy/Convert		0 / 240	Inactive		
11	aaa	Align definitions		0 / 60	Inactive		
12	Test_ShortAudios	File check		0 / 30	Inactive		
13	Test_ExportXSL	Export		0 / 120	Inactive		
14	LOCAL CC	Copy/Convert	●	0 / 240	Working	Converting Dust in the wind 35%	✓
15	LOKAL FC	File check	●	1 / 180	Waiting		✓
992	aaa	Index		0 / 180	Inactive		
995	test_align	Align definitions		0 / 60	Inactive		
996	Task1	Index		0 / 180	Inactive		✓

With this remote GUI, you can start and stop tasks, monitor tasks, activate or deactivate debug logging for tasks, monitor the log output and terminate or restart the BUS engine. You cannot remote-configure tasks. This feature is only available from the BUSGUI.EXE.

Some hints regarding the log output: With the menu commands in “View” you can temporarily stop the output to the log window with the command “Lock protocol view”. In this case, the remote GUI still receives log message but only stores them in an internal buffer. If you deactivate this lock, the output to the log window will include all stored log messages. This is different if you deactivate the item “Transfer of protocol from BUS is active” in the View menu. In this case, the remote GUI tells the BUS engine that it is no longer interested in log messages and the engine will no longer send them. Therefore, lines from the log output will be missing. Nevertheless, this might be a useful option in case you permanently have the remote GUI windows open but are only interested in the state of tasks.

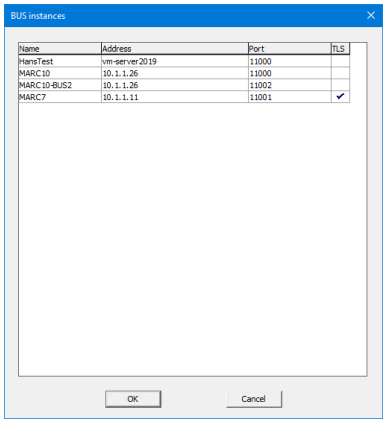
7.1.8.2 **BUSGUI**

The BUSGUI looks nearly identical to the main BUS program:



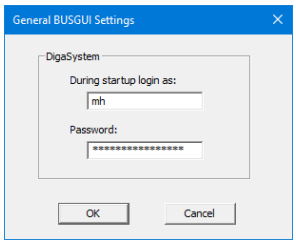
You will only see slight differences - like a different title. It says “DigaSystem Broadcast Utility Server GUI”. You can start multiple instances of the GUI. Beginning with instance 2, the instance number is displayed after this text. You can also use the feature to specify a registry key with the “PRF” or “config” command line parameter. In this case, the key name is displayed in brackets.

You might also notice the yellow button with up-arrow in the toolbar to connect to a running BUS instance. If you press this button, you will get a dialog, which displays the available BUS instances to which BUSGUI can connect:



To populate this list, BUSGUI uses the same parameters from the DigaSystem registry as DigAIRange: you need to list available BUS instances below the key “Digas\BUS” (see the screenshot above for DigAIRange). If you want BUSGUI to directly connect to a specific engine after the start, you can specify the command line option “-BUS:…”. The name you have to specify is one of the registry key names below the key “Digas\BUS”.

The user / password used to establish a new connection is the one, which is configured in the general settings of BUSGUI:

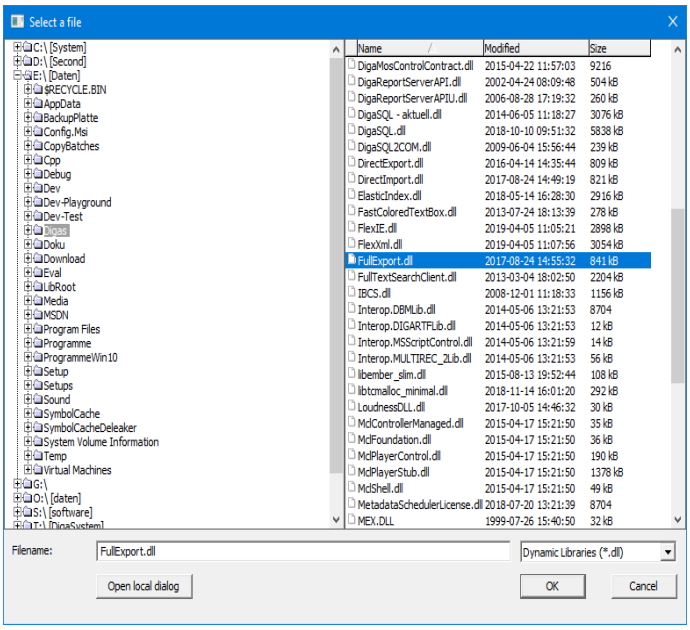


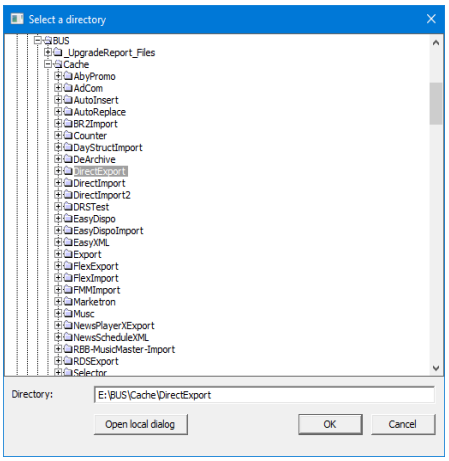
If you leave these fields empty, BUSGUI will ask about the credentials for every startup. Wile BUSGUI is running, a new login is not possible. You need to restart the application.

Controlling the BUS engine via BUSGUI is not very different from controlling it directly. You can terminate or restart the engine. Please have in mind that a termination means that you need a second access to the engine computer to start the engine again and that a restart requires that Restart2.exe is correctly installed on the engine computer. Starting and stopping tasks or settings the debug log level should work normally.

A remote configuration of the general settings of the BUS engine is possible, but you should always restart the engine after changing any of the settings! There are two reasons: first, the settings are only applied during startup of BUS and second, each remote BUSGUI connection works on its own copy of settings. All changes are written to the registry when the settings dialog is closed, but the changes will not be visible in the settings dialogs of other connections or of the BUS engine itself, unless BUS is restarted. This behavior might be confusing and can be avoided by always restarting BUS.

With the BUSGUI it is possible to remotely create, delete or configure tasks. This is a quite sophisticated feature, because accessing the file system is redirected to the engine computer. For example, if you open a file picker dialog in one of the configuration dialogs, you will see the files of the engine computer and not of the GUI computer. This also happens if you do the same action in one of the modules used by BUS, e.g. in the import or export dlls. If you press a "…" button in one of the configuration dialogs, the following dialogs will appear for file selection and directory selection, respectively:





These two dialogs display only the local file system of the engine computer, meaning: everything, which is reachable via driver letter. There is no free "network" access. If you need to browse the network, you can open the corresponding local Windows dialog on the GUI computer by pressing the button "Open local dialog". BUSGUI will automatically detect, whether it is running on the same computer as the engine and use the appropriate dialog. You can enforce the usage of a dialog type by holding the "Shift" or "Ctrl" key when pressing a "…" button. If you hold the "Shift" key, the special dialogs are opened. If you hold the "Ctrl" key, the regular Windows dialogs are opened. If you want to create a new directory within the special dir picker, you can enter the directory name in the edit field of the dir picker and press "OK". BUSGUI will prompt you, whether you want to create the new directory.

For opening the configuration dialog of one of the plugin modules (like an import dll), BUSGUI performs these actions:

- It reads the specified filename for the module, which is valid for the engine computer. It uses only the filename (not the path) and tries to load the dll by only specifying the pure filename. In this case, Windows has a standard search order for locating such a file. Typically, Windows starts to look in the execution path and in the path, in which the main application is located. This is also the place, where you should put all the dlls you need for configuration! Windows will continue to search in other directories, like the Windows system path, but you should not use this feature. It is recommended to put all dlls aside of the BUSGUI.EXE.

- After loading the local dll, BUSGUI will compare the full version number with the one on the engine computer. If the version numbers are different, you will get a warning, which displays both version numbers. Either you can ignore this fact, or you can cancel the operation. Normally, it is recommended to always use the same versions on the engine and the GUI computer, because otherwise you might configure settings, which are no longer available or interpreted differently on the engine computer. A hint: BUSGUI does not automatically download missing dlls. This could be done only for some of the possible dlls, but

others need a real installation with depending other dlls, or ActiveX controls, which must be registered, or even C++ runtime files or .NET frameworks can be necessary. It is not possible that BUSGUI has knowledge about all these details and provides installers for all cases.

- The data to the configuration dialogs of the modules is read from the engine computer and after a dialog is closed with OK, the data is transferred back. The module itself is not aware (and need not be) of the fact that it is configuring data of a different computer. As BUSGUI intercepts some file access calls, the modules need not know anything about all the remote stuff. Nevertheless, some of the dlls might use a special Windows function for displaying a file selection box, which cannot be intercepted. These dlls first need to be converted, so that they make use of a Windows call, which can be intercepted. This has already been checked or changed for the most important import/export modules, like: DirectImport, DirectExport, FlexIE, FlexXML, NewsScheduleXmlIE. If you need to remote-configure other modules, try out the file selection dialog. If you always get the local dialog of Windows, you can contact DAVID Systems so that we can provide you a changed version.

7.1.9 The Task Concept

The word “task” is used in some different ways. On one side, it means some special sort of operation, e.g. checking files. On the second side, it is used for an entry in the list of all configured tasks that means a set of parameters describing the task. On the third side, a running task is implemented as an independent thread which performs the assigned operation.
The main thread of BUS handles the user interface. It shows the window with the task list and the protocol view, it shows the parameter dialogs and it is responsible for starting, stopping and surveying the tasks. If you start BUS for the first time, there are no tasks in the list. You can add new tasks by pressing the plus button. A dialog appears in which you can enter a name, select the type of the task and specify the location where the task properties are stored. All these values cannot be changed for existing tasks. After the general dialog a task specific dialog appears, these dialogs will be described below. A configured task list could be this:

ID	Name	Type	Hea	State	Info
1	Stress	Stress test		Inactive	
2	XSL-Export	Export		Inactive	
3	Export-Test	Export	●	Waiting	
5	Import-Test	Import		Inactive	
6	AdCom-Import	Import		Inactive	
7	Selector-Import	Import		Inactive	
8	Def-Align	Align definitions	●	Working	
9	AbyPromo-Import	Import		Inactive	
10	CopyConvert	Copy/Convert	●	Waiting	
11	Checky	File check		Inactive	
12	Löschen	Delete	●	Waiting	
13	Checky2	File check	●	Waiting	
14	Sepl-Import	Import	●	Waiting	

The first column is a consecutive number. Local tasks start with number 1, globals with 1000. The fourth column shows an icon depending on the health of the task:

- No Icon: The task is inactive
- Yellow ball: The task is starting or stopping
- Green ball: The task thread is running, the task itself is either waiting or working
- Red ball: The task has a problem and is halted

If a red ball is shown you must look to the next column “State”. If it says “Incapable” the task will never start working because there is one or more condition for running which is not fulfilled. Most probably the task settings are wrong. To change any task settings, you have to stop the task first, because it is not allowed to change the parameters of a running task. But the state can also be “Hanging”, “Blocked” or “InTrouble”. In these cases the task will automatically be restarted after a small waiting time, typically a few minutes. This is realized by a “watchdog“ of the main thread. Every task thread has to send live messages to the main thread regularly. If they stay away, the task is treated as hanging. First the main thread sends the task thread the command to stop itself. If it does not react on this command either, it is simply killed and a new thread is started for this task. This function has been implemented for security reasons only. In normal operations such a task kill should never occur and it should always be taken seriously, especially because it is not guaranteed that the new thread can resume the operation.
The next column “Info” shows some task dependent information when the task is working.
One column is not shown here, it is the “DebugLog” flag. A check in this column shows the log level of the task which can also be changed with a context-sensitive menu of the task list while the task is running.

7.1.10 Macros and filter expressions

Some of the tasks (at the moment: export task, metadata scheduler and consistency check) support macros and / or expressions for filtering nodes. This allows limiting the handled nodes in some way.

The macro language is the same as for TurboPlayer. So, if you are familiar with it, you already know how to write macros for BUS, too. So far, only consistency check task uses macros, in which you can write multiple if-statements, define variables, perform computations, etc.

All other tasks - and also the show filter for consistency check - are only filter expressions for nodes. These expressions are a sub-part of the whole macro language. It is a condition - the part, which can be written within an if-statement. Such a filter expressions is applied to all nodes of a whole tree – normally a rundown – by walking through all nodes within the tree. For each node of type “Group”, “Story” and “Element” the filter applies the expression and gets a result which is either 0 (=false) or any other value (=true).

Pay attention when the node is being removed from the tree if the expression for this node yields true. When a node is removed, the filter can no longer walk through the children – the children are already gone, even if the expression would have yielded a different result for them than for the deleted node.

If a node is deep in the tree, it is possible to query properties of the parent or grandparent node, too. If you use a parent node, which does not exist, or a field, which does not exist, an empty string is assumed but the evaluation of the expression is not stopped with an error. An error occurs only if you write a function name which does not exist or otherwise an invalid syntax.

Here is a description of the syntax of the macro language:

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 writing 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.

Local variables:

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

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

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. 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()`
- `Proc22 ( Show, 1000 )`
- `Proc33 ( Start, \{{ Text, comma, or something \}} )`

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

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:

```
DataOfParent ( "Class" )
```

Defined macro function:

It is possible to define your own macro functions. This can be done on each block (=stack level). 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 IsNewsOrMusic ( )
{
    return Data ( "Class" ) == "News" || Data ( "Class") == "Music"
}
```

Defined macro functions are available in BCS modules version 6.0 or higher.

Intrinsic functions

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”, “on”, “2”, non-empty string, any number != 0 • 0 for: “false”, “no”, “off”, “0”, “”, 0

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 ) )
```

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

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 ( Data("MediaType") == "Audio" )
{
    …
}
else
{
    …
}
```

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.

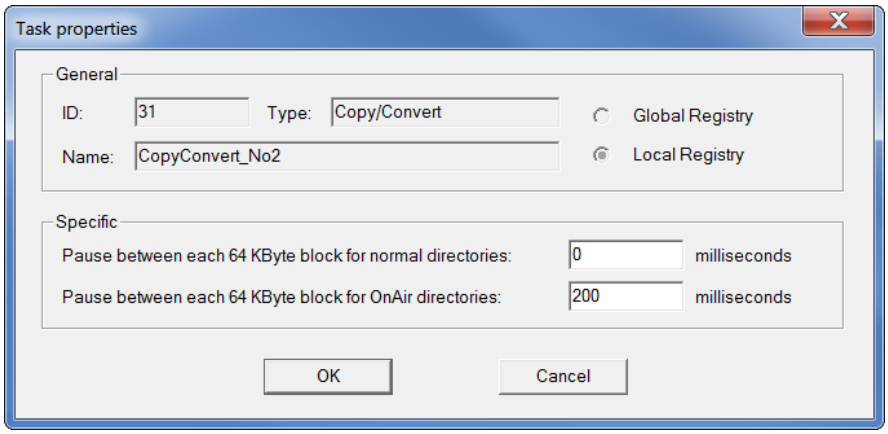
Extensions within BUS:

So far the description of the general macro language. In addition, the following extensions, functions and “local variables” can be used for the conditions in BUS:

`%<Field>`	Within pure filter expressions (=conditions) local variables are interpreted as the specified field for the node to be filtered. For example “%Title” is the content of the field <Title> of the current node. This is identical to the function Data (“Title”). Within complete macros you can use local variables regularly, meaning: define them in a macro and assign a value and then you can later make use them. Pay attention within defined macro functions! Do not define a function parameter with the name of a metadata field (e.g. "Title") or you will be confused, whether the function parameter or the metadata field is being used, when you write the variable name. (Hint: function parameters are preferred.)
`Data ( <Field > )`	The content of <Field> of the current node / the node to be filtered, e.g. Data (“News_Author”).
`DataOfParent ( <Field> )`	The content of <Field> of the parent node of the current node / the node to be filtered, e.g. DataOfParent (“RegioBlock”)
`DataOfGrandParent ( <Field> )`	The content of <Field> of the grandparent node of the current node / the node to be filtered, e.g. DataOfGrandParent (“AllowedClasses”)
`Tagname ( )`	The tagname of the current node / the node to be filtered.
`TagnameOfParent ( )`	The tagname of the parent node of the current node / the node to be filtered.
`TagnameOfGrandParent ( )`	The tagname of the grandparent node of the current node / the node to be filtered.
`HasChildren ( )`	Returns 1 (=true) if the current node / the node to be filtered has at least one child node.
`DataOfTrack ( <Field> )`	The content of <Field> of the track node – if the node to be filtered is within a track. Otherwise an empty string. Example: DataOfTrack (“Number”)
`DataOfShow ( <Field> )`	The content of <Field> of the show. This function is only supported within the full macros of consistency check task.
`strcmp ( "*", s, o )`	All these functions are identical to the intrinsic functions but if you specify a star as the first parameter, the function operates on *_all_* fields of the current node to be filtered. If there is a positive result for one of the fields, the function stops evaluating further fields and returns 1.
`strsearch ( "*", s, o )`	
`regex_match ( "*", r, o )`	
`regex_search ( "*", r, o )`	

7.2 Copy/Convert Task

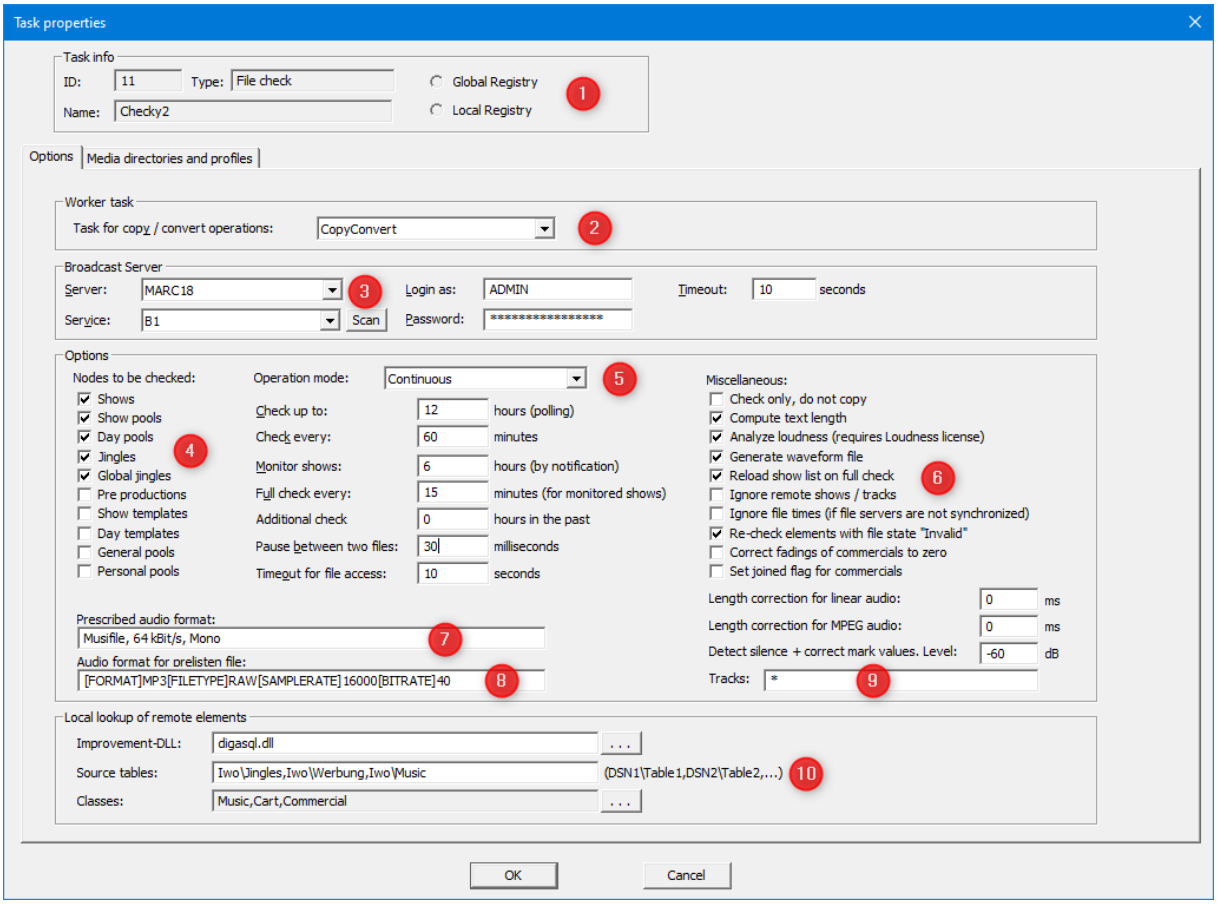
The copy/convert task is the easiest of all tasks. There are nearly no parameters and the task does never start working itself. It is only a helper task for the filecheck task. Whenever the filecheck task has to copy or convert a file, it sends an order to the copy/convert task and monitors this order until it has been finished (successfully or not). Therefore, a filecheck task can start only, if there is a copy/convert task running to accept orders. If you start the tasks manually, you must pay attention to this fact. If BUS starts the tasks automatically, it starts them in the right order.
There are two reasons why copying and especially converting has been implemented as a separated task. The first one is that copying and converting can take a very long time. If the Check task did the file handling itself, it could not treat other entries in this time. But in the current implementation the check task only has to put an order in the copy/convert queue and can continue to check other files. This is especially important if there are some changes (by other clients) during a long copy/convert operation. E.g. it is possible that BUS is just in a very long (but unimportant) convert operation when someone inserts a new entry only minutes before it is going to be sent. There you have the second reason why the tasks are separated: now the check task is notified about the new entry and can decide that the treatment of the new entry is much more important than for the long entry. Therefore, the copy/convert task has to interrupt its current operation and will treat the files of the new entry. When it has finished it will continue with the long operation. The priority of treatment is calculated by the difference of the current time and the start time of an entry. Entries in the past have the lowest possible priority. For a maximum effect you should start only a single copy/convert task per computer (or possibly per processor) because only then it is guaranteed that the computer uses as much resources as possible for important files if necessary. Otherwise multiple copy/convert tasks could try to take away processor time from a task treating an important entry.
The dialog of the copy/convert settings is very easy:



There are two fields to specify the pause between each 64 Kbyte of copied or converted data, one for normal and one for OnAir directories. A sensible value is some hundred milliseconds. Normally it is not necessary to set any pause value but if you encounter any network problems of if the copy/convert tasks needs to much processor time it can be useful to reduce the speed.

7.3 File Check Task

Before reading this chapter, you should check out the chapter “File Management in DigaBroadcastSystem” for a general overview of the media file handling.
The file check task is responsible for checking, copying or converting of all media files. It runs in multiple steps. After each step it sets the value “FileState” of the corresponding entry and continues with the next step. First it checks whether a specified file exists and sets the state to “existing” or “missing”. Next it checks the content and sets the state to “accessible” if the content is o.k. or “invalid” otherwise. If the file is located in a wrong directory the file is copied to one of the media directories first. And if the audio format does not match the prescribed audio format it is converted (to a media directory too). If there are additional media directories the file is copied to them in the next step. Last, corresponding text files are copied too. If any of these files changes later, the process starts anew. There is a special treatment for “virtual” and “recording” files which are surveyed continuously, to make them “existing” when they stop increasing. Remote elements are also handled in a special way. Now let’s have a look on the parameters of the check task:



(1) The task info describes the number, name and type of the task and its storage location in the registry. These values cannot be changed after a task has been created.

(2) An essential parameter you must set is the worker task for copy / convert operations. As described above it is absolutely necessary to assign a copy / convert task because the check task does only delegate these operations. Normally you should have only a single copy / convert task running and you can pick it.

(3) The settings for the broadcast server should be quite clear. After entering the password, you can press the scan button “…” which will query the available services from the server. Now you can select one of them in the service combo box. The timeout value applies to the TCP/IP connection to the broadcast server and can also be 0 - which means infinite.

(4) Here are all rundown nodes in BCS listed which can be checked. The task operates only on the selected nodes.

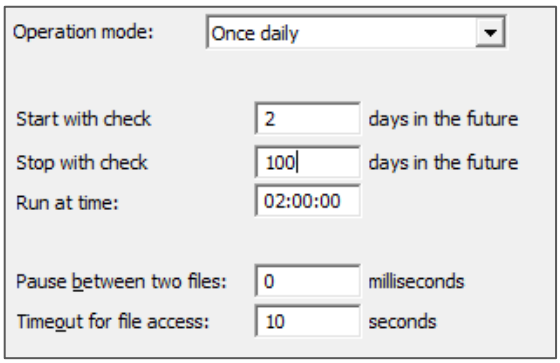
(5) An important combo box is the “operation mode” in the next group. Three modes are available: “continuous”, “once daily” and “Externally triggered”. Continuous is used to check elements, beginning with the present show until some hours or perhaps one or two days into the future. The “once daily” mode is used in a second check task typically, complementing a first, continuous task. It can run in the night and it can check very far into the future. In mode “Externally triggered” the task only connects to the specified BCS and then waits for an external

request to check specific nodes. Such a request can be sent from DigAIRange (context menu of the tree view) or from TurboPlayer (macro command TP_TriggerBUSFilecheck). Depending on the selected operation mode the scan options on the right part of the dialog change. The screenshot above shows the options for the continuous mode.

The options need an explanation. Of course, not all files, which are referenced by entries in the whole schedule can be checked, there are too much of them. The file check is limited to an interval, which starts a few hours in the future and ends at the current show. In addition, the interval is separated in two parts: One part which is nearer to the current show and a second part which is more in the future. The second part is checked only periodically (polling), in the example above every 60 minutes. The first and more important part is checked more often: this part is registered for notifications from the BroadcastServer, so every change in this interval is immediately reported to BUS. That means for most events BUS will be informed and can check new or changed entries at once. Unfortunately, there are some events which do not produce notifications from the BroadcastServer. E.g. if someone edits a text file with a non-client tool as MS-Word. The tool will change the text but no notification will be generated and BUS does not perform any treatment of the changed file. Therefore, all entries within the first interval can be checked periodically, to detect such changes too.

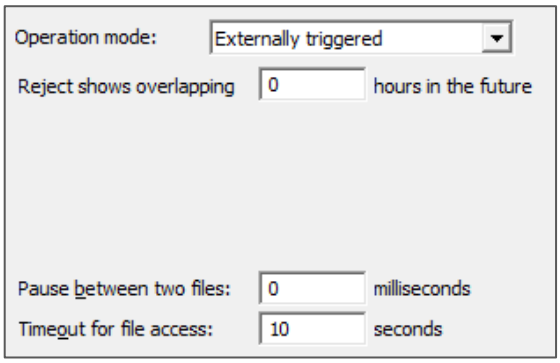
Now for the parameters: the first two fields specify the long interval in the future and how often the entries are checked. The time should not exceed some days and the interval should not be smaller than some 10 minutes, because such a run normally means checking thousands of files, which can be a heavy network load. The same applies for the next two values which are the same parameters but for the smaller interval near to the current show. Sensible values are shown in the example above. The fifth option is an interval into the past. Normally a check of elements in the past is not needed so you can set this value to zero. The pause between two files can be used to slow down the task. The file timeout must be specified and is used to limit the blocking time during file existence checks.

If the operation mode is “Once daily” the scan options are different. Here is the screenshot:



The task will check one day after the other, beginning with the first one, specified in the first field and ending with the last day, specified in the second field. You should not set the first day to 0 if you have another check task running for the same program to avoid checking the same elements twice. The third field is the daily time when the task starts with checking.

If the operation mode is “Externally triggered” the scan options look like this:



If you set the “Reject shows…” field to 0 as in the screenshot, the option is not active and the task will accept all shows to be checked. If you have another FileCheck instance running in continuous mode, you should set the “Reject shows…” value to the same value as the value “Check up to … hours (polling)” of the continuous task. Then the externally triggered task will reject requests for shows, which start before the time <now> + the specified value. This avoids that two FileCheck instances work on the same show. Pay attention to set the desired “Nodes to be checked”, because BUS rejects requests for node types, which are not checked.

(6) The miscellaneous part on the right side of the dialog has many options:

Check only, do not copy:
Check, if you do not want the FileCheck task to copy any files into the media directories. If activated, media files are analyzed only but not copied or converted (exception: remote files). Therefore, this option cannot be used if you want to prescribe an audio format or allowed directories.

Compute text length:
If active, the task performs a text length computation for all text elements and for the additional presentation text of all other elements. (The additional text of group and story nodes cannot be treated.) The task takes the current speaker of the element or of any parent nodes into account. Desktop applications like DigAIRange compute the text duration themselves. This option is intended to be used with web clients which do not handle text length computation. BUS uses the private metadata field TLCI to keep track of elements for which the text length was already computed.

Analyze loudness:
This activates a loudness analysis. It is only available if you have the loudness license dll ("BcsLoudness.dll") and the dll with the loudness algorithms ("LoudnessDll.dll") installed. The analysis is done for all audio elements which do not have the loudness values filled out (which can happen e.g. if the element has already been analyzed by the loudness workflow of DPE system). After a successful analysis the 5 loudness fields "Audio_ILK", "Audio_LRA", "Audio_DBTP", "Audio_MAXML", "Audio_MAXSL" are filled.

Generate waveform file:
It is possible to activate the generation of waveform files for the elements. Please note that this will only happen if the waveform file is not present yet in the fields MEDIUM of the element. If it is, the path of the waveform file will simply be copied to the "File_Waveform" field. This re-use of the filename form the medium list is not done if the filename does not point to a defined media directory and the option "Remove filenames which do not point to a defined media directory" is active.

Reload show list on full check:
This option should be activated if your shows change very often, especially if new shows are created. Otherwise BUS will not see the new shows. This option will also help if you are using media directory profiles (see below) which are defined in the metadata of shows. If the profiles of shows can change after the task has seen a show for the first time, it is necessary to reload the show list on a full check and possibly to set a small value for the "Full check every … " parameter.

Ignore remote shows / tracks:
Remote shows / tracks are hosted somewhere else and are only mirrored to the local BCS. Normally these shows / tracks are treated by FileCheck, too. The task will copy the referenced files from the remote system to a local media directory, so that the files are available locally. Sometimes it is not necessary to get these remote files, e.g. if you use remote links only to watch what is going on on a remote BCS system but if remote shows are not being played on the local system. Then you can set this option.

Ignore file times (if file servers are not synchronized):
This option applies when the task has a list of multiple filenames (mirrored ones) and has to decide which one is the "newest" (the source for any copy operation). By default the task looks at the last write access of the files and selects the newest one. But if the clocks of your file servers can differ, this might be the wrong decision. Therefore, you can check the option and the task will only evaluate the primary filename specified in the XML data in Tag "File_PrimFilename". This tag is written by all tools integrated in the BroadcastSystem. It will not work, if you edit files with external tools, like WordPad. They will only change the last access time but not the XML tag.

Re-check elements with file state "Invalid":
Normally the task does not check elements again if they have the file state "invalid". There is no reason to do this, because "invalid" is only used when an element has a media file and the file could be analyzed but this yield an invalid format. Re-checking will not change this fact, normally. If you have a very special setup or if you encounter problems with the analyzing you can check this option.

Correct fadings of commercials to zero:
If this flag is set, the task will look through all lists it checks and will change the link and fade values of commercial elements and the elements immediately before or behind commercials in a way that they are played without any fadings.

Set joined flag for commercials:
This will make the task to set the "joined" flag for the second element of each commercial-commercial sequence. This was intended to achieve a special behaviour in playout: all commercials are played on the same channel. This option has become obsolete because TurboPlayer can show this behaviour with the help of its "SequencedOnSameChannelClasses" parameter. In addition, the "joined" flag creates a special handling of each sequence in MultiPlayer which makes interruption of the sequence difficult or even impossible. Therefore: do not use this option any longer.

Length correction for linear / MPEG audio:
These two fields are correction values for MPEG and linear audio files, respectively. These correction values are applied after analyzing an audio file and before entering the evaluated audio duration into the BCS data. You can correct the values by a few milliseconds if you know that your playout tool adds some silence when playing a file. E.g. this is typical for MPEG decoder which always adds a silent frame at both file ends.

Detect silence + correct mark values:
If you set this value to something different than 0/<empty>, the task will perform an audio analysis to detect silence at the beginning and at the end. Afterwards the mark values are corrected so that the silence is excluded. If the task finds sensible mark values within the audio (e.g. because a user has already set "real" values) no correction is done.

(7) Another option is the prescribed audio format. This value is a list, separated by spaces or commas, of the following keywords: "Musicam", “MP3” or "Linear"; "Wave", "BWF", "Musifile", "Raw"; "Mono", "Stereo" or "Joint Stereo"; the sample frequency in Hz “16000”, “22050”, "32000", "44100" or "48000"; the resolution in bits per sample: "8", "16" or "24"; and the bitrate in kBit/s: “8”, “16”, “24”, "32", “40”, "48", "56", "64", "80", "96", "112", "128", “144”, "160", “176”, “192", "224", "256", “288", "320", “352", "384". You can add the units "Hz" or "kBit/s" if you like. The order does not matter. An example: "Musicam, Wave, Stereo, 48000Hz, 256 kBit/s". If you do not specify a category, it is kept unchanged. Example: if you specify "44100 Hz", only the sampling frequency is forced to be 44.1 kHz, all other characteristics are not changed. The field can be empty too and the files are never converted.
Since version 5.0 BUS does also accept a description of the format as a section string. Like before, only those attributes, which are specified are changed, all others are kept. Here is a table with a short description of the possible options:

Section	Description	Possible values
[FILETYPE]	Format of file container	RAW, MUS, MUSIFILE, WAV, WAVE, BWF
[FORMAT]	Format of audio compression	LINEAR, MPEG (=MPEG-LAYER2), MP3 (=MPEG-LAYER3)
[SAMPLERATE]	Sample frequency in Hz	16000, 22050, 24000, 32000, 44100, 48000
[BITRATE]	Bitrate in kBit / second	8, 16, 24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160, 176, 192, 224, 256, 288, 320, 352, 384
[RESOLUTION]	Bits per sample	8, 16, 24
[MODE]	Used channels	MONO, STEREO, JOINT, DUAL, LEFT, RIGHT, 5.1, 7.1, 5.1 WITH STEREO, 7.1 WITH STEREO

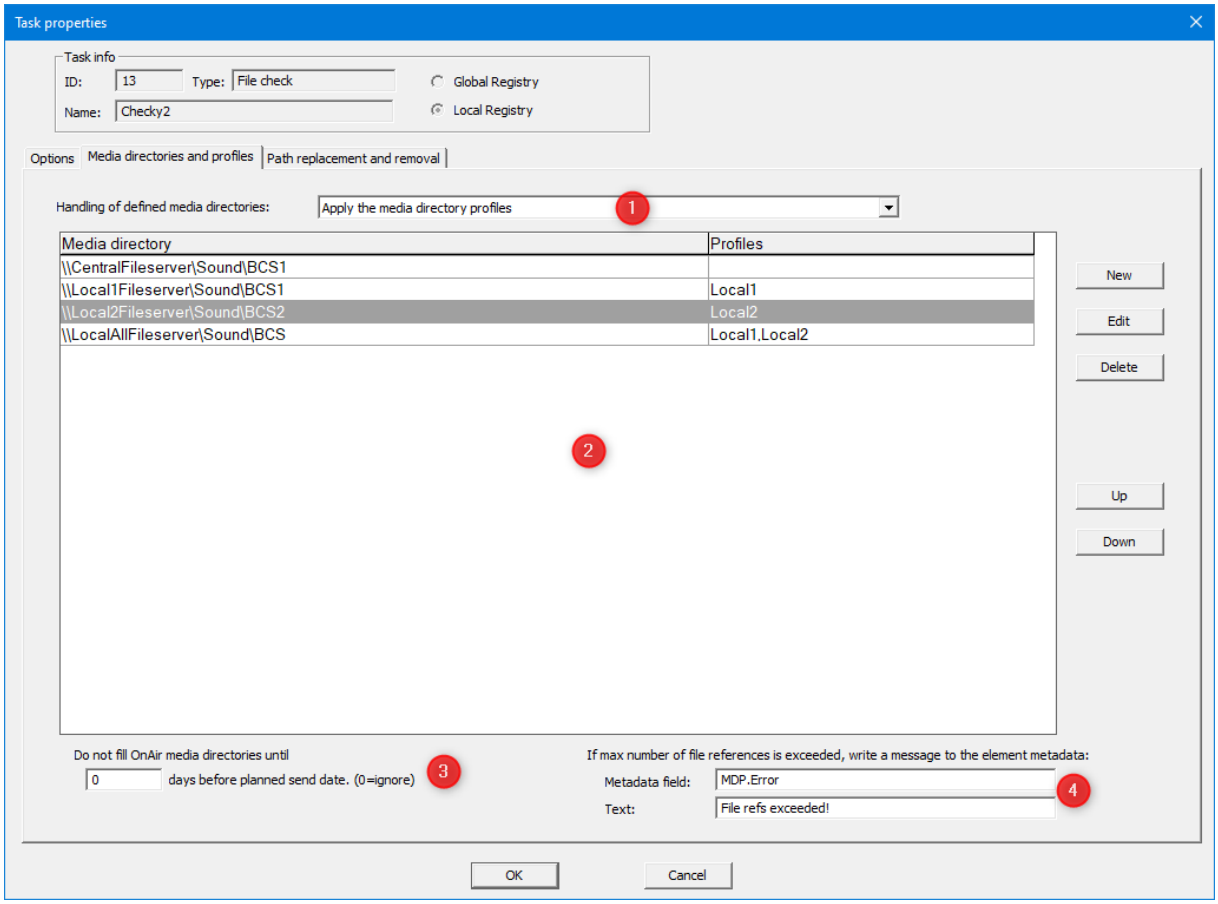
(8) Since version 5.0 BUS can also create “low-res” files of audio elements for prelistening. These files are created from the primary audio file by audio conversion. Typically, MP3 with high compression is being used, especially if the prelisten files should be usable via internet. The format of this setting is identical to the one for “Prescribed audio format”, therefore you can either use a pure string or a section string like it is described above. If you want to create MP3 files you must specify the file type “RAW”, you should specify the bitrate, you can optionally specify a samplerate and a mode but you should not specify a resolution. If you don’t want to create prelisten files leave this field empty.

(9) In the “Tracks” field you list all tracks on which the task should operate. It can be a comma-separated list of track numbers. Non-schedule tracks (with a number >= 1000) should not be treated, normally. The special joker value “*” means: operate on all schedule tracks (0-999).

(10) The last group in the dialog is "Local lookup of remote elements". It is useful if the task has to copy files from a remote sytem to the local system. Normally the task copies all files equally. Let's assume there are 30 music elements from the database in a remote show but only 5 elements which were produced for the show. If the music titles are already copied to the local system on database level (e.g. with DigAlign) the task need not copy all 35 files. It is sufficient to copy the produced files only and to use the music titles from the locally available database. This behaviour can be configured with the fields in the local lookup group. The improvement DLL will be the DigaSQL.dll normally. The "Source tables" are the local tables, which should be scanned for files and "Classes" are the classes for which the lookup should be done – e.g. Music. The task

will look for the usual ID fields: MusicID, CDID and Motive and will check in addition the first kByte of the file if it really is identical. If a file cannot be found in the local database lookup, it is copied to the local BCS system. Here is an example how a configuration would look like:

There is a second tab in the settings dialog which contains data specific to media directories:



Without the option of “Media Directory Profiles” the settings are reduced somewhat: you will not see anything related to these profiles in the names or settings and the second column in the list is also missing.

(1) This combo box allows to define what the task should do with the configured media directories. There are 2 / 3 options, the third one only if media directory profiles are available:

- **Use defined media directories as allowed directories**: If you do not list any directory, all are allowed. If you list some paths here, they must include every media directory which has been defined for the service (use the manage dialog of DigAIRange to define them). But this field can list more directories, typically some of the database directories. If a media file is not located in one of the allowed directories, it is copied (or converted) to one of the media directories of the service. Defining multiple media directories for a service means BUS will create a mirror file in each of them. Hint: If you have a single media directory BUS does not create a mirror file, this process starts with the second media directory. But if you want to force BUS to copy each file into this single media directory you must make it the single allowed directory.
- **The defined media directories replace the media directories of the service**: If you choose this option the list of media directories is interpreted differently: the task ignores all media directories which are defined within the BCS tree. Instead, only the directories listed in the task settings are used as media directories. This can be useful e.g. to have one task for each track of the shows and each track should have its own media directories.
- **Apply the media directory profiles**: This option is only available if the media directory profiles are activated with the corresponding license dll. The settings allow assigning a list of comma-separated profile names to each directory. Typically, this would be region names and you would list these names as profiles,

e.g: “Texas,Nevada”. The idea is that the assigned directory is the storage location for these regions and the task should only copy files to this directory if the files are needed for playout in these regions. How does BUS know in which regions elements are going to be played? At the moment it is only possible to define the playout regions in the metadata of a show or a jingle group. The field “MediaDirProfiles” can hold a comma-separated list of profile names, which can be used to define in which regions a show is going to be played. The task will copy files to all media directories with a matching profile name and will ignore all other. Exception: a media directory which does not have an assigned profile is always being used ! This does also apply for all media directories, which have been defined in the BCS metadata of the service or the show. Pay attention to always define at least one media directory for the service. This is needed for all other clients, because only BUS is aware of the directories and the profiles defined here.

(2) The list of media directories and optionally assigned profiles (see above). The buttons on the right side of the list allow to add, edit, remove or move list entries.

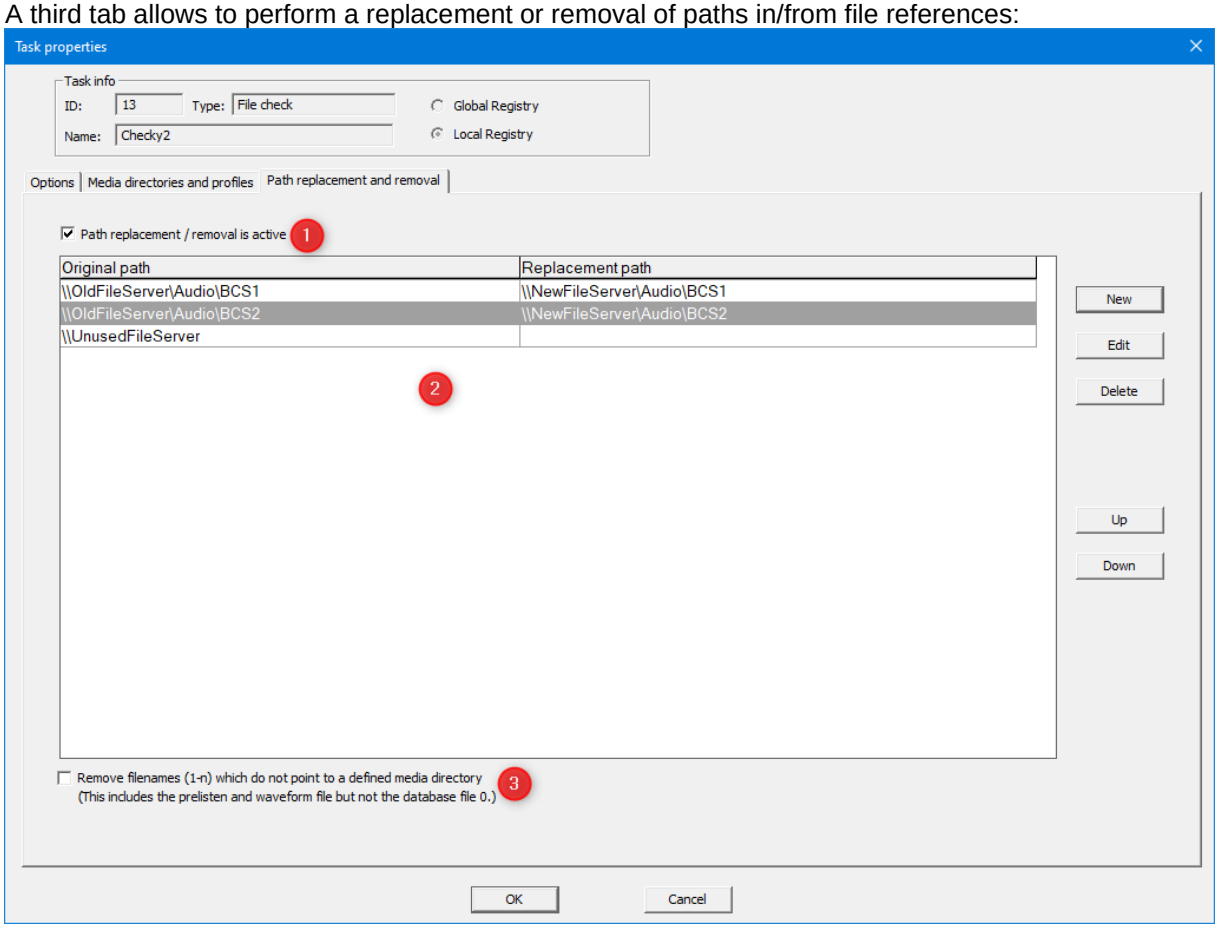
(3) There is one additional option:

Do not fill OnAir media directories until…:

This option is the limit when to start filling OnAir media directories. Such a limit can be useful, if you have big servers for the normal media directories, which can hold many days full with files, but you do not want all of these files to be copied to the OnAir computers too. If you set this limit to 1 for example, files will be copied to the OnAir directories only if an element starts today or tomorrow, if it starts two days or more in the future, only copies in the normal directories are created

(4) When you use media directory profiles, it can happen that too many profiles and/or media directories are configured. This is no problem as long as not too many of them are used for a show. If too many are used it is no longer possible to create file copies in all desired directories, because in BroadcastSystem it is only possible to store up to 10 file references for each element. If that limit is reached, BUS will stop creating additional file copies and the file state will never be set to "Mirrored". In addition to an error message in the BUS log it is also possible to set an arbitrary text in one of the metadata fields of each affected element. In principle, you can use any suitable field, but it is strongly recommended to use a custom specific field, like "MDP.Error". This field can then be included in the grid view of DigAIRange and is easily visible.

A third tab allows to perform a replacement or removal of paths in/from file references:



These settings are intended for a correction of file references, e.g. in case of moving media directories from one server to another one. If this is active, the FileCheck task will compare the paths of file references with the specified paths and in case of a match the path will be replaced. Some details:

- A comparison of the paths is done before any other action is performed for the element. In case of a match the metadata of the element is first updated on the BCS and only then the regular FileCheck operations start.
- The comparison is a pure case-insensitive text comparison. The path is compared from the beginning of the text. Jokers like ? or * cannot be used.
- All fields of the file part ("File_...") and of the medium part ("MEDIUM….") are evaluated. Other fields are not evaluated. Especially the RemoteFile part is not evaluated. Reason: remote file references are implicitly generated by BCS when translating data for a remote system. They must only be modified by BCS and will automatically change if the corresponding values are changed on the remote system.
- This feature will modify the paths for all elements, which are checked according to the settings of the task. This will leave elements with "wrong" file references in the BCS tree, but this are normally elements in the past or in the far future.

(1) You need to explicitly check this option in order to activate path replacement. If you uncheck the option, you can temporarily deactivate path replacement without loosing the settings in the list.

(2) In this list you can define pairs of paths. The left column shows the original path as it might be present in the metadata of elements. The right column describes the path, which is entred to the metadata instead of the original path in case of a match. If you leave the replacement path empty, the task will remove any file reference, which uses the original path of this line.

(3) This option is independent from the path replacement settings. It is also not necessary to activate box (1) to make this option work. Here is what this option does: Whenever the task finds a filename which is located outside of the defined media directories it is removed. The file itself is not removed, of course. This rule does apply for the waveform and the prelisten file, too, but it does not apply for Filename0, as this points to a database directory which is never part of the BCS media directories. And this option does not mean that a filename is removed when a file is not available. A simple string comparison is done to decide whether a path is a media directory or not.

Distribution Endpoints

The settings regarding distribution endpoints cannot be configured in the settings dialog. Nevertheless, the FileCheck task evaluates a value in the settings for distribution endpoints in the DigaSystem registry:

```
Common\DistributionEndPoints\...\FileCheck
```

This value can be a comma-separated list of operations, which are executed by FileCheck task on each element. Possible operations are:

```
Mirror,Loudness,Waveform,Prelisten
```

These values correspond to: creation of mirror files, loudness analysis of audio files, creation of waveform and prelisten files.

Each element can have one or multiple distribution endpoints listed in the metadata field “DistributionEndpoints”. The FileCheck task evaluates this field and then picks the desired operations from the registry. If an element has multiple endpoints, the sum of all defined operations is executed. You can also use the special value “All” or “*” for the list of operations, which means: execute all possible operations. If an element has no field DistributionEndpoints or if this field is empty, it is assumed automatically that the endpoint for TurboPlayer playout is meant (this is done for backward compatibility). If no such endpoint was defined in the registry, “All” operations are executed.

All these operations will only be executed, if they have been generally activated in the settings of the task. These are the settings:

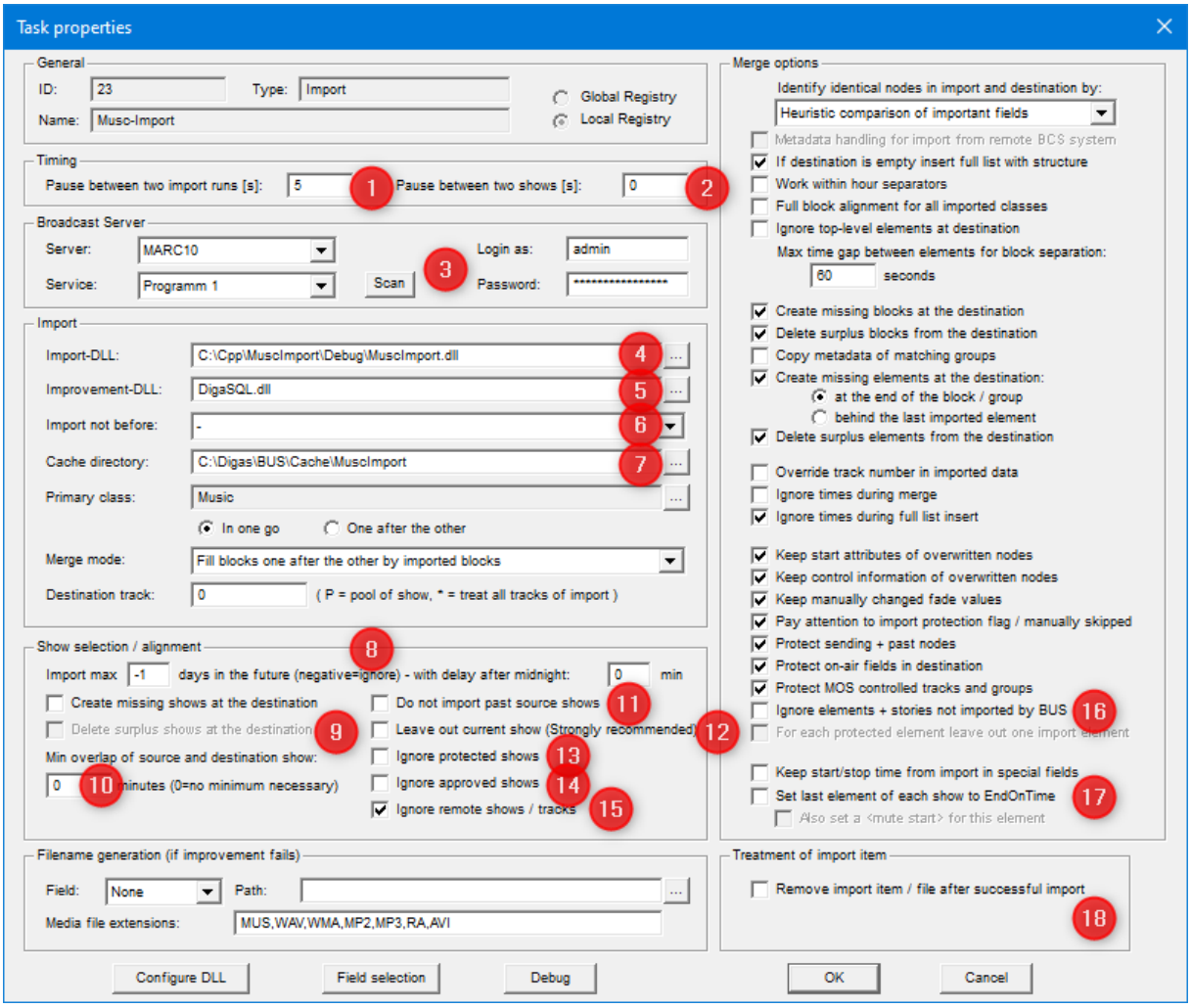
- Mirror: In order to get mirror files, you need to define two or more media directories.
- Loudness: Activate the setting “Analyze loudness”.
- Waveform: Activate the setting “Generate waveform file”.
- Prelisten: Define a format for the prelisten files in the setting “Audioformat for prelisten file”.

In addition, do **not** check the setting “Check only, do not copy”, because if this setting is active, only operations, which do not create new files are executed (loudness analysis).

7.4 Import Task

Before reading this chapter, you should check out the chapter “The import function” for a general overview how the import in the DigaBroadcastSystem works. This chapter also describes most of the possible settings for this task. We discuss only special BUS settings here.

BUS is responsible for the “full-automatic import” which runs in the background without any user interaction. Here is the dialog to change the import settings:



1) Pause between two import runs:

Every x seconds the import task will poll the import DLL, which will return a list of all import items it knows about, with a change counter for each item. Each item, which is new or has been changed is (re-) imported. An example: an item can be a file in a specific directory and the change counter can be the last write time. Each item contains one or more shows. The import task tries to find a corresponding show for each show in the item and merges them. Do not specify a too short value because this can generate a very high BUS workload.

2) Pause between two shows:

Here you can specify a pause time to reduce the workload for the BUS and the BCS. The pause is applied between two shows in the step “show structure alignment” and once again in the step “merge content”.

3) Broadcast Server:

Specify the server and service to connect to and the login data. The user need not be the Admin, but he must have the right to write the imported class at least.

4) Import-DLL:

Specify the import module to be used. Be aware that not every import DLL can run in full-automatic mode. The DLL must provide some special functions and it must be possible to configure it in a way that it does not have to show any dialog boxes.

5) Improvement-DLL:

The improvement DLL will be the DigaSQL.DLL typically. You do not have to specify its full path, because the instance already loaded into memory is used anyway.

6) Import not before:

Here you can select one of the other defined import tasks. If selected, the task will not treat a show until it was filled by the other import task. This is useful if the other task is the one which creates the show structure and this task only fills remaining placeholders. E.g. the first task is the music import which also imports empty groups for the commercials. Then the second import for the commercials must wait until the music import did run.

7) Cache directory:

The load for the Broadcast Server is kept as low as possible: Bus will write the whole data of an imported show into its cache directory. Whenever an import DLL reports a show as changed, the show is reloaded from the DLL but it is transferred to the server only if it differs from the cache data. The cache directory must be created for this task only, it must not contain any other files or the import task will refuse to use it.

8) Import max ... days in the future ... with delay after midnight:

Normally the task imports any item it finds, independent of the date. But sometimes you want to limit the imports up to only a few days in the future. This might be necessary if the day templates of BCS are only applied for some future days.
If you have multiple import tasks with the same limitation of days in the future, a problem might arise: shortly after midnight the tasks might want to import the items which just passed the day filter. If many tasks do this at the same time, a very high BCS load might be the result. To avoid this, you can give each of these tasks a different delay time after midnight, which causes the tasks to run at different times.

9) Create missing / delete surplus shows at the destination:

By checking these options, you activate the show structure alignment. Some of the options in the part “show selection / alignment” will disappear, others will appear. See the screenshot below.

10) Minimum overlap of source and destination show:

This option is only available if show structure alignment is not active.
This parameter might be helpful in case the show structure of the import and the destination are different. As said in the general chapter about imports you will get the best results when the show structure in import and destination is the same, but sometimes this is not possible. A typical case is: you want to have multi-hour shows in BCS but your music rotation handles only one-hour shows. This is still quite easy because the border of imported shows is at each full hour, as is the border of the BCS shows. The BUS task will pick the right shows automatically. But in case the borders of the imported shows do not match the borders of the BCS shows, you might need to configure how big the overlap of import and destination show must be to make the task run on a specific pair of shows. E.g. let's assume the music rotation yields shows which are longer than an hour because of reserve elements which start in the next hour and which stretch the end of the show. If you import such a show into one-hour shows on the BCS a music show will be merged twice: once into the hour for which is intended and once into the hour afterwards (because there is an overlap for this hour, too). Normally this behavior is unwanted. Here you can eliminate the second merge by defining that the minimum overlap time must be 59 minutes (assumed that the backlog of the music show cannot exceed 59 minutes, of course).

11) Do not import past source shows:

Normally BUS will import every show it gets from the import DLL, it does not care whether it is in the future or in the past. If this behavior is undesired, check the box and BUS will ignore each show, which lies in the past.

12) Leave out current show (strongly recommended):

The current show needs an extra setting: “Leave out current show”. As you can see in the dialog, it is strongly recommended to check this box. Here is the reason: During import, BUS locks the whole destination show, merges it with the source show and writes the show back to the server. Typically, the whole show will be locked for a few seconds. If this happens at an inopportune time (and Murphy says it will) the OnAir player might be disturbed and undesired behavior can be the result. You should always manually import individual elements into running shows. Therefor you must use DigAIRange, of course.

13) Ignore protected shows:

You can check this option to avoid that the task imports into a show, which has already been declared as "approved & protected". Since version 4.9 of the modules a show can also get the flag “Import protection” assigned in DigAIRange. Regarding this option both fields work identically and a show, which has either the protected state or the protection flag set, will be ignored during import.

14) Ignore approved shows:

You can check this option to avoid that the task imports into a show, which has already been declared as "approved" or “approved+protected”

15) Ignore remote shows / tracks:

If checked the import will ignore all shows, which have links set to a remote BCS. If the show itself is not a remote show but only some tracks have remote links, the behavior depends on the step: the show structure alignment step will ignore shows with at least one remote track. The merge step will import into all tracks, which are not remote and will ignore the remote tracks.

16) Ignore elements + stories not imported by BUS:

All elements, which were imported by BUS get a data field: ImportedBy="BUS\<Import module>...". If you check this option, the import task will exclude all elements, which do not have this field or the field data does not start with "BUS". This is useful in case the import task will perform re-imports and the users are allowed to insert elements with the class of the import manually. Without this option, the elements inserted by the users will be overwritten by a re-import.

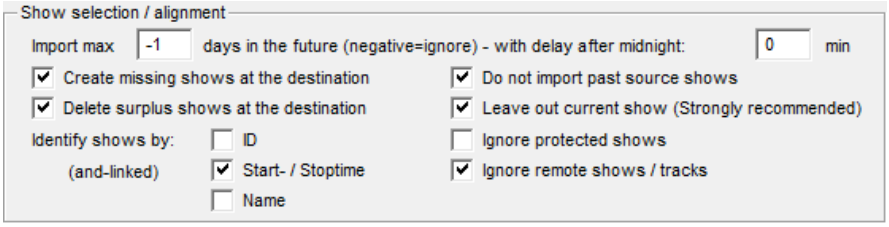
17) Set last element of each show to EndOnTime / mute start:

After a successful import the task can set three fields in the last element of the show: Time_StartMode="EndOnTime", Time_StartType="FixedEnd" and – if desired - Time_MuteStart="1". The end time will be the end time of the show. If the last element is exchanged the start attributes of the formerly last element will automatically be reset to: floating/sequenced/no mute start.

18) Remove import item / file after successful import:

After a successful import, the task can remove the imported item. As BUS does not know where the data came from, it cannot delete anything. This is done by the import DLL on a request from BUS. Therefor the DLL must support the deletion of imported items. If the DLL does not support this, the check box will by grayed out.

If you check one of the options “Create missing / delete surplus shows” the show selection / alignment part looks like this:

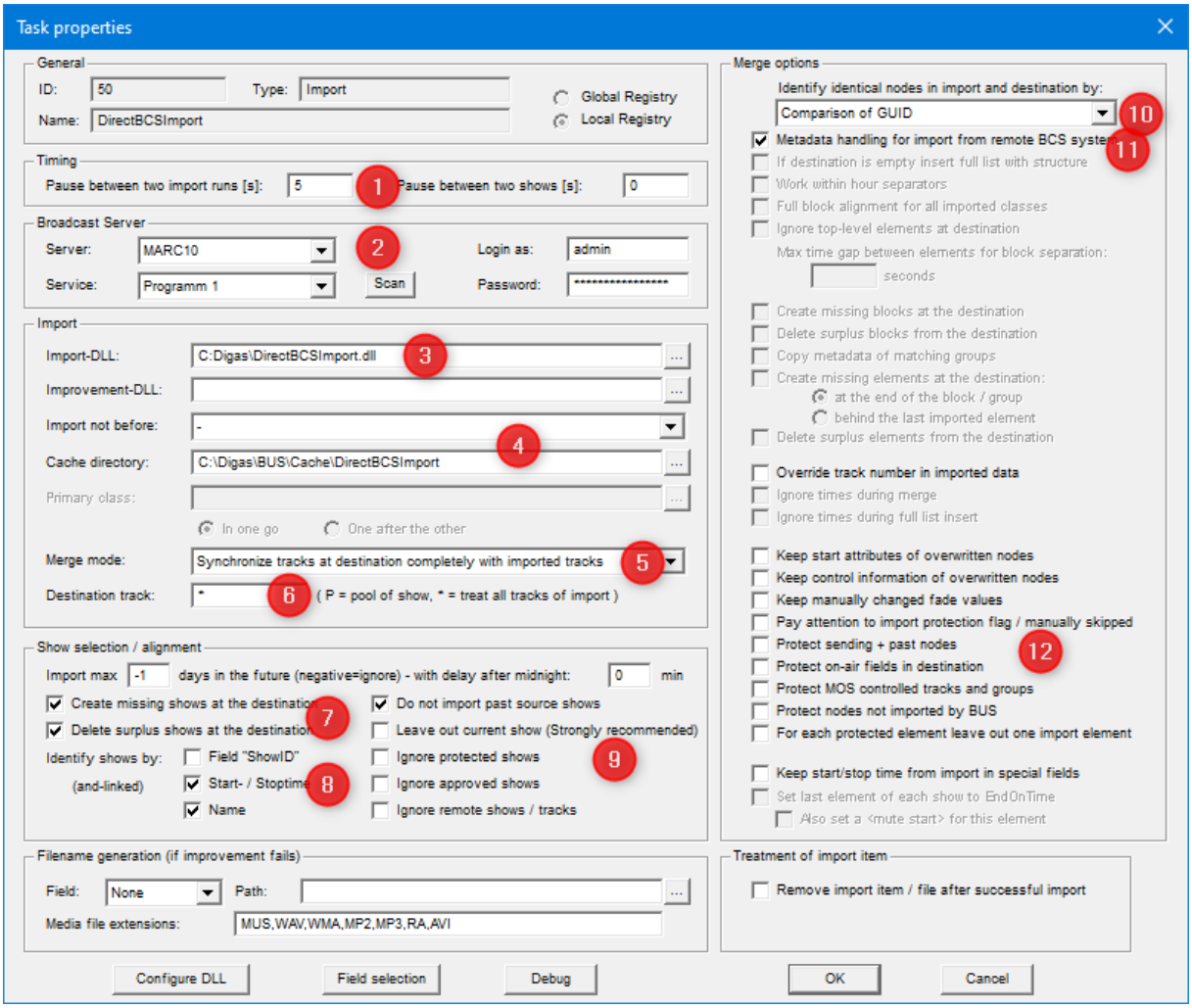


These options activate the “show structure alignment” which is described in chapter 5.1.5. above. In the screenshot you can see that the option “Minimum overlap of source and destination show” did disappear and instead you can select the metadata fields which are used to identify identical shows on import and destination side. If you check multiple criteria, all of them must be fulfilled.

7.4.1 Using Import Task for duplicating shows to a remote BCS system

Given the situation that you want to copy your rundowns including all referenced media files from one master location to another one in order to have a backup in case the master location fails completely and is no longer able to play out anything. Then the backup location can take over and start broadcasting. This chapter describes how you can set up a synchronization process between the master and the backup location with the help of the BUS Import task.

Here is a screenshot with the settings of the BUS Import task with the recommended settings. Below you can find many detailed explanations for the individual settings.



1) Pause between two import runs

Please have in mind that using the import for a rundown synchronization can create a quite heavy load for the destination BCS. Reason is that even minimal operations on the source side (like a single UpdateNodeData changing only one field of one element) will create a whole cascade of operations for the destination side:

- The show node is locked.
- The show node is read by the import task in order to work on current data.
- The import task performs the synchronization of the source and the destination rundown.
- The modified show node is written back to the destination BCS as a full tree update.
- The show node is unlocked.

If the destination BCS is powerful and does not have a lot of other active clients, it might not be a problem to run the import every second. (The configured value is only the minimal interval if permanent changes on the source side occur.) We recommend to use a bigger value, like 10 or 5 seconds. If you really want the quickest possible synchronization, use 1 second but then make some tests with typical shows first: modify the show permanently on the source system and monitor the BCS CPU load on the destination system and perhaps with a DigAIRange, displaying the destination show, whether the synchronization can be handled in an acceptable way.

2) Broadcast Server

Here you configure the destination BCS.

3) Import-DLL

As import dll you should use the DirectBCSImport.dll. It can directly connect to the source BCS and you do not need to use an indirection like: export on the source side into a text file and import this text file on the destination side. The configuration dialog of the dll is described above in chapter 5.2.9.

4) Import values

Don't configure an "Improvement-DLL" or the setting "Import not before".

An improvement of metadata should not be needed, because it is assumed that the source is a BCS system, which already holds all needed metadata. In case you think about avoiding unnecessary file transfers because you have a synchronization between main and backup location on database level: please check the settings "Local lookup of remote elements" in the FileCheck task.

An exclusive cache directory is needed like for every import task.

5) Merge mode

The merge mode is essential! Always use "Synchronize tracks at destination completely with imported tracks". Other modes do not offer the required detail options.

6) Destination track

Typically, you will want to copy all tracks from the source show, then enter "*". Otherwise, specify only the tracks you want to be imported.

7) Create missing / delete surplus shows at the destination

This is essential in order to get the show structure copied to the destination. You can only omit it, if you have all destination shows already existing – e.g. because the templates on the source and the destination BCS are identical.
Please be aware that the two options only include creation and deletion of shows. It is not guaranteed that any other changes of a show on the source side (e.g. modifying the show length) are copied to the destination.
Another hint: in order to reduce the load for the destination BCS, the Import task does not perform show deletion immediately when a source show is deleted, because this would require too much polling of the currently defined shows. There will be a delay of 0-10 minutes until a show at the destination is deleted.

If you want to exclude some shows from replication, you can use the filter option of the DirectBCSImport.dll.

8) Identify shows

It is recommended to use the start/stop times and the name of a show for identifying identical shows on the source and the destination side. This should be o.k. for most scenarios.

9) Show alignment options

If we assume that the destination side should strictly follow the source side, there must not be a protection of any shows on the destination side – except the past shows. In contrast, handling of past shows is not needed normally.

10) Identify identical nodes in import and destination by

This is a very important setting! Reason is that the import task must be able to detect elements, which have already been copied to the destination and for which local files have already been created. Only if the task can identify these identical elements, it will keep the filenames (and other states) on the destination side. If this does not work, the import will overwrite the local file references with every update (which can take place very often) and the FileCheck task at the destination side has to recreate the files again and again.

The best choice for the setting is "Comparison of GUID". The GUID is existing for all nodes in the BCS tree and it is kept even if you move nodes outside of their hierarchy. Possibly, using "Heuristic comparison of important fields" might also work. Or better: it will work in most cases for sure and if sometimes file references get lost it might be bearable. Other choices do not work well. E.g. using the node ID of the source might seem to be natural, but the node ID changes too often when changes in the BCS tree are made. Other choices will probably be ambiguous in a typical rundown.

11) Metadata handling for import from remote BCS system

If this option is not activated, there will not be any special handling of the filenames (and some other fields). As a result, the filenames will appear in the destination metadata as they are in the source metadata. This is a valid choice if you do not want to get the referenced files copied to the backup location, because the file references will then point to the files in the master location.

If this option is activated, the import task will copy all file references as remote file specifications into the destination metadata. E.g. the field “File_Filename1” will be copied as “RemoteFile_Filename1”. The FileState on the destination is set to “Remote”. These remote file specifications will not be visible to any regular BCS client. Only BUS FileCheck task can evaluate them and copy the files from the master location to the backup location. This file handling is identical to the one, which you get if you are using remote show or track links when using regionalization. This is described in detail in chapter 11.3.

It should be clear that there must always be BUS FileCheck tasks running on both sides to make all the file handling work properly!

12) Protect sending + past nodes / Protect on-air fields in destination

In order to get a real synchronization, you should not set any of the options, which protect elements or the metadata of elements on the destination side. An exception are the two mentioned options. If you want the backup location to only monitor passively what’s going on in the source side but not play out at the same time, then don’t check the two options.

This is different, if you want to have a simultaneous playout on both sides. In this case it would disturb the backup TurboPlayer if any on-air relevant metadata fields (like the SendState or the real times) are copied from the source to the destination. This can be avoided by checking the second option. Then all on-air relevant fields from the source are filtered out and are not copied to the destination. This affects the SendState, the real times, the OnAirPlayed… fields and the very special fields “Time_IIRS” and “TurboPlayerGUID”. For the SendState there is a special treatment: the state “Skipped” is copied to the destination, if the skipping was done explicitly by a user on the source side. The SendStates “Placeholder”, “Planned” and “Cleared” are copied, too.

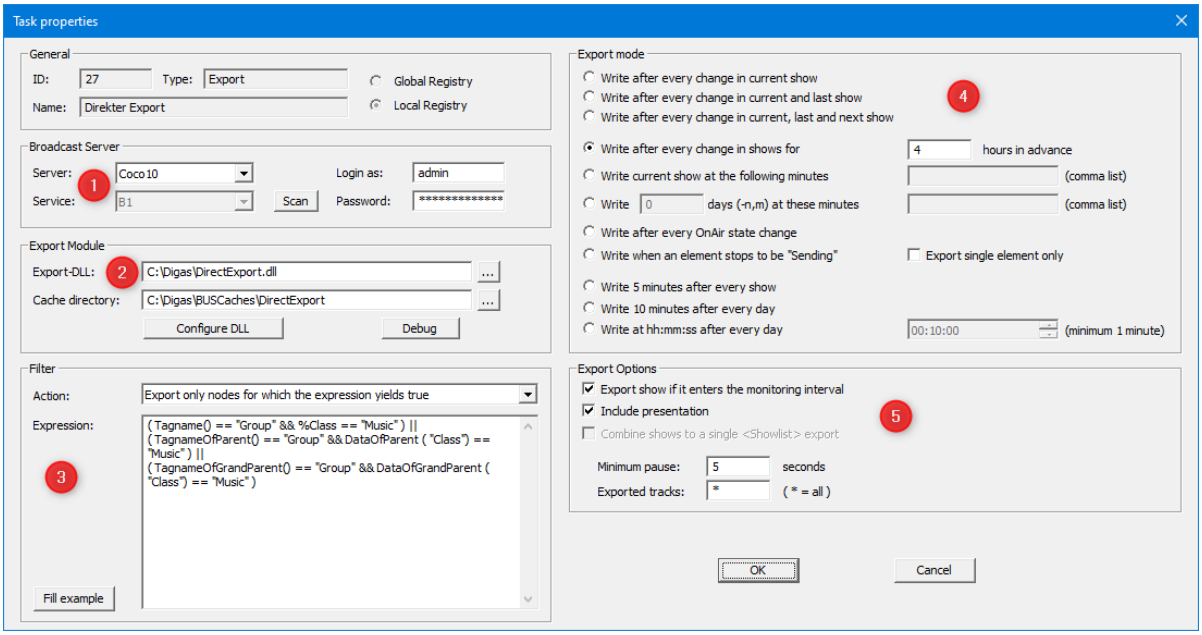
The first mentioned option is somehow stronger than the second one. It does not only protect some very specific metadata fields, but all fields and also the existence and the location in the hierarchy of an element. This might even be a too strong protection, because then something like shortening the mark-out of an already playing element would not be synchronized to the playing element on the destination side. We recommend to perform some tests first with only the second option checked. Anyhow, normally there should not be any changes to the past part of shows by a user and therefore, there should not be the need for using the first option.

Please note that running two TurboPlayers simultaneously is a challenging endeavor. It should be clear that it will only work for fully automated shows. Last minute changes to the rundown or even changes to already running elements should be avoided. Anyhow, you cannot expect a synchronous playout, meaning: sample exact. But a playout with only a second difference might be achievable.

A last hint: the method of using an Import task for duplicating rundowns from one BCS to another one works well for shows. But as the import does not handle jingles at all, it is not possible to use the same method for duplicating jingle groups. Duplicating jingles would need significant extensions of the import routines (which was not done yet.)

7.5 Export Task

The export task performs the reverse operation of the import task, but it is much easier because there is nothing like the merge process. The whole show with all its tracks, groups and elements is transferred to the export modules. Of course, some of the modules can be configured to extract only a specific part of the show. First, we present the settings dialog:



1) In the Broadcast-Server group, you select the server and service to export from.
2) The DLL to use is set with the first field of the Export-DLL group. The cache directory must be a directory, which is used for this purpose only. There must be no other files in it. BUS uses it to store lists of all the exported shows. With the “Configure DLL” button you can call the DLL specific settings dialog, the "Debug" button opens a dialog to configure an output directory for writing log files (with all exported XML data) about each export call of the DLL.
3) In the filter part you can configure an expression which filters the nodes before they are transferred to the export module. With the “Action” combo box you can deactivate the filter or you can select that all nodes for which the expressions yield true are either suppressed or are exported. The “Fill example” button writes a little example into the expression field.

   Normally, you will not use a filter and export all nodes, which are present in the data. But with the expression filter you can exclude certain nodes from being included in the exported data. The expressions you can enter are explained in the chapter 7.1.9 “Filter expressions” above. The example shown in the screenshot above would generate an export, which contains only groups with class “Music” but all stories and elements in such a music group independent of the class of the story/element. Other possibilities to filter the exported data are the field “Exported tracks” of this dialog or you can use the DirectExport.dll and execute a vbs script on each call, which could filter the data.

   Beginning with BUS version 6.0 you can also define your own macro functions in order to make the code more readable. If you do that, first define all functions you need and then define a last function with name "filter" (!), which must return either true (1) or false (0). Do not mix functions and declaring only a single expression.
4) In the group “Export mode” you can select the following operations modes:

   Write after every change in current show:

   This mode monitors the current show. That means: the server notifies BUS about every change. After a change in a show it will be (re-)exported. Use the minimum pause value to avoid a too high rate of exports !

   Write after every change in current and last show:

   Like before, but the previous show of the current show is included. BUS will call the export module once for each show. Therefore, each show will produce one file, typically.

   Write after every change in current, last and next show:

   Like before, but the next show after the current show is included, too. BUS will call the export module once for each show but you can set the option "Combine shows to a single <Showlist> export". If checked BUS will integrate the three shows in a single XML string with root node "<Showlist>". Be aware that the

export module must be able to handle this special tagname. Most can handle only a top level <Day> or <Show>.

Write after every change in shows for ... hours in advance:
Like before, but in addition to the current show all shows which start in the specified interval into the future will be monitored too and are exported when they are changed. This mode always exports individual shows.

Write current show at the following minutes:
The current show is written at the specified minutes of each hour – independent of a change in it.

Write ... days (-n,m) at these minutes:
The specified day interval of n days in the past, today and m days in the future are exported at the specified minutes of each hour.

Write after every OnAir state change:
This mode works like "Write after every change in current, last and next show". The differences are: the task becomes only active if there is an on-air related change, meaning: an element is going on-air or off-air. In any case all three monitored shows are exported at the same time in a single showlist. This mode is intended to be used together with the FlexXML.dll which picks out the currently playing elements, the next ones and the last which were playing and writes this information into a file. This can be used for a display of "what is just on-air" on a web page.

Write when an element stops to be "Sending"
Like before the "OnAir state change", but an export is only done when a formerly sending element is stopped. Hint: it does not matter, whether the new state of the element becomes "Sent", it is sufficient that the element has no longer the send state "Sending".

When the option “Export single element only” is checked, BUS calls the export function once for every element, which is becoming “Sent”. This is done with very special data. The data will contain only the element, which becomes “Sent”, embedded in a single show (the one, in which the original element is located). This requires that the receiving export module is able to handle the data correctly, otherwise each new export would overwrite the data from the previous one and at the end it might seem that only the last element was exported. BUS is able to catch up with some recent elements in the current and previous show, which became “Sent” while the task was not running. This allows to shut down the task for a short time – e.g. for changing settings or for updating BUS – but when the task is not running for a longer time, not all sent elements will be exported.

Write 5 minutes after every show:
In this mode the task will run only once: 5 minutes after a show has completed this show is exported.

Write 10 minutes after every day:
In this mode the task will run only once: 10 minutes after midnight the past day is exported.

Write at hh:mm:ss after every day:
This mode is for exporting a whole day at a specified time of the following day.

5) The Export options are:

- The checkbox "Export show if it enters the monitoring interval" can be used with all operation modes, which are triggered by changes. The option allows avoiding the problem that a future show, which appears in the monitored interval, is not exported because no one makes any changes in the show. If the option is active, the export task will export each show at least once – when it is covered by the monitoring interval for the first time.
- The checkbox “Include presentation” must be checked if you want to include the text of the RTF file for each entry in the generated XML data. This is a time-consuming operation because all files must be found, read and included in the XML string, so only use this option if you really need it.
- The option “Combine shows to a single <Showlist> export” is only available for the mode “Write after every change in current, last and next show”. Without this option, the task would export three individual shows. Sometimes it is desirable to combine the three shows into a single export. This will be done by integrating the three shows into an XML with root tagname <Showlist>. Please be aware that the export module must be able to handle such a root tagname. (Not all export modules can do this.)
- As it is possible that hundreds of changes per second occur, it is not practical to export a show so many times. Therefore, the “Minimum pause” parameter exists. You should always enter a few seconds. BUS will wait at least this time before it exports a show anew.

- "Exported tracks" should be self-explanatory. Be aware that this affects only the tracks, which are included in the call to the export module. Export modules typically filter their input for a single track or even for some elements in a specific track and write only the filtered information.

Note that you cannot combine all export modes with all export modules. Some of them can handle multiple data sets, but others will run only with specific data. For example, the FlexXML.dll has to write the current, next and previous element with every change, so it needs the current show and possibly the previous or next show. But it is not capable of writing a whole day.

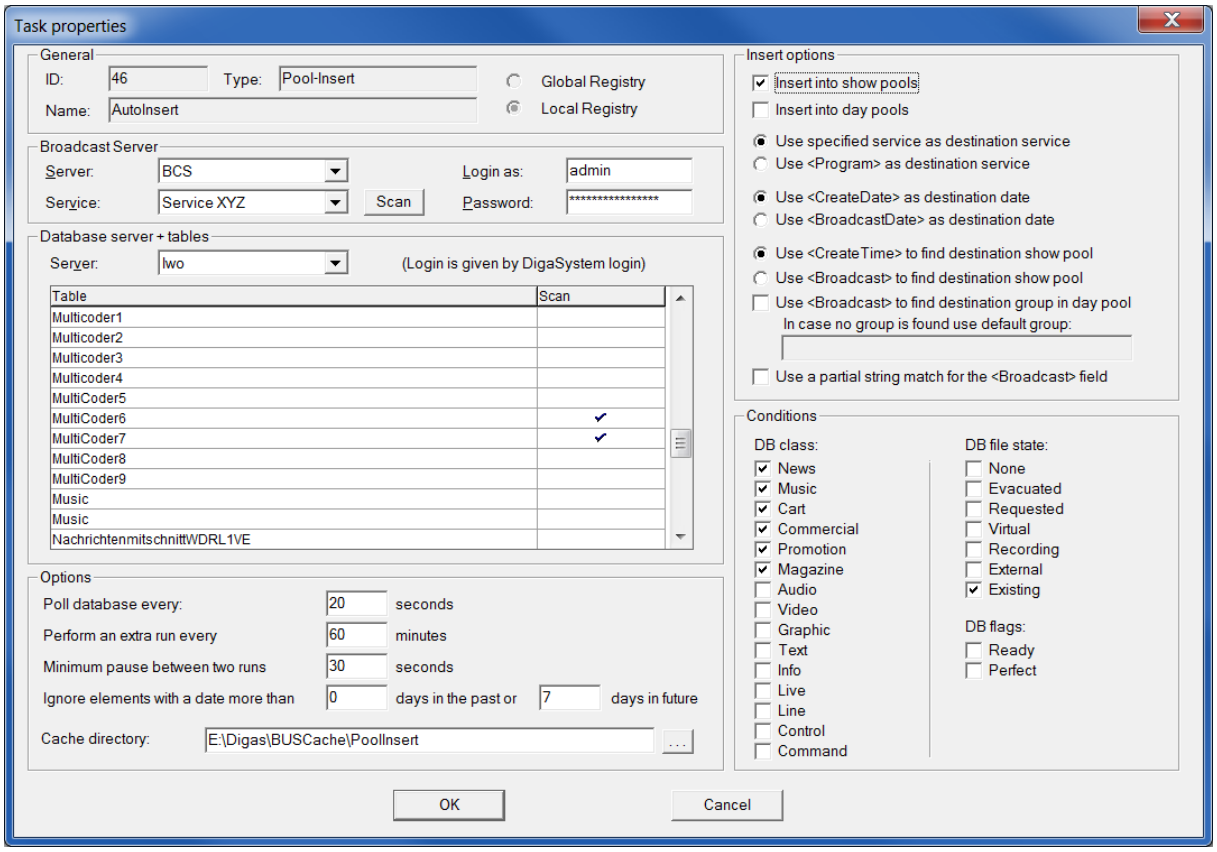
You should be aware of the fact that some of the exports may take place too late if you change the list of defined shows while the export task is already running. For example, if you want to export a whole day but there is no show for this day defined at all BUS will look for new shows not before the next day. If you define new shows, they are exported one day too late. Something similar can occur for the other export modes but the intervals are shorter: in Actual mode it is 5 minutes, in Future mode it is 1 hour and in After-show mode it is 15 minutes.

7.6 The Insert Tasks

In contrast to the import task which imports and merges a whole rundown into the BCS rundown, the insert tasks insert only individual elements from a DigaSystem database into the BroadcastSystem. There are two different insert tasks: the pool-insert and the show-insert. Both scan tables within a DigaSystem database to detect new entries. The first one inserts new entries into day or show pools, the second one can fill new entries to a show directly.

7.6.1 Pool-Insert task

Here is the settings dialog:



On the left side you can select the BroadcastServer and the service into which new entries are inserted and the database tables which are scanned. It is not possible to scan multiple databases, therefore you have to use multiple tasks.

Below you see the **database options**:

- Poll database every: Here you specify the poll interval for the database, a value of some seconds is appropriate.

- Perform an extra run every: In an extra run the tasks queries all database entries, not only the changed ones. This might help to catch "forgotten" entries in case of errors.
- Minimum pause between two runs: Here you should set a pause time of some seconds to avoid a too big load for the database and/or the BCS.
- Ignore elements with a date…: Here you can exclude entries which would be inserted into a past day or a day too much in the future. Remark: here the day is meant, into which the entry will be inserted. This day is either given from the creation date (here the date limitation is not very useful) or the intended broadcast date (which can be easily out of range).
- Cache directory: As usual this must be a directory used for this task only. The task stores information about the last DB entry which was treated per table in an XML file.

The **insert options** are:

- Insert into show pools: The task will insert new DB entries into show pools.
- Insert into day pools: The task will insert new DB entries into day pools.
- Use specified service…: The task will insert entries into the service which is specified in the box "BroadcastServer" on the left side. The alternative is to use the database field <Program> to identify the destination service. The service specified in the DB field must be hosted within the connected BCS, otherwise the operation cannot be performed.
- Use <CreateDate>…: The task will insert entries into the day which is given by the DB field <CreateDate>. The alternative is to use the DB field <BroadcastDate> to find the destination day.
- Use <CreateTime>…: The task will insert entries into the show (pool) which is identified by the DB field <CreateTime>. The alternative is to use the DB field <Broadcast> to find the destination show (pool).
- Use <Broadcast>…: Normally the entries will be inserted as last element into a day pool. If there are groups in the day pool - one for each show of the day - the task can also insert the entries into the corresponding groups. The group title should be identical to the content of the DB field <Broadcast>.
- In case no group…: If no destination group is found within the day pool a default group with the given title can be used.
- Use a partial string match…: If checked the task will not compare the whole group titles within the day pool with the field <Broadcast>, but instead only a partial match will be sufficient.

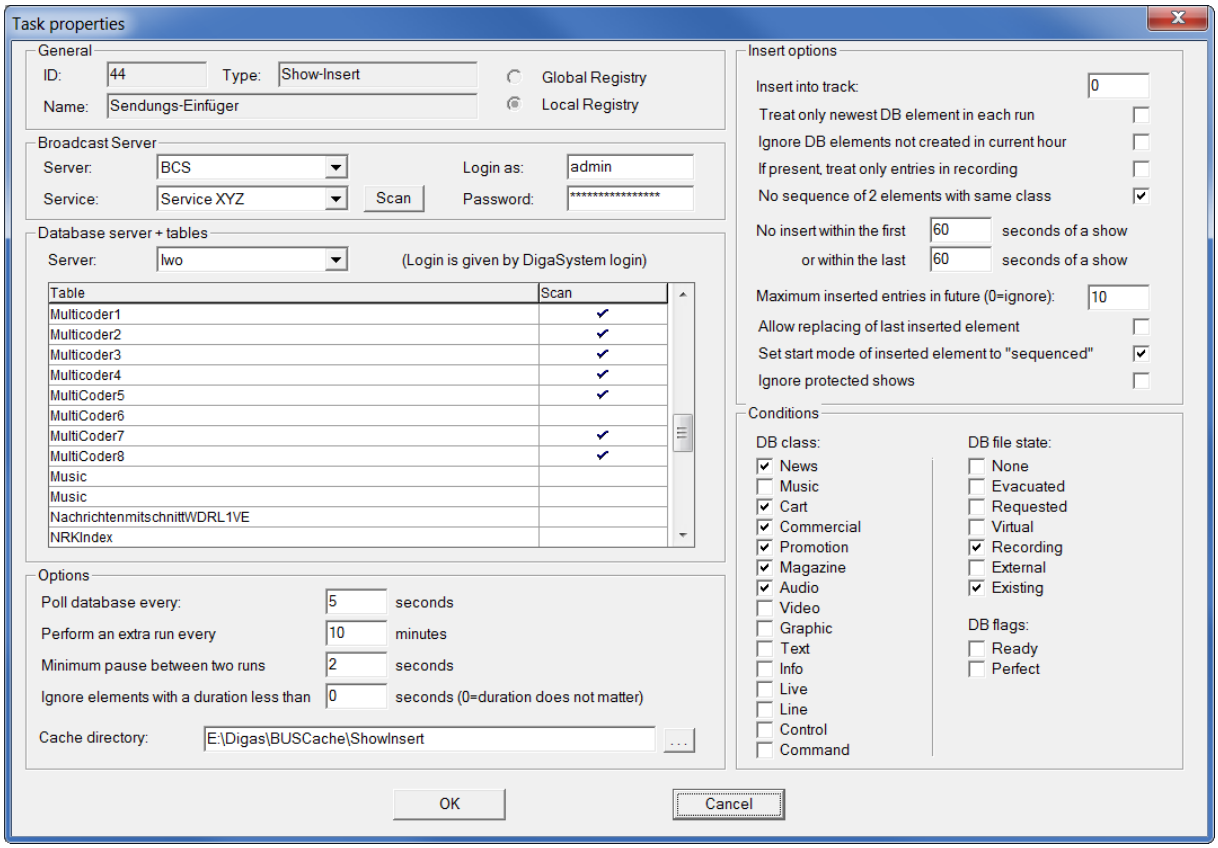
In the **conditions** the database entries can be preselected / filtered:

- DB class: Only DB entries with a checked class are treated.
- DB file state: Only DB entries with a checked file state are treated.
- DB flags: Only DB entries having a checked flag set are treated.

7.6.2 Show-Insert task

This task inserts new database entries directly to a show. The task loads the current plus two future shows. If a new entry is detected it computes a place to insert the entry within these three shows. The first possible place is the <next> position in the rundown: the place after the last playing/played element. If the previous element is running and the remain time is less than 10 seconds an appropriate place in the future is searched. The same is true if the place is already occupied by an automatically inserted element, or if one of the rules you can setup with the insert options would be violated. The task advances in the rundown until it finds a possible place for the new DB entry. If a place can be found, the entry is inserted. If no place can be found, the entry is ignored and will not be inserted in later runs. This is very important, therefore once again: If no appropriate place can be found for a new DB entry, the entry will never get inserted ! There is only one exception: if the entry is in state "recording" and has not yet reached the minimum length the task waits and continues to survey this element.

Here is the settings dialog:



The left side is nearly identical to the previous PoolInsert task. There is only one difference: instead of a limitation of the destination date you can specify a minimum duration of the database entries. If an entry is in state recording and has a duration, which is too small, the task waits until the entry reaches the minimum duration. The DB conditions are also identical to the PoolInsert task. Here are the different options:

Insert options:

- Insert into track: This is the destination track number.
- Treat only newest DB element in each run: If checked the task won't treat any DB entry except the newest one it finds during a DB scan.
- Ignore DB elements not created in current hour: If checked database elements having a creation date with an hour different from the current hour (when the task is running) are ignored.
- If present, treat only entries in recording: If checked the task won't treat any non-recording DB entry if at least a single entry in state "recording" is found during a DB scan.
- No sequence of 2 elements with same class: If checked the task will not insert an entry to a place in the rundown if the result would be two elements with the same class in sequence.
- No insert within the first … or within the last seconds of a show: Here you can set up an interval around each show border in which no insertions are performed. This rule means: at the moment when the task wants to insert an element, the estimated start and stop time for the element must both be outside of these intervals, otherwise the element is either ignored or inserted at a later position (depending on the other settings). If you specify 0 there is no exclude interval.
- Maximum inserted entries in future: If you set this value to 0 the number of inserted entries is unlimited. (Attention: the task loads only the current plus 2 future shows.) If you enter a positive number, the task will not insert DB entries if all "available" places in the rundown are already filled.
- Allow replacing of last inserted element: This checkbox is only active if a maximum is given with the previous field. Normally, if the limit of future elements is reached, new DB entries are no longer inserted and ignored. If this box is checked, new DB entries can replace the last element which was already inserted.
- Set start mode of inserted element to "sequenced": This might be useful to assure that the element starts automatically.
- "Ignore protected shows" can be set to avoid that the task imports into shows which have been declared "approved & protected".

7.7 Auto-Replace Task

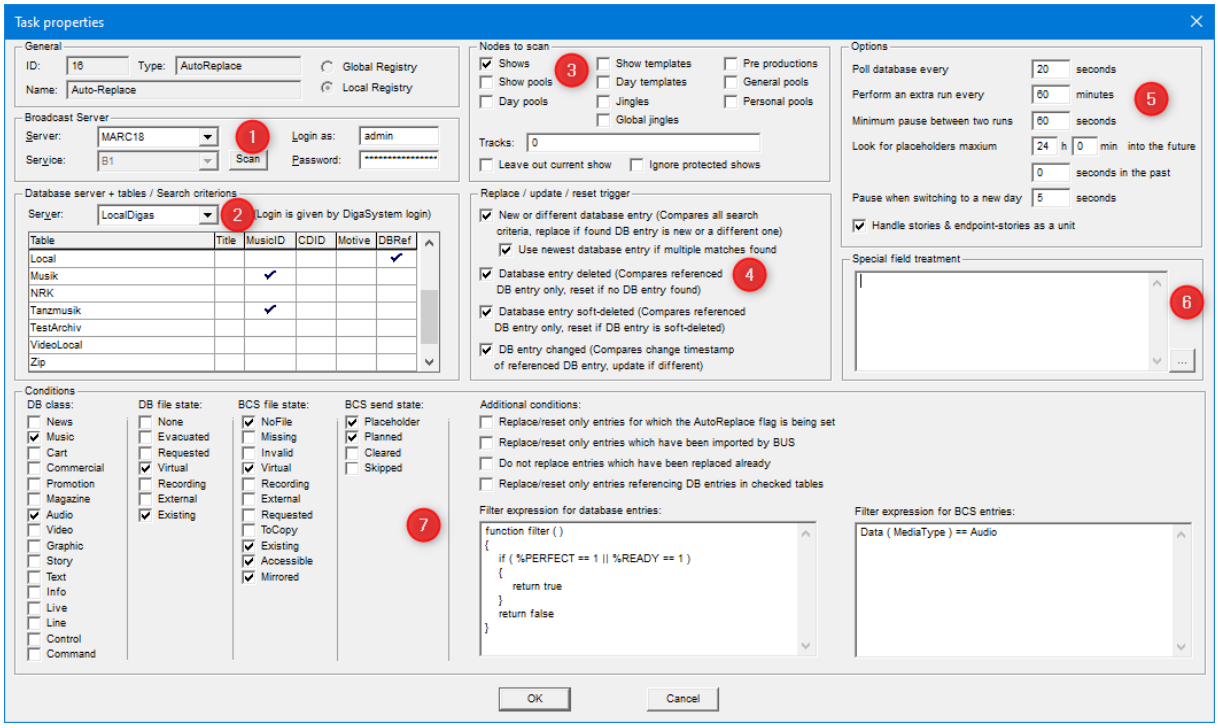
7.7.1 Settings

The BroadcastSystem and the DigaSystem database are two separate systems with independent data storage. Therefore, there is the advantage that you can edit elements within the BroadcastSystem without changing the original database entry. But sometimes it might be desirable that the elements in various places in the BroadcastSystem are only some kinds of “view” of a single database entry. The separation of both systems does not allow this, but the AutoReplace task can help to achieve something similar.

The task watches a certain interval in the schedule (starting from the current time) and some tables of the DigaSystem database. Whenever a change occurs within the database, the task tries to find corresponding elements in both systems and will replace or reset the BCS elements possibly. As you will see in the next picture, there is a whole bunch of options you can use to specify which elements should be replaced.

But first, here are two typical uses of the task: You can use the task together with some automatic recording tool, like the MultiCoder. It enables you to put an element/placeholder in the schedule list, which is not yet recorded and for which even no virtual entry does exist. When the coder creates a new database entry, the AutoReplace task will replace the placeholder with the database entry. The task is also helpful for the following problem: your system imports rundowns from a music rotation. The FileCheck-Task finds out that some entries are not available online. Now someone grabs the missing titles from CD and enters them into the DigaSystem database. The AutoReplace task can find these new entries and can insert them automatically for the elements with missing file.

Now here is the settings dialog of the task:



1) The parts “**General**” and “**Broadcast Server**” should be clear.

2) Below there is the part with the "**Database server + tables / Search criterions**". The task can scan a single database server only; therefore, you have only a combo box to select the server name. All known DigaSystem tables of the selected server are displayed in the table below. The check you set specifies which database fields are compared with the corresponding BCS field to find corresponding elements. The last column “DBRef” is somewhat special, it is used only in conjunction with the fourth trigger (see below) and there it means: compare only the database reference number to find corresponding elements.

3) The part “**Nodes to scan**” allows to select node types within BroadcastSystem, which are being watched. For shows you can specify the tracks to watch. The track list must not be empty, every track you want to be scanned must be listed in a comma separated list.

4) The part “**Trigger**” tells BUS when to perform a replace or reset operation. Four triggers are available:

– New or different database entry. BUS will use the title or one of the ID fields to find a database entry. If the reference number of this entry differs from the DBRef of the BCS element, BUS knows that the DB entry is new or has been exchanged and will replace the BCS element.

  If the option "Use newest database entry if multiple matches found" is used for this trigger, then if multiple matching database entries are found, the BUS task will use the newest one (CreateDate,CreateTime). Otherwise, multiple matches will be considered to be an error.

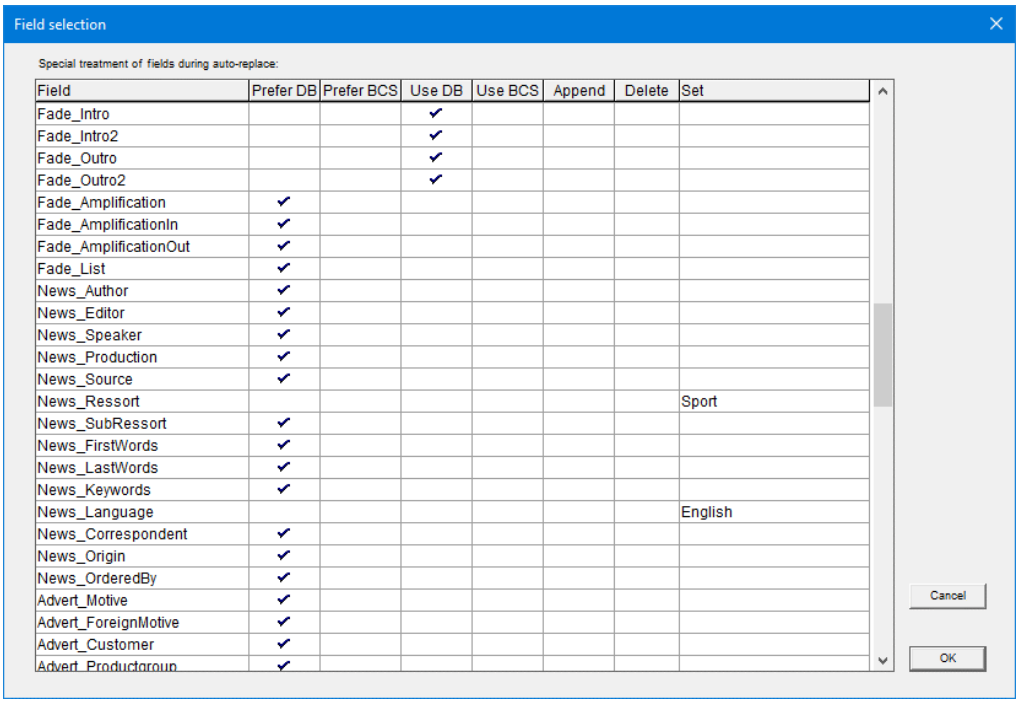
– Database entry deleted. A database entry copied to the BC system will store the original reference number in the field DBRef. The task tries to locate the referenced database entry, but it looks only in the referenced table ! If it does not find such an entry, it resets the BCS element (makes a placeholder out of it).

– Database entry soft-deleted. The same as before, but it resets the BCS element if the original DB entry is only soft-deleted.

– DB entry changed. Again, BUS tries to locate the referenced database entry. If this entry has changed in any field, an update of the BCS element is performed. BUS uses the BCS field “Create_ChangeTime” and compares it against the DB fields “CHANGEDATE” and “CHANGETIME”. If the timestamp differs, the DB entry is considered as being modified.

5) In the “**Options**” you can set the database poll interval. Normally, the task waits for a change in the monitored tables before performing a compare-and-replace run. But you can also make it perform an extra run without any database changes (second option). In such a full run the task will scan through all elements in the BCS tree within the monitored interval and compare it to the corresponding database element. So, if a user makes a change in BCS system but not in the database, only a full run will detect the situation that an element needs to be exchanged.
With the third option you can limit the load generated by this task. Especially if there are a lot of database changes it is useful to slow down this task a bit. As it is not possible to scan an infinite time interval, you must limit the time range this task will look into the future. This is done with the option “Look for placeholders maximum … hours ... minutes into the future”. Do not make this value bigger than necessary, because the task has to scan all shows in this interval for each database change.
Replacing elements in the past is not very useful normally, but the option "... second in the past" can extend the lookup interval by a few seconds to avoid problems with an unprecise time synchronization or changes in the last second.
With the pause option you can define a pause when switching to a new day. The idea is to avoid a too high load for BCS when the task handles a lot of days in a sequence. Especially if the task operates far in the future, BCS has to load all these days into memory first. Therefore, the task has an additional feature to reduce the BCS load: it will tell BCS that a handled day can be removed from memory again when the day is more than 1 week in the future. BCS will then remove the day from memory – but only if no other client has active locks or notifications for this day.
The last option defines the handling of stories and endpoint-stories. If you check this option, every change of a story/endpoint-story entry in the database will trigger a synchronization of the list of children. If the user inserts a new story member in the database or removes an existing member, this will set the change timestamp of the story entry and with this option being active, all children on BCS side will be synchronized (=created, moved & deleted). Without this option, story children are only compared and exchanged as individual elements. Activating this option has another implication: then a Story or EndpointStory can replace a single element on BCS side and vice-versa. This allows to create a single-element placeholder in BCS and let it be replaced by a Story/EndpointStory later.

6) In the part "**Special field treatment"** you can define for each possible field how it is treated in the replace operation. Though you could do it, you should not edit the list displayed in the big edit box directly, instead click the "…" button and a dialog appears, which allows to set the treatment for each field in a more comfortable way:



Field	Prefer DB	Prefer BCS	Use DB	Use BCS	Append	Delete	Set
Fade_Intro			✔				
Fade_Intro2			✔				
Fade_Outro			✔				
Fade_Outro2			✔				
Fade_Amplification	✔						
Fade_AmplificationIn	✔						
Fade_AmplificationOut	✔						
Fade_List	✔						
News_Author	✔						
News_Editor	✔						
News_Speaker	✔						
News_Production	✔						
News_Source	✔						
News_Ressort							Sport
News_SubRessort	✔						
News_FirstWords	✔						
News_LastWords	✔						
News_Keywords	✔						
News_Language							English
News_Correspondent	✔						
News_Origin	✔						
News_OrderedBy	✔						
Advert_Motive	✔						
Advert_ForeignMotive	✔						
Advert_Customer	✔						
Advert_Productgroup	✔						

In the column "Field" you see some predefined field names. You can extend this list by adding a new field to the end of the list. The next columns specify the treatment. The standard treatment is to prefer the database field, which means if the field is present and filled in the database it is used; otherwise the content from the BCS field is kept. The same is true for "Prefer BCS" but the other way round. "Use DB" and "Use BCS" means: always use this field from DB/BCS, never fall back to the other system, even if the field is empty. "Append" will append the content of the BCS and the DB field with a separating space. "Delete" will remove the field from the resulting BCS element. For "Set" you must specify the new content of the field in the resulting element. Attention: this content will be written to the result unchecked, so be careful ! You should be aware that the two operations "Delete" and "Set" are performed as last modifications before the result is being stored, so it is easy to overwrite or destroy any essential information. Therefore, we do not recommend changing any of the fields, needed by the system. This is true for the following fields: file references, state information like FileState, most time information including StartMode/Type, fade values and the fields "Source" and "ImportedBy".

7) The "**Conditions**" part allows you to specify which conditions a database entry and the corresponding BCS element must fulfill, before a replace or reset operation will be performed. Most conditions like the class or the file state should be clear. If in one of the filter columns "DB class", "DB file state", "BCS file state" and "BCS send state" not a single item is checked, the corresponding filter is not applied at all. **Important:** if you leave the conditions for "BCS send state" empty, all BCS elements will be replaced. This includes sending and sent elements and this is probably not what you want. Therefore: always check the desired BCS send states conditions!
A hint to second and third option of the "Additional conditions": the check whether a BCS element has been imported by BUS or been replaced by AutoReplace is done with the field "ImportedBy". In the first case the field must contain something like "BUS\DirectImport" (only the first part is checked against "BUS"), in the second case it must **not** contain "BUS\AutoReplace". The condition "Replace/reset only entries referencing DB entries in checked tables" means: The treatment performed by the triggers described below (with the exception of the "New " trigger), will only take place for BCS elements with a DB reference to one of the checked tables (at least one match condition must be checked for the table).
Since version 6.1 you have the possibility to specify search expressions for the database entry and for the BCS node. The syntax for the expressions follows chapter 7.1.10 "Macros and filter expressions". If you specify an expression, it must yield true to make the entry/node be handled. You can also use functions as alternative. In that case there must be one function called "filter()", which must be the outermost function and which must return either true or false. Like before, an entry/node is only handled if true is returned.
Within the filter for database entries, you must use field names as they appear in a DBE section string from the database! Examples: %TITLE or %READY. In section strings there is no field hierarchy. Fields within a hierarchy are flattened by adding the field names separated with a slash. Examples: "MEDIUM/FILE#1/FILEREF" or "STATION/CUSTOMFIELD". In that case you cannot use the syntax with the preceding %, because the parser will interpret the slash as a division. Use the syntax: Data ("<field name>").

In the screenshot above you can see an example for a filter expression in the BCS filter field and the usage of the filter()-function for the DB filter.

One last hint concerning the task settings: you should not activate the “DebugLog” for this task if you do not need this extended information. Of course, the debug output can be extremely useful if you try to locate a problem, e.g. if the task does not replace an entry it should replace. But the problem is that the task will generate a much higher database load if DebugLog is active. Reason: Normally the task recognizes very early if a BCS element does not need to be checked, e.g. because it has the wrong send state. If DebugLog is not active the task will not look into the database but instead it will continue with the next element immediately. But if DebugLog is active, it will read the database information nevertheless to gather the information needed for the debug output.

7.7.2 Variables and Wildcards

In case you use the title as a search criterion the task does not perform an exact string comparison. The task allows you to use variables and wildcards in the title of an element. Possible wildcards are ‘*’ and ‘?’ like for the DOS shell. Each variable must be limited by two %-characters. Each variable is replaced at the moment when the task runs and scans the schedule lists. The following variables are defined:

- DATE: The date. You can specify an offset in days like “+1” or “-7” and a format in brackets like (DD.MM.YYYY).
- TIME: The time. You can specify an offset in milliseconds (not so useful) and a format in brackets like (12) or (24).
- YEAR: The year. An offset and the two formats (YY) or (Y) are possible.
- MONTH: The month. An offset and the two formats (MM) or (M) are possible.
- DAY: The day. An offset and the two formats (DD) or (D) are possible.
- HOUR: The hour. An offset and the two formats (HH) or (H) are possible.
- MINUTE: The hour. An offset and the two formats (MM) or (M) are possible.
- SECOND: The second. An offset and the two formats (SS) or (S) are possible.
- SERVICE: The service (program) name.
- COMPUTERNAME: The name of the computer BUS is running on.
- SHOWDATE: The start date of the show which is currently being scanned, if a day pool is scanned: the date of the day pool. You can use offsets and a format like for <DATE>
- SHOWTIME: The start time of the show which is currently being scanned. Not defined for day pools. You can use an offset and a format like for <TIME>
- SHOWNAME: The title of the show currently being scanned. For day pools the predefined name “day pool”.

A complete example “%SERVICE%-Recording from %DATE-7(MM/DD/YY)%”

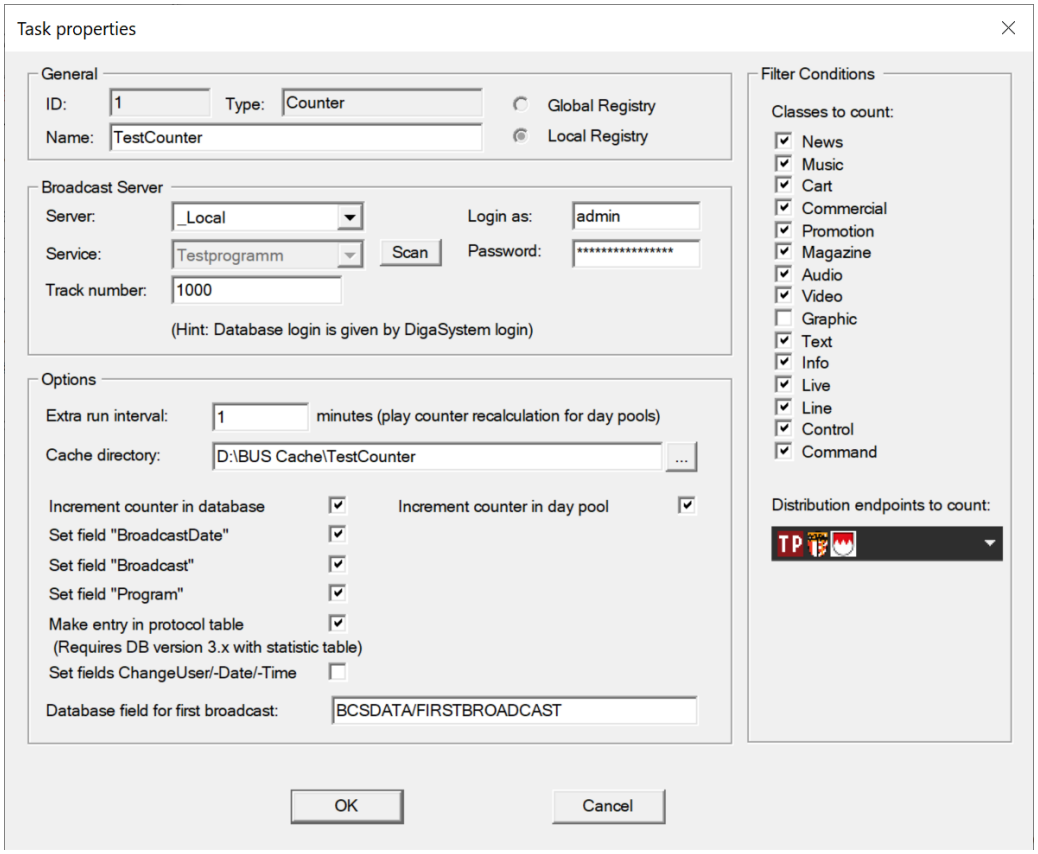
Or if you have an hourly recording: “Recording from %DATE(HH.MM.YYYY)% %HOUR-1(HH)%*”

If you have problems with the title matching, enable the debug log level. Now this task will list every entry it compares (also with replaced variables) and perhaps you can see the problem.

7.8 Counter Task

The counter task can be used to increment the “Broadcastings” column of a DigaSystem table whenever the corresponding entry is sent. Internally the task works like this: It does not use server notifications to survey the current show. Instead it stores the start time of the last element which was treated. Regularly (in an adjustable interval) it tries to find elements, which have been sent in the time between the last element and the current time. Whenever such an element is found, it evaluates its database-ref back-link. If it has access to the element, it increments the counter in the column “Broadcastings”. Afterwards it stores the time of the treated element in a file. This behaviour has been chosen to avoid any information loss if the task is not running for some time and to avoid incrementing the counter of an element multiple times. If a database connection is disturbed, there is an additional function: each element, which cannot be accessed because a connection is down is stored in a file in the cache directory. Later this directory is scanned and the files are deleted when the task succeeds to increment the corresponding counter.

Below you have an example of the settings dialog. The fields should be clear: you can define the server and service and the track number which is surveyed, typically you should use the main protocol track (1000). Do not make the poll interval too small to avoid stressing the BCS, something like a minute or a half should be o.k. The cache directory must be a free directory used only for the files of this task.



One installation hint: If the task starts for the first time, the file “LastElementTimestamp.txt” in the cache directory does not exist and the task starts to look for elements on the current day at midnight. If you want it to scan old entries, which have been sent long ago, you have to edit the text file manually. It simply contains a timestamp of the last treated entry and you can insert any time in the past where you want the task to start looking for sent entries.

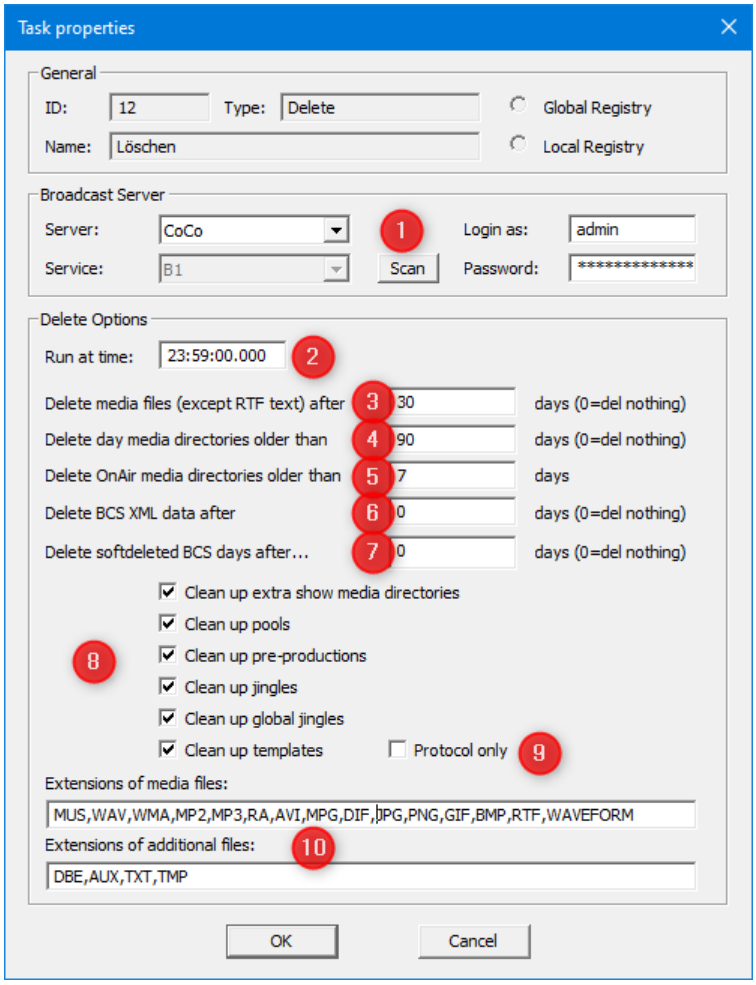
There are seven checkboxes in the option group. You can check them if you want the corresponding database field to be set whenever the broadcast counter is incremented. You can set the field “Program” (=service), the field “Broadcast” (=name of the show), the field “BroadcastDate” and the fields "ChangeUser", "ChangeDate" and "ChangeTime". Setting these fields is only sensible if you do not use them for planning purposes but for evaluation instead.
In addition, it is possible to make an entry in the protocol table in the database and to increment the use counter of the same element in the day pool (therefore the element must be inserted from the database via day pool. The counter shows, how many times the element was broadcasted per day).

Optionally, you can specify a database field, where the timestamp, in format “YYYY-MM-DD hh:mm:ss”, of the first broadcast of the element is stored. Usually this would be a custom field, indicated by a “/” or “.” In the field name. If the “Increment counter in database” option is not set, the first broadcast option is unavailable, because without an incrementing counter, the task cannot know if a broadcast has been the first one.

There is a list of checkboxes corresponding to the filter for classes to be included for counting. There is also a checked combo box to select the distribution endpoints to be included for the counter task. When nothing is selected, the counter task continues as usual. If no information is present in element metadata, turboplayer endpoint is assumed.

7.9 Delete Task

Before reading this chapter, you should check out the chapter “File Management” for a general overview of the media file handling.
This task deletes media files and media directories, including waveform and prelisten files that may have been generated by the file check task. We must distinguish between day media directories and others directories (Jingles, Pools, …). Day media directories with all their files are always kept for an adjustable period of time (this includes dead files). After this period the directories are deleted completely. The other media directories are treated differently: Once a day BUS compares the content of these directories with all files, which are listed in the corresponding node of the server tree. Files, which are not listed in the tree are regarded as unnecessary and are deleted. With this preparation, you can look at the possible settings of this task:



1) The fields in “Broadcast Server” should be clear. If you want to delete multiple services, you have to start multiple tasks.

2) In “Run at time” you can define the time of the day when the task starts deleting. This is valid for both, day and other media directories.

   In the next three fields, you can adjust how long day media directories are kept.

3) The first field says how many days all media files in non-OnAir directories are kept. After this period, the task deletes all media files, except the RTF files.

4) The second field specifies when the non-OnAir directories are removed completely. If this value is shorter than the first one, all files are deleted together with the directory itself.

5) The third value specifies when the OnAir media directories are removed. Typically, it is not necessary to keep these directories longer than a day. If you enter “0” in any of the tree fields, the corresponding day deletion is not performed at all. If you did not run this task for a long time (more than a few days) it might be necessary to delete some of the day directories manually because BUS will not walk back the directory list for endless.

6) It is also possible to delete the XML data within BCS after some days. Normally this should be not necessary because the describing metadata is by far smaller than the storage needed for media files. In case you enter a value here BUS will only trigger a deletion of old days, BCS itself will delete old files.

7) You can also remove soft-delete days from BCS XML data. Soft-deleting can be done from DigAIRange when deleting whole days and checking the option "Move days into the list of deleted days".

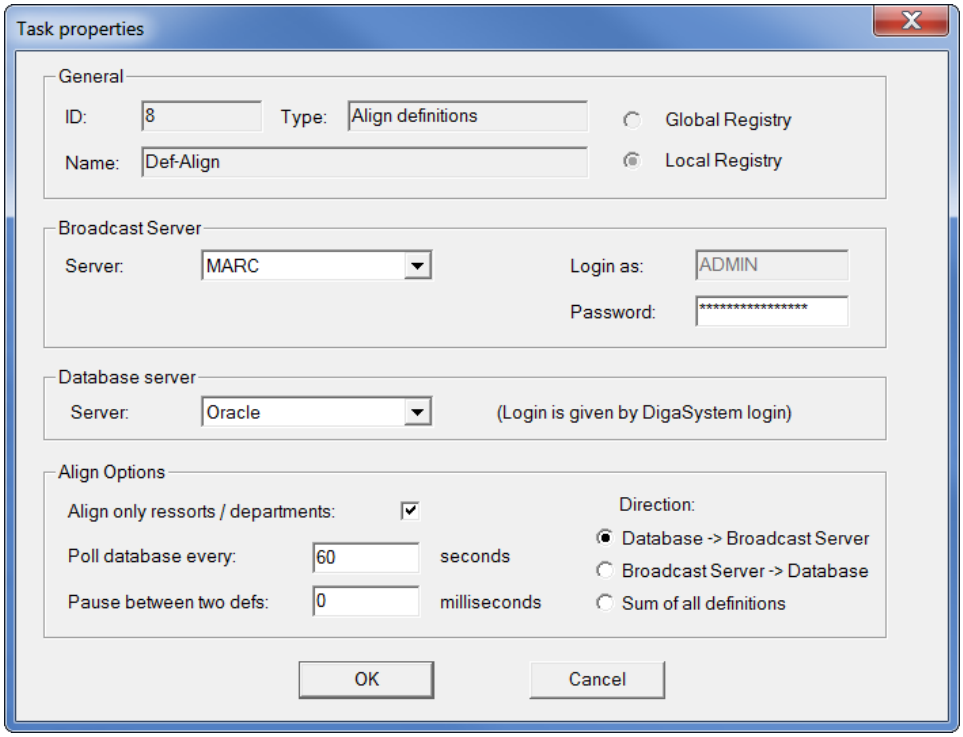
8) The first checkbox “Cleanup extra show media directories” should only be checked if you use these extra show media directories in your system. (These are the media directories, which you can assign to each show individually.) If activated, this creates a big load for BCS, because the task must read a lot of past days from BCS in order to find out the show media directories in the show metadata.
The five other checkboxes at the left side can be checked if you want the corresponding tree node (aka directory) to be treated. The deletion for these nodes works differently as for the normal show/day nodes. BUS builds two lists, one of the items found in the BCS tree node and one of the files found in the corresponding directory. If a file is found which is not referenced by an BCS item, it will be deleted.

9) Important: If you run the task for the first time you should always check the “Protocol only” box. Then BUS will not delete anything but it will print only messages in the log. You should check these messages first before running the task in “live” mode.

10) The last two fields are present for security reasons. Normally there are only DigaSystem media files in media directories and BUS (as part of DigaSystem) knows when to delete them. But let’s assume there is a file which is not known to DigaSystem in one of the non-day media directories. The delete task sees this file and as it cannot find a reference to this file in the server tree it will delete the file. To avoid any problem which could arise from this behavior, you can define a list of file extensions in the first field. Any file with an extension, which is not listed in the first field, is ignored by BUS. The second problem is that even some DigaSystem files are not listed in the server tree, e.g. an AUX file. Such a file would have the same destiny as a non DigaSystem file: it would be deleted. But there is a difference between both files: the AUX file must be deleted when the corresponding audio file is deleted. Therefore, BUS needs a separated definition of these additional files. We recommend to use the lists as they are printed in the example above as first try.

7.10 Align Task

The purpose of the align task is aligning definitions and ressorts (departments) between a Digas Broadcast System and a DigaSystem database. At the moment the usefulness is limited, because definitions and ressorts are not used within the BCS. Every field (including defs and ressorts) are stored in plain text in XML strings. But that my change in the future.
There are only a few settings for the align task. You can select both servers, the broadcast and the database server. The most important parameter is the direction. You can copy the data only in one direction which means you cannot change defs or ressorts in the destination system. Or you can let BUS generate the sum of all defs/ressorts in both systems. This has the disadvantage that you can no longer delete any def/ressort. In all cases the lists are made equal in both systems.

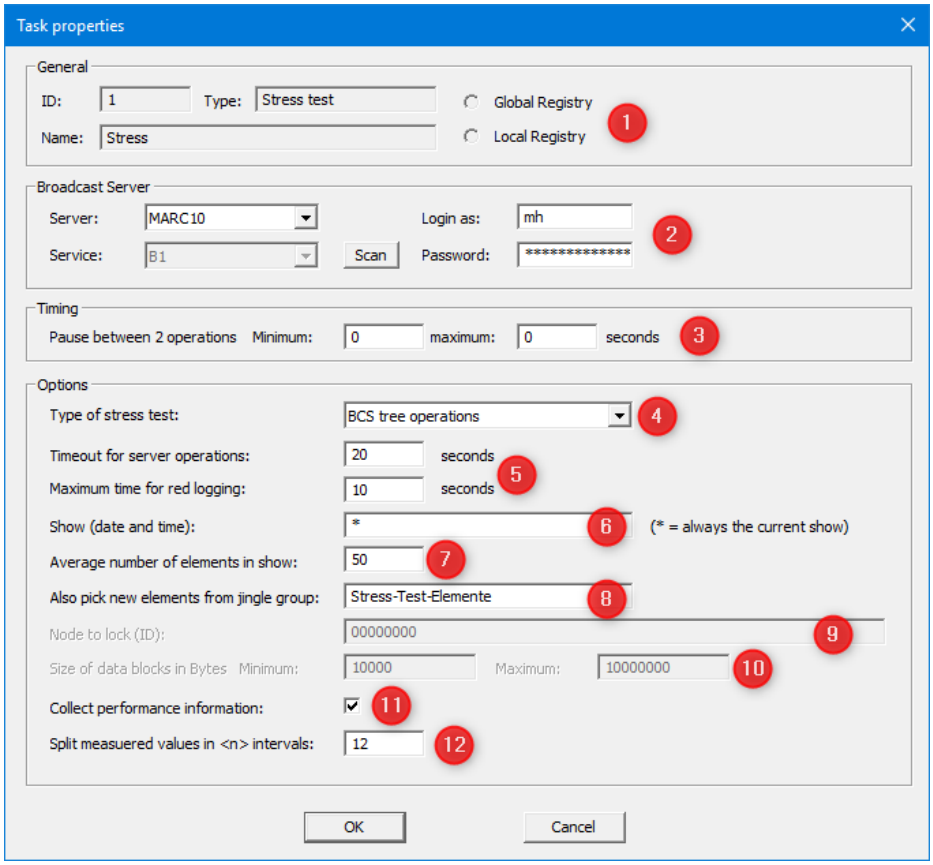


The rest of the align options is simple: You can limit the task to treating only the ressorts (departments), you can specify how often the database server is queried for changes and you can slow down the task with the pause value.

7.11 Stress Test Task

The stress test is intended for testing only. You can use it to generate a workload for a BCS and to measure the time needed by BCS to perform some operations. Be careful if you use this task with a productive BCS. You can easily create a very high load, which prevents a proper operation for regular clients.

This is how the settings dialog of the stress test taks looks like:



1) The usual general task settings.
2) The server and service to be used together with login credentials.
3) The pause in seconds between two operations. A real value is picked randomly within the specified range. Minimum and maximum time can both be equal and can both be zero.
4) Here you pick the type of operations performed by the task. Depending on the selected type, other settings become available or greyed out. Five types are available:
   - BCS tree operations: BUS will perform random operations on the specified show, including generating, changing and deleting elements. Therefore pay attention: The show will be destroyed by the task ! The main purpose is to start this task multiple times on multiple computers to evaluate the behaviour of the Broadcast Server and to estimate the number of possible clients for the server.
   - BCS read operations: BUS will use the same show like for “tree operations”, but BUS will not perform any write operations. It will only read the specified show. The time until the show data is received and can later be printed as statistics. This type of operation can be used to simulate regular clients, which most of the time only monitor a show and do sometimes a read operation but never perform a write operation.
   - Connect/disconnect: BUS will only connect to the specified BCS, login with the specified credentials and then disconnect again.

- Measure network throughput: BUS will generate random data with the specified size and send it to BCS. BCS will neither perform any operation on the data nor do anything with the data tree. It simply sends the identical data back to BUS. The time needed is measured and later a statistic is logged. This can be used to measure the throughput of the network and the TCP/IP stacks on both computers. This type of operation requires a BCS version 5.7.363.0 or newer.
- Measure locking times: BUS will call a special function in BCS, which only tries to get the internal lock for the data tree and then directly releases the lock again. The time needed to get the lock is measured and is printed in a statistic when the task ends. This can be used to measure the load by other clients for the BCS, because each regular client will request the internal data lock with every read and write operation. This type of operation requires a BCS version 5.7.363.0 or newer.
- Simulate out-of-memory: If activated, the memory will be eaten up within less than ~ 2 minutes. This is intended to test this special situation and check, whether BUS is restarting itself. Hint: if BUS is running as a Windows service, it is up to the Windows Service Manager to restart BUS. BUS itself or any other program cannot do it.

5) Here you can specify the timeout for performing BCS operations. You should use quite high values if you really want to generate a high load for the BCS.
6) Here you can specify the show to be used for BCS tree operations. Use the format YYYY-MM-DD HH:MM:SS. If you write a star “*”, BUS will always use the current show and will advance automatically when time advances.
7) This is the average number of elements in the specified show for BCS tree operations. BUS will try to keep the number of elements near to this value, but nevertheless there will be some fluctuation.
8) BUS inserts new elements without media files, but if you want to have “real elements” be filled into the show, you can specify a jingle group (the name) from which BUS will then pick elements.
9) This setting is only for a future extension. It allows to specify a node to be locked in type “Measure locking times”. At the moment, BCS only has a single lock object for the whole data tree. Therefore, this setting is not evaluated. Nevertheless, the root ID 00000000 corresponds to the lock BCS applies at the moment.
10) For type “Measure network throughput” you can specify the lower and upper limit of the size of the random data being generated. Both values can be identical.
11) Check this box if you want to perform a performance measurement. For the two types “Measure network throughput” and “Measure locking times” this option is always active.
12) With this setting, you can influence the statistics being logged when the task is stopped. Internally, the task stores the time of each individual operation, but in order to no print thousands of values, the values can be classified into <n> intervals between 0 and the maximum measured value. <n> is given here.

7.12 Day Structure Import

The Day Structure Import can be used to import the day structure (=list of defined shows for a day) from a file, merge this information with day templates in the BCS (BroadcastServer) and put the result into a day on the BCS.

The task makes use of the already existing DirectImport.dll. With the help of the DirectImport.dll it watches a configurable directory for files in BCS-XML format which contain a full day and have a filename "YYYYMMDD.XML".

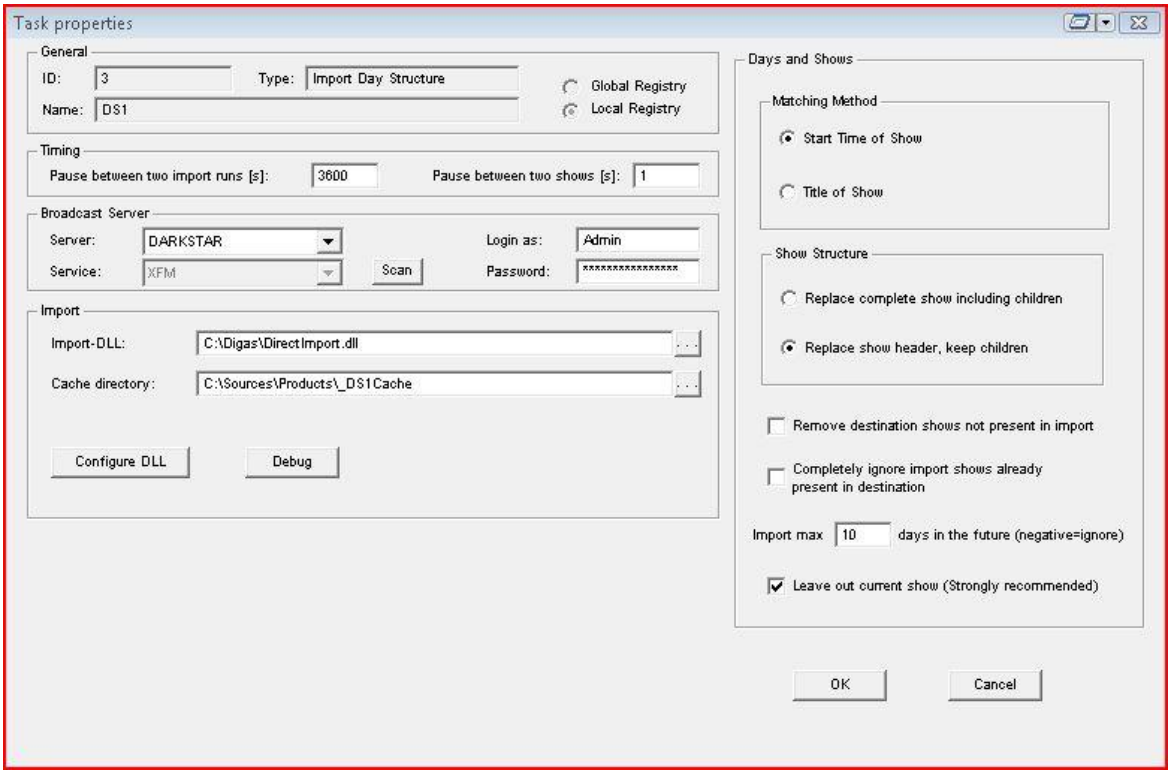
The task runs with a configurable interval and treats all found import files up to a configurable number of days into the future. Every day is treated only once, a re-import is not done.

In principle the information of the import file replaces the destination day. But there is also a merge process between the imported and the existing data:

- If there is a matching show in the destination day (in the BCS) for a show in the import day (the file), the show of the destination day is kept and the details of the merge process are determined by the configuration explained below.

  If no matching show in the destination day is found, the imported show is created in the destination day. In case the new show has no track at all, at least track 0 is created automatically.

- The day pool of the destination day will be kept.



GUI elements and their meaning:

Control group "Broadcast Server": BCS Server and Service (Program), DigaSystem account

Control group "Import"

Import-DLL: Import-DLL

Cache Directory: A local and dedicated directory used to save internal information for the BUS task.

Control group "Days and Shows"

Matching Method:

This setting determines, which shows should be considered as matching – existing shows in the BCS vs. shows found in the imported item.

Show Structure:

* "Replace complete show including children"

  For matching shows the scheduling tracks (0-999) with content are removed from the destination show and then from the imported show they are inserted into the destination show.

* "Replace show header, keep children"

  For matching shows only the properties of the destination show itself are replaced by the properties of the imported show.

Remove destination shows not present in import:

Shows in the destination day, which cannot be matched to shows in the import day, are deleted.

Completely ignore import shows already present in destination:

Shows in the destination day, which can be matched to shows in the import day, are not touched at all.

Setting this option disables the options in "Show Structure".

Import max. days in the future:

Number of days counted from NOW, for which the task tries to import items.

Leave out current show: The current show will not be touched by the import.

Other configuration

* Pause between two import runs: meaning obvious

* Pause between two shows:

  Wait time in seconds between the import of the shows of a day, one second might be reasonable.

7.13 Rebroadcast

The Rebroadcast task copies sent shows from the past onto shows in the future, which are flagged as rebroadcast-shows.

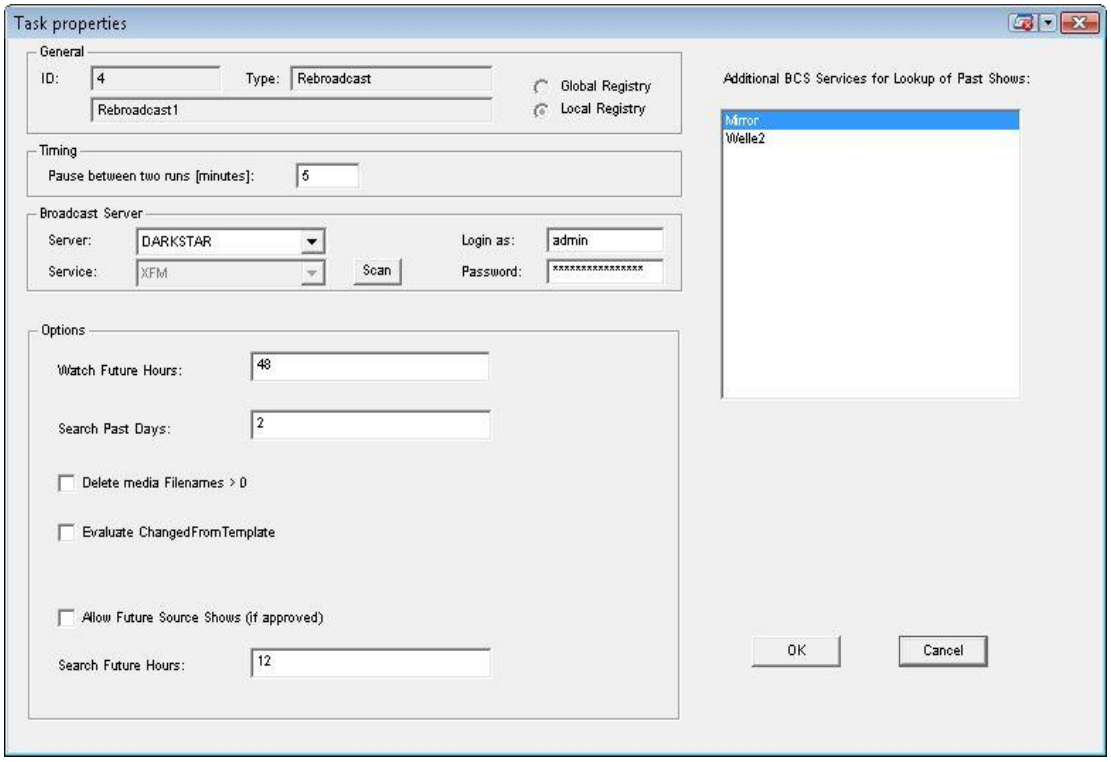
In the XML data of a show two new fields have been introduced: <ShowID> and <Rebroadcast>. The ShowID field (any string) identifies the rebroadcast-shows and the Rebroadcast field is a flag (0/1) indicating that a show is a re-broadcasted show. (The ID and the flag can be set in DigAIRange in the manage dialog for shows and in the show status dialog.)

The task runs every M minutes and looks up to N hours into the future to find empty shows with a set rebroadcast flag and a non-empty ShowID. If such a show is found the task looks up to K days into the past to find a show with the same ShowID.

The task treats every show only once.

When the task copies an old show it performs some adjustments:

- Only the scheduling tracks (0-999) are copied.
- Send states are reset (sending, sent, skipped -> planned).
- Start times of fixed start elements are adjusted to keep the same offset relative to the show start.
- If desired (configurable) all file specifications except Filename0 (the original database file) are removed. This is not done, if a Filename0 does not exist – in this case the other existing filenames are kept. Besides, presentations are always copied into a media directory of the destination show.
- The file state is reset (accessible, mirrored -> existing).



GUI elements and their meaning:

Control group "Broadcast Server":

BCS Server and Service (Program), DAVID account

The service selected here determines where shows in the future (starttime > NOW), which are marked as rebroadcast shows, have to be looked for.

"Additional BCS Services for Lookup of Past Shows":

The shows in the past, which are used to fill the shows in future, have to be searched for first in the service mentioned above, and if this was not successful searched for in the marked services of this listbox.

Control group “Options”:

"Watch future hours"

Number of hours from NOW, where rebroadcast shows have to be searched for.

"Search past days"

Number of days from NOW in the past, where "source shows" for the filling of rebroadcast shows have to be searched for.

"Delete media Filenames > 0”

After the copy of the scheduling tracks from the source show to the destination show, on the element level of the destination show this cleanup is performed:

If tag File_Filename0 exists then all occuring tags of shape File_Filename<N> with N > 0 are removed from the element.

"Evaluate ChangedFromTemplate"

Future shows are touched only if the field ChangedFromTemplate is not set.

“Allow Future Source Shows (if approved)”

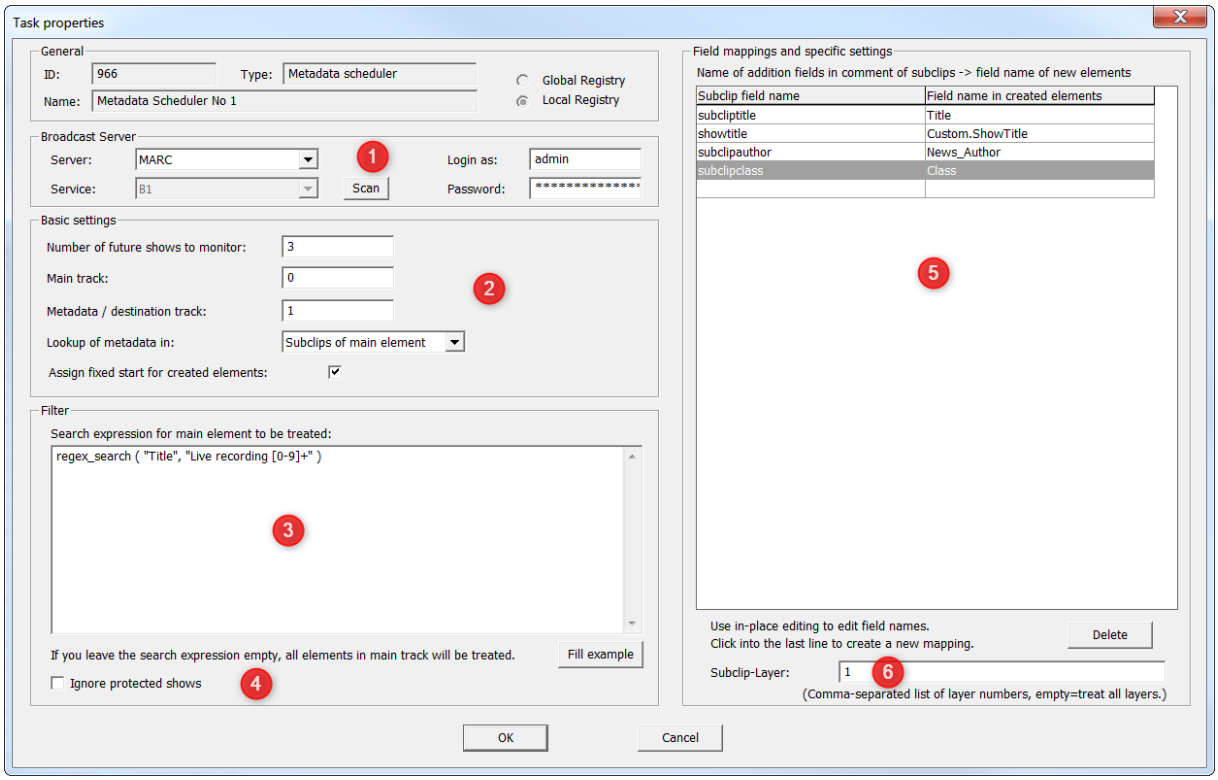
Shows in the future are allowed as search shows if this option is set and the respective shows are approved.
The search for a suitable source show starts “Search Future Hours” in the future then and goes back in time from there.

“Search Future Hours”

See “Allow Future Source Shows (if approved)”.

7.14 Metadata scheduler

The Metadata scheduler is part of a workflow to re-use broadcast content. There is an extra chapter below (-> “Reusing broadcast content / Metadata track” which describes this workflow. This chapter assumes that you have read the general workflow description and describes only the settings dialog of this task – which looks like this:



1) In the group “Broadcast Server”, you can configure the BCS and the service on which the task will operate together with logon credentials.
2) In the group “Basic settings” you have the following settings:
   - Number of future shows to monitor: If set to 0 the task will monitor only the current show. You can set a higher number to achieve that some future shows are monitored, too.
   - Main track: This is the number of the track, which contains the live recordings.
   - Metadata / destination track: This is the number of the track in which this task will create – and possibly also update and delete – live metadata elements for each recording element in the main track.
   - Lookup of metadata in: Two options are available: the first one is: “Subclips of main element”. This means that the task will evaluate the subclip information of the recordings and use this information to create the live metadata elements. The second option is “BCS-Protocol”. In this case the base for generating metadata elements is a BCS protocol corresponding to the original time when the elements, which were recorded by the live recording were played out. If you select the second option the right side of the settings dialog changes – see below.
   - Assign fixed start for created elements: If you check this option, the task will create groups and live elements with a fixed start time. (In the chapter about reusing broadcast content below you can see an example.) Fixed times are helpful in order to see when a live element is going to be started. The start itself is independent of the fixed start time, because TurboPlayer will internally trigger the start if a specific subclip or control position within the live recording is reached. If you prefer having floating elements, uncheck this option. The task will automatically assign floating start type if the computed start time is outside of the show.
3) In the search expression of the filter group, you can define an expression, which must yield true for all recording elements, which should be treated by the task. The chapter 7.1.8 “Filter expressions” above explains how to write these expressions. The example in the screenshot would treat all elements which have the text “Live recording “ plus a number with 1 or more digits anywhere in the title field. If you do not specify a search expression, all elements, which are found in the main track, will be treated.
4) The option “Ignore protected shows” can be set to make the task leave out all shows, which have the state “Approved + protected”.
5) The field mappings allow converting fields, which are hidden in the comment of subclips, into regular BCS fields. The problem is that the available fields for a subclip in the DigaSystem database are very limited. Therefore all

known workflows miss-use the comment field to store additional data. The task can handle fields, which are stored as a JSON string, as an XML string or a section string. Examples:

```
{
    "subcliptitle": "Money, money",
    "subclipshow": "Morning show"
}
```

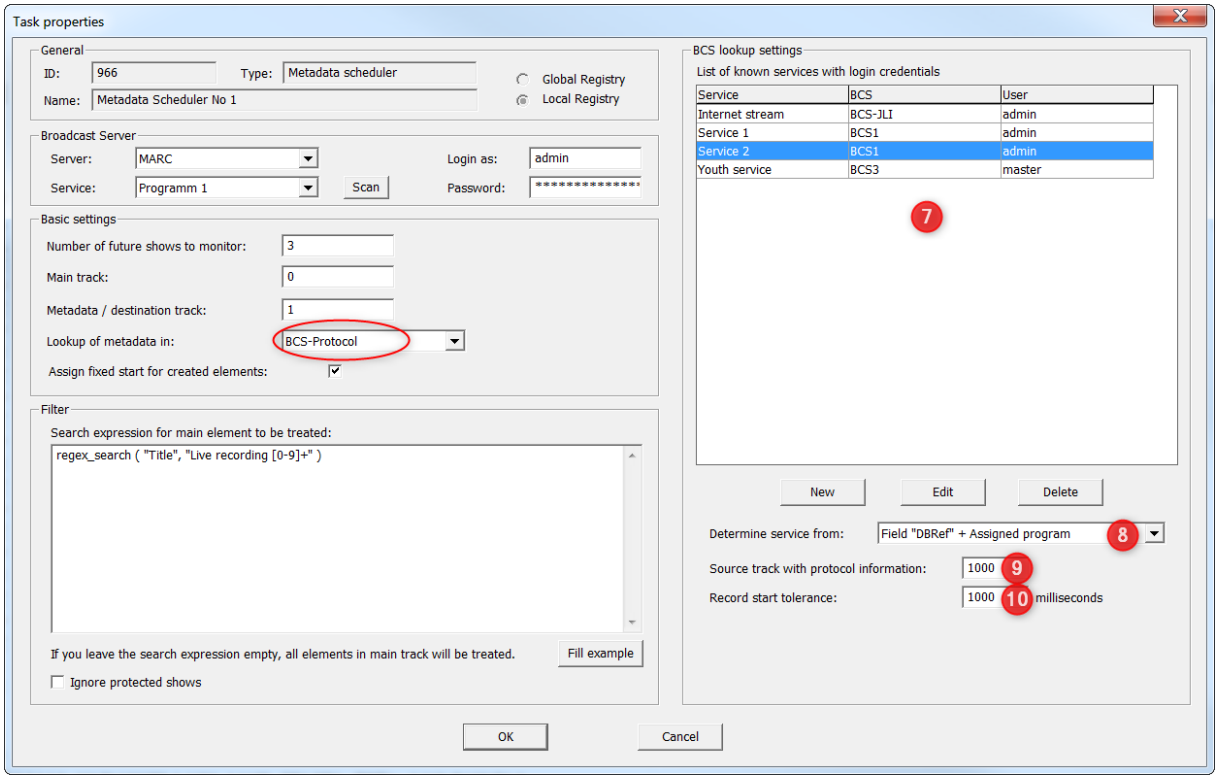
```
<Root>
    <subcliptitle>Money money</subcliptitle>
    <subclipshow>Morning show</subclipshow>
</Root>
```

```
[subcliptitle]Money money
[subclipshow]Morning show
```

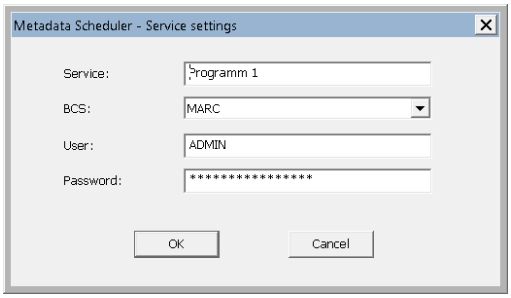
In all three cases the fields must appear on the level below the root. Deeper hierarchies are not supported. Now you can configure that a field like "subcliptitle" should appear as field "Title" in the generated live metadata elements.

6) The subclip-layer setting defines which layer value a subclip must have to be used as a source for generating live metadata elements. You can enter multiple layer numbers by separating them with a comma. If you leave this setting empty, the layer of the subclips will be ignored and all found subclips will be handled.

If you have selected the option "Lookup of metadata in: BCS protocol", the right side of the settings dialog looks different in order to configure other parameters:



7) The task is able to look up protocol elements in a different service and a different server as the primary one – the one configured in the group "Broadcast Server". The "List of known services with login credentials" provides the necessary information to the task how to connect to all the services/servers, which need to be queried. If you edit one entry of the list, you will see that the server name is a drop down box:



You can select only BCS names, which are configured in the DigaSystem registry in key "Digas\Planserver". The service name is a plain text field. You must assure yourself that you enter a correct service name. User and password are the login credentials to be used for connecting the server. The task will only establish a single connection to each BCS, even if this BCS hosts multiple services. Therefore, you should specify exactly the same credentials for each service of a single BCS. The task will keep the connection to such a server open and will close the connection if it has not been used for more than 10 minutes. If you list additional services of the

primary server of the task, it is not necessary to specify login credentials. The existing main connection will be used, anyhow.

8) "Determine service from" tells the task how to determine the service in which it should look up the protocol For this purpose metadata of the live recording and the DigaSystem registry can be evaluated. The following 3 settings are possible:
   - Use primary/configured service: The task does not evaluate any data of the live recording. It always uses the service and the server as they are configured in "Broadcast Server".
   - Field "DBRef" + Assigned program: The task evaluates the field "DBRef" of the live recording element. This field has the format <data source name>\<table name>\<reference number>. The data source name and the table name are extracted and a lookup in the registry should yield the service name. Therefore it is necessary that the parameter "AssignedProgram" is configured in the key "Digas\Database\<data source name>\<table name>".
   - Field "Program": This metadata field is read out from the live recording and it is used as service (=program) name.
9) In "Source track with protocol information" you can enter the number of the track in which the protocol information of played elements can be found. This need not necessarily be a protocol track. Any existing track number is o.k. The task only reads information of elements, which have send state sending or sent.
10) The "Record start tolerance" solves a specific problem: If TurboPlayer triggers a live recording when an element is starting, it can and it will happen that the real start time of the played element is earlier than the start time of the live recording. If you later reschedule the live recording element, the task will pick all elements from the protocol, which have a start time, which is after the beginning of the recording. Without a slight tolerance, the first played element could be missed. Use something like ~ 1 second, also depending on the time synchronization of the playout and the recording computer.

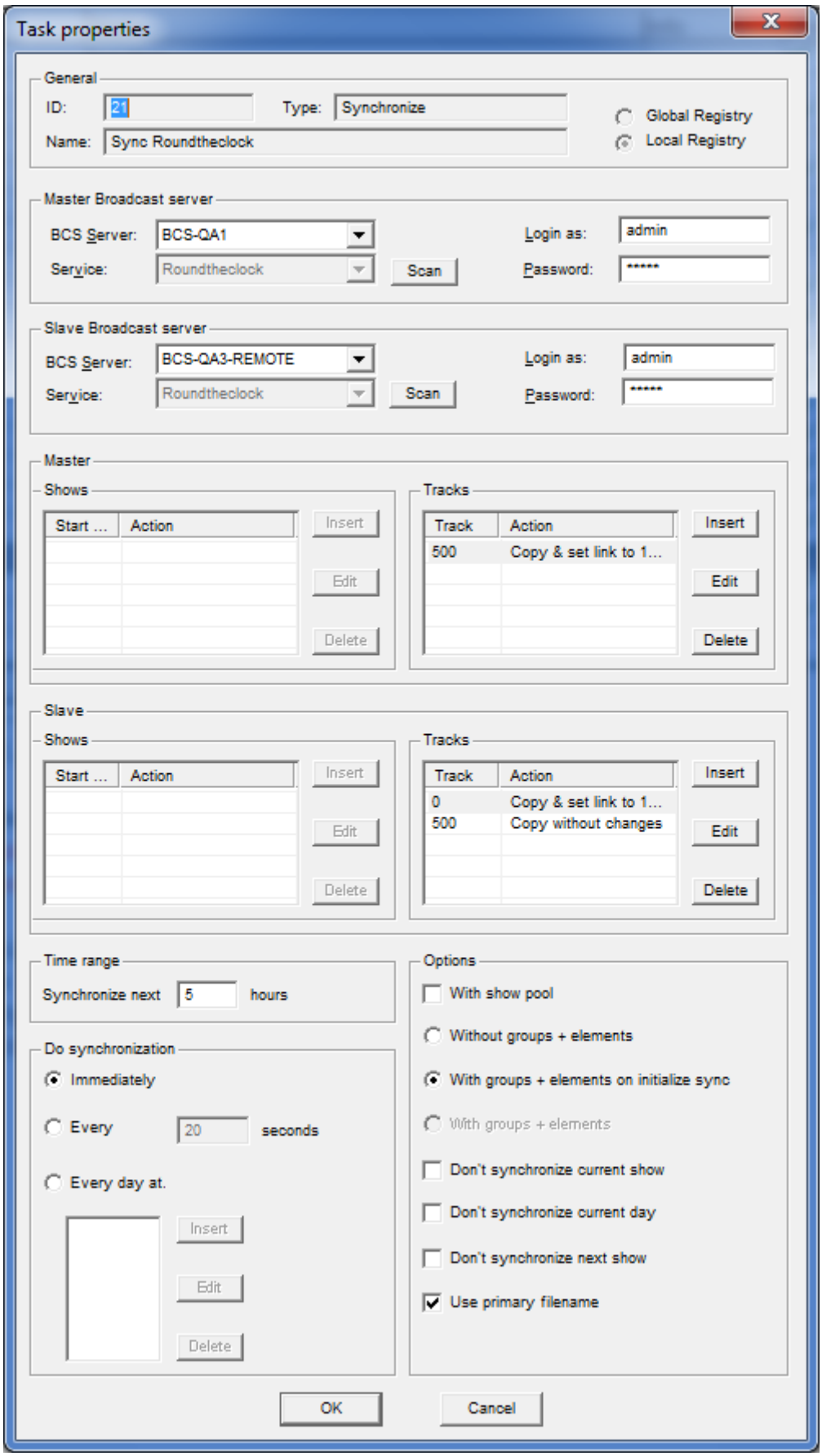
7.15 Synchronize

The Synchronize task synchronizes the shows from one BCS Server/Service (Master) to a different BCS Server/Service (Slave). The synchronization is unidirectional.

The example below works with one BCS Server and two different Services, but as this BUS task is mainly intended to support failover for playout with TurboPlayer it allows using different BCS Servers also.

Remarks:

- It is not allowed to manually insert, move or delete items on the slave side (e.g. using DigAIRange).
- The Synchronize task does not copy files or change filenames, nor does the BCS on the slave side.
- The TurboPlayer that is connected to the slave BCS must therefore have access to the directories where the files are saved by the Master BCS.


Task properties
General
ID: 21
Type: Synchronize
Name: Sync Roundtheclock
Global Registry
Local Registry
Master Broadcast server
BCS Server: BCS-QA1
Service: Roundtheclock
Scan
Login as: admin
Password: *****
Slave Broadcast server
BCS Server: BCS-QA3-REMOTE
Service: Roundtheclock
Scan
Login as: admin
Password: *****
Master
Shows
Start ... Action
Insert
Edit
Delete
Tracks
Track Action
500 Copy & set link to 1...
Insert
Edit
Delete
Slave
Shows
Start ... Action
Insert
Edit
Delete
Tracks
Track Action
0 Copy & set link to 1...
500 Copy without changes
Insert
Edit
Delete
Time range
Synchronize next 5 hours
Do synchronization
Immediately
Every 20 seconds
Every day at.
Insert
Edit
Delete
Options
With show pool
Without groups + elements
With groups + elements on initialize sync
With groups + elements
Don't synchronize current show
Don't synchronize current day
Don't synchronize next show
Use primary filename
OK
Cancel

BCS master	
BCS Server	IP or resolvable name of BCS master
Login	BCS master login
Password	BCS master password
Service	BCS master program name

BCS slave	
BCS Server	IP or resolvable name of BCS slave
Login	BCS slave login
Password	BCS slave password
Service	BCS slave program name

Master shows (BCS master)	
Insert	Insert new show for synchronization (see below)
Edit	Edit currently selected show (see below)
Delete	Delete currently selected show

Warning! Always **all shows** (except user defined options, see below) will be synchronized “as is” (mirror mode). This list is only for definition extra actions for remote linked shows

Master tracks (BCS master)	
Insert	Insert new track for synchronization (see below)
Edit	Edit currently selected track (see below)
Delete	Delete currently selected track

Warning! Only specified tracks will be synchronized. If tracks list is empty, than **all tracks** will be synchronized

Slave shows (BCS slave)	
Insert	Insert new show for synchronization (see below)
Edit	Edit currently selected show (see below)
Delete	Delete currently selected show

Warning! Always **all shows** (except user defined options, see below) will be synchronized “as is” (mirror mode). This list is only for definition extra actions for remote linked shows

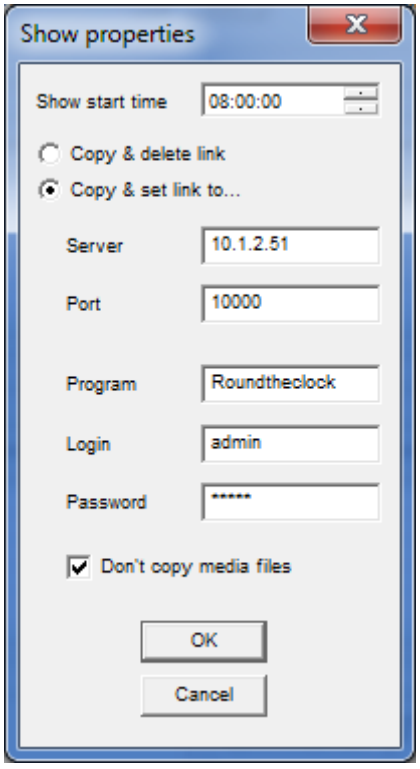
Slave tracks (BCS slave)	
Insert	Insert new track for synchronization (see below)
Edit	Edit currently selected track (see below)
Delete	Delete currently selected track

Warning! Only specified tracks will be synchronized. If tracks list is empty, than **all tracks** will be synchronized

Do synchronization	
Immediately	Do synchronization action immediately after every change on BCS master.
Every XX second	Do synchronization action every XX seconds. All changes between synchronizations are stored in memory and then flush once to BCS slave.
Every day at…	Do synchronization action at user defined time(s). All changes between synchronizations are stored in memory and then flush once to BCS slave.

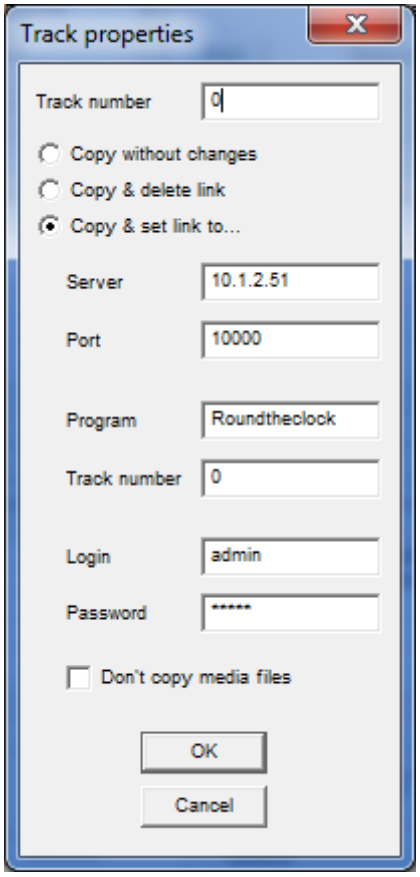
Time range	
Synchronize next … hours	Synchronization interval. The BUS task synchronized all shows with start OR (not AND) stop time between current time and end of interval

Options	
With show pool	Synchronize tracks and show pool in shows (otherwise tracks only).
Without groups + elements	Synchronize only shows and tracks without groups and elements.
With groups + elements on initialize sync	Synchronize only shows and tracks. Groups and elements will be synchronized only once on start sync.
With groups + elements	Synchronize shows and tracks with groups and elements.
Don't synchronize current show	Show is current if current date & time are between show's start and stop date & time.
Don't synchronize current day	Day is current if current date is equal show's start or stop date.
Don't synchronize next show	Show is next if this is the 1-st show after current or the 1-st show if no current.
Use primary filename	Copy only primary filename from BCS master, otherwise all filenames will be copied.



Shows	
Start time	Show start time
Copy & delete link	Copy show and remove remote link from show.
Copy & set link to…	Copy show and set remote link to another BCS
Server	IP or resolvable name of BCS
Port	BCS port
Program	Program at BCS
Login	BCS login. Used only for checking is program on BCS really exist.

Password	BCS password. Used only for checking is program on BCS really exist.
Don't copy media files	Flag about remote access to files



Tracks	
Track number	Track number
Copy without changes	Copy track without changes (mirror mode)
Copy & delete link	Copy track and remove remote link from track.
Copy & set link to…	Copy track and set remote link to another BCS
Server	IP or resolvable name of BCS
Port	BCS port
Program	Program at BCS
Track number	Track number
Login	BCS login. Used only for checking is program on BCS really exist.
Password	BCS password. Used only for checking is program on BCS really exist.
Don't copy media files	Flag about remote access to files

Key for disable check for remote links

Name: CheckRemoteLinks
Possible values: 0 (check disabled) or 1 (check enabled)
Default value: 1
This key disabled check for remote link. This can be usefull for remote links to another BCS (no master and no slave).
Warning! It's possible with disabled check to organize "ringed" configuration with remote links. E.g. BCS A -> BCS B -> BCS A. **Be aware!** This configuration is not allowed.

7.16 Consistency Check

7.16.1 Introduction

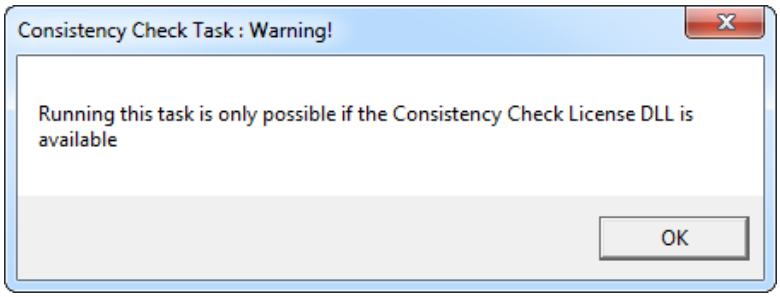
Consistency check is a background task hosted in BUS to carry out automated checks for the content and structure of a show and report the “health” status of the show. Consistency check enables reporting and monitoring of BCS rundowns in terms of content and scheduling. For quality management and to assure service continuity, the control room needs a centralized monitoring of all system components and predictive alerts to get proactive notifications concerning show consistency and the status of the show preparation. Logical issues in the rundown like gaps or overlaps, availability of all necessary metadata, or for instance, availability and integrity of all media files, etc. are analyzed based on BCS metadata, and if inconsistencies are found, steps could be taken to correct them before they affect the OnAir program. This is especially important for automatic shows or for shows where a variety of people and systems contribute to the content of the show. The consistency check task proactively analyzes the rundown schedule in advance, and reports inconsistencies to a centralized alert monitoring system. Inconsistencies are reported in a file in XML format, which can be easily forwarded to a website or an external monitoring or reporting system. Therefore, the help of the BUS task “Consistency XML Writer” is needed (see below). Since version 6.0 of BUS and BCS the results of the checks can be stored in the BCS tree, too. From this place they can then be read from DigAIRange or WebDigAIRange and can be visualized.

7.16.2 How it is used (in short)

Below is a short overview of how this task can be configured and used.

- BUS versions 5.x need a license dll “BCSConsistencyCheck.dll” to enable the consistency check task as a whole(*). Since version 6.0 the consistency check is included without extra license.
- A BUS consistency check task is created.
- The broadcast server settings are defined in the configuration dialog of this task.
- A “Consistency XML Writer” task is selected and / or writing the results back to BCS is activated.
- At least one checkset should be configured with at least one start trigger.
- In this checkset, more than one check can be selected and configured.
- Multiple checksets can be created and configured (every checkset must have at least one start trigger) and combined within one task.

*If the license dll is not found, a BUS 5.x will display this message when the task is opened:



Detailed explanation of the consistency check task configuration dialog, checksets, start triggers, checks and the XML writer task are given further in this chapter.

7.16.3 Components

Below is a description of the main concepts of the consistency check task, which are involved in the configuration and execution of the consistency check task.

7.16.3.1 Checksets

The task allows an unlimited number of checksets to be defined. Each checkset is a combination of the following:

- Name of the checkset
- One of the start triggers with options
- Tracks to be treated
- An optional filter expression for show names
- Consistencies to be checked for

Each checkset is independent of all other sets. All sets run in the same thread, i.e. the single thread of the consistency check task. If the start triggers in the checksets are configured for the same time, they run one after the other in an undefined order. If an administrator wants to run checks simultaneously, he has to use multiple tasks or multiple BUS instances. The checks in the checkset are executed for each track in the specified show. If no problems are found for a track, this track is also reported, but as a good track.

7.16.3.2 Start triggers

Start triggers define when a checkset should run. The task treats shows on the whole. It does not cut shows at the borders of a configured future time / interval apart and treats only a part of the elements of a show.

Every checkset can have one of the following triggers:

1) Check … before show starts
2) Check daily at
3) Check continuously
4) Check from external request

More information about start triggers and how they can be configured is provided in item 11 and 12 of 7.16.4 Configuration of the main ConsistencyCheck task dialog.

7.16.3.3 Consistency checks

The following checks, which are at the element level, are currently available in the consistency check task.

- Files
- Gaps of elements
- Overlaps of elements
- Over- or underfilled groups
- Show limits
- TurboPlayer stop warning
- TurboPlayer load failure
- TurboPlayer start failure
- Media duration
- Loudness
- Playout channels
- Misc
- Generic metadata

In general, elements not meant for classic radio playout (w.r.t. distribution endpoints) are ignored by the checks. Detailed explanation of each of these checks is given in Section 7.16.6 Configuration of the individual ConsistencyCheck dialogs, where also their configuration is dealt with.

7.16.3.4 Reporting to a central XML file

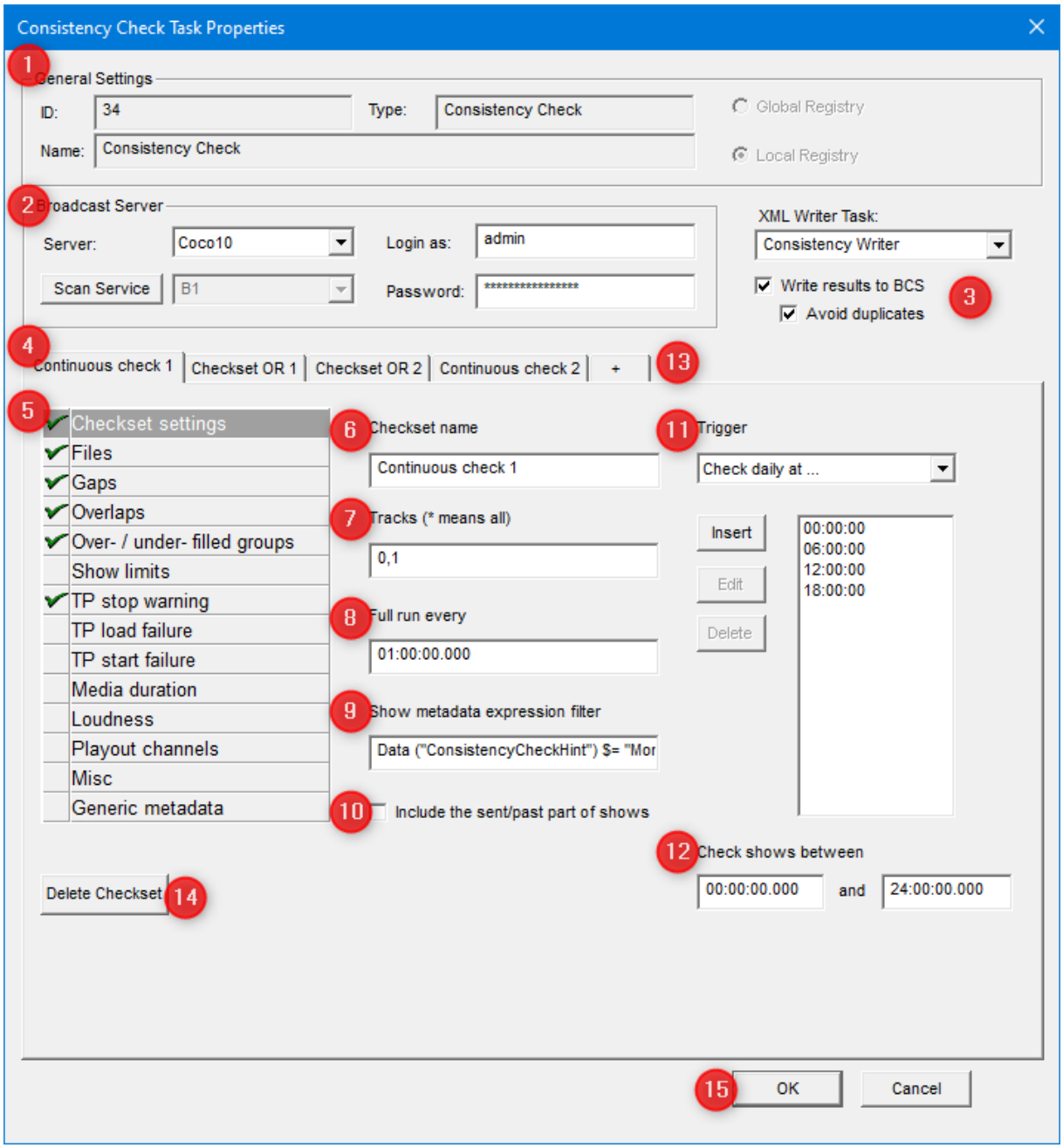
All task instances should be able to write reports about consistency issues into one shared file in XML format. This report can be converted to an HTML report via XSL. The consistency check task allows multiple clients to read from / write to the file. If the file is locked, multiple tries are done to open it. In order to work (especially for deletion) only with the inconsistencies of the right checkset, a "CheckID" is used. The CheckID contains the computer name, the BUS instance, the task number and the checkset ID.

7.16.3.5 Reporting back to BCS

Since version 6.0 of BUS the task can report its results back to the connected BCS. Therefore, BCS needs to have version 6.0 or higher, too. All results are stored in the subnode "ConsistencyCheckResults" with fixed ID 00000005 of program nodes. Each checked show/track gets a subnode and below that each individual report is entered as "Inconsistency". For more details see the description of the XML data structure in chapter 2.

Deletion of these reports is done by a background thread in BCS. This happens n minutes after the end of the checked show, wileas <n> can be configured. This is done in the registry parameter "BroadcastServer\<Config name>\Settings\MaxAgeForConsistencyCheckResults". The value is in minutes after the end of each show. BCS will also remove consistency check results automatically if the corresponding show is being deleted.

7.16.4 Configuration of the main ConsistencyCheck task dialog



After a BUS consistency check task has been created, it can be configured as usual by opening it using a double click on the task. Steps below correspond to the numbering in the screenshot above.

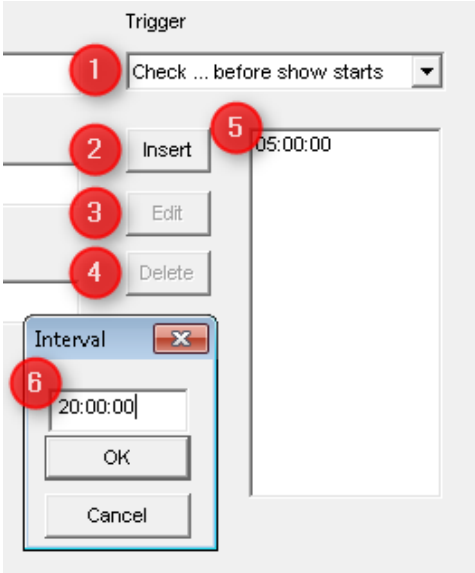
1) Task properties are predetermined by the configuration during the task creation.
2) Similar to other BUS tasks, broadcast server settings for BCS server, BCS login, BCS password and BCS service have to be configured. These parameters are used in every configuration specific to this task independent of checks, start triggers, and any other parameters.
3) Here you have to select to which destination the results should be sent. This can be the BCS, where the examinations are made and/or an XML writer task. Such a task has to be created in advance, for it to appear in the dropdown list. It writes the consistency check results to an XML file as configured in the XML writer task. More about the XML writer task is described later in this chapter in Section: Consistency XML Writer Task
   The parameter "Avoid duplicates" applies only to writing the results to BCS. It is useful if you configure multiple checkset and/or tasks, which check the same shows and perform partially the same checks. Each of these checksets/tasks will report its found problems independently of the others and as a result, the problems might appear multiple times in the results. You can avoid that by checking the option to avoid duplicates. This will cause the task to look for the same report by other checksets/tasks and if it is already present, a report will not be added to the results.
4) On first use, a new checkset is automatically created in the tabbed dialog.
5) On the left side of this tabbed dialog, a list is present that contains the checkset settings and all the checks that can be selected. Selecting any of these using the green check mark means, including the checks in this checkset for the task to carry them out. Please note that “Checkset settings” is always selected since this is not a check, but the settings of the checkset. Any item in this list can be configured by clicking on that item (is highlighted in a blue color when clicked). The corresponding configuration dialog appears on the right inside the tabbed dialog.

Configuration of checkset settings

6) Checkset name: On the creation of a new tab, this field is set to “New-Checkset” by default. Please enter a meaningful name for the checkset which is readable, e.g., “12 hours in advance”, “1 day in advance”. This field also corresponds to the name of the tab. Any changes in the name are automatically reflected in the tab name. If left empty, it is automatically named as “Unnamed” when this field loses focus.
7) Tracks: Enter the track numbers that need to be treated, for example "3, 5" or "*" for all schedule tracks (<1000). It is set to ‘*’ by default. It is only possible to enter numbers and commas in this field.
8) Full run every: This field specifies how often a full run must be carried out. When configured across multiple checksets, the smallest configured full run interval is picked in order to be frequent enough. It is set to 1 hour by default, when a new checkset is created. Please use values greater than or equal to a minute to avoid overloading the system. In a full run, start triggers are calculated based on user configuration and show timings and sorted. Unless the checkset is configured to run in a continuous mode, and no shows are found, show information is read only in a full run. The shows read depend on the user configuration of intervals/times, based on which, the maximum extent to which shows need to be treated is calculated. In the beginning, always a full run is performed, and then based on the configured interval value, a full run is performed. If a checkset is configured to run in a continuous mode, the full run also performs an initial check and subsequent checks are made on a notification basis. For the trigger type "Check from external request" a full run interval must be set, too. In this case the task only makes basic work like checking the BCS connection, but it does not perform any checks in a "full run", it simply continues to wait for an external start trigger.
9) Optional Expression filter on show metadata. Only shows are checked that show metadata matches to this expression, e.g., Data ("ConsistencyCheckHint") $= "Monday". See chapter “Macros and filter expressions” for details about the syntax. If the filter expression is empty, no filters are applied, and every show is checked independent of its metadata.
10) The ConsistencyCheck task is intended to monitor the immediate future and give a warning in case of upcoming problems. It is not intended to make an evaluation or a statistic of past rundown parts. Therefore, the checks are normally only performed for elements, which are planned/cleared, whileas sent/skipped elements are ignored. Hint: depending on the exact check, sent or playing elements are taken into account, nevertheless. For example, a gap between a playing element and the next planned one is reported, whileas a gap between two sent elements is not reported. If you want to change this behaviour and perform the checks on all sent/skipped elements, too, activate this option.

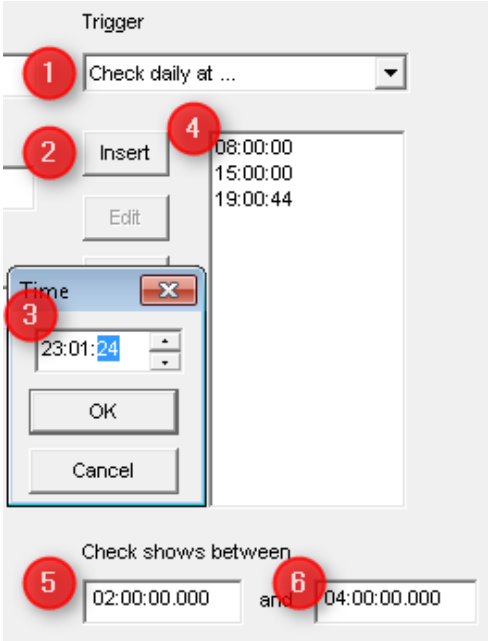
11) Trigger: One of the 4 start triggers explained below can be chosen. By default, “Check … before show starts” is selected. Note: For the modes "Check ... before show starts" and "Check daily at ..." the start trigger list must contain at least one value.
12) Time range for checking based on current time point. Only shows which overlap the range <now + interval minimum, now + interval maximum> are treated. See details below.

12.1) Check … before show starts: A check can be performed at a fixed time before show start. The checks configured in this checkset are run XX:YY (hours:minutes) before the start of a show. Past shows are not checked, only past elements in the current show are checked.



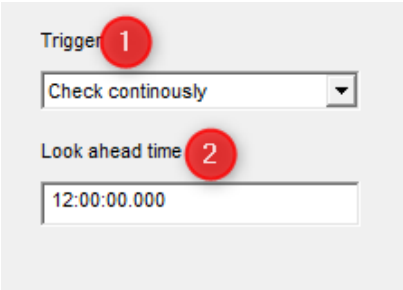
Selection of this start trigger is made as shown in the figure above at (1). An interval before show start can be chosen by clicking on “Insert” at (2). A pop-up dialog appears where the time interval can be set at (6). The configured interval now appears in the list at (5). Multiple intervals can be configured for each set. Each of these values can be edited by clicking on “Edit” at (3) or deleted by clicking on “Delete” at (4).

12.2) Check daily at: Checks can be carried out at planned time points every day. Past shows are not checked, only past elements in the current show are checked. If the time occurs in the past, the configured time is saved for the next day’s check.



Selection of this start trigger is made as shown in the figure above at (1). The time points are selected by clicking on “Insert” at (2). A pop-up dialog appears where the time interval can be set at (3). The configured time now appears in the list at (4). Multiple time points can be configured. In addition, in the option “*Check shows between … and … ”*, a time interval must be configured by specifying the minimum value at (5) and maximum value at (6). Only shows which overlap the range <now + interval minimum, now + interval maximum> are treated. This filter is applied while filling up the start trigger list based on show information retrieved for the maximum interval in future in a full run cycle.

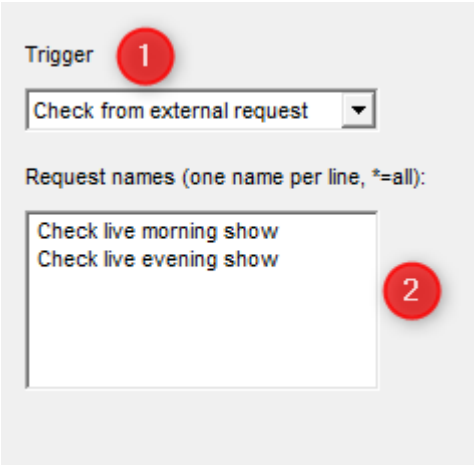
12.3) Check continuously: The checks configured in this checkset are carried out in a continuous mode (once every 3 seconds) Continuous mode is triggered based on BCS notifications. Past shows are not checked, only past elements in the current show are checked. If the show list is empty, shows are re-read. If a checkset is configured to run in a continuous mode, the full run routine performs an initial check and subsequent checks are made on a notification basis.



Selection of this start trigger is made as shown in the figure above at (1). A time interval in the future specified by “Look ahead time” at (2) must be configured, which relates to how far into the future the shows need to be collected for the check. All shows which overlap the range <now, now + look ahead time> are continuously monitored and checked after every change.

12.4) Check from external request: For this start trigger BUS does not start automatically. It waits for a trigger, received via BCS from another BCS client. You can use DigAIRange to send such a start trigger. In the tree view of DigAIRange you need to execute the menu command "Trigger BUS consistency check", which is available for all show nodes of the timeline. The request is then sent via BCS broadcast message to all running and connected BUS consistency check tasks. The message contains an information about the service and the show to be checked.

Since version 6.0 DigAIRange allows the user to select a specific request, meaning: the user can select a "request name" from a list of configured names and this name is then sent via BCS messages to all running BUS tasks. A listening BUS CC task will react on the request if the name specified in the request matches one of the names, which are configured in the settings:



These request names can be configured in the box (2). It is possible to enter the name of the checkset into this box as single request name, of course. This can make things clearer for the administrator. But then a user can request to run exactly one checkset. In contrast, with the extra list of request

names the configuration can be more flexible. It makes it possible to start multiple checksets on a single user request.

If you specify a single star * as the only request name (which is the default), this acts as a joker. The checkset is then triggered by any request with any name – even an empty one.

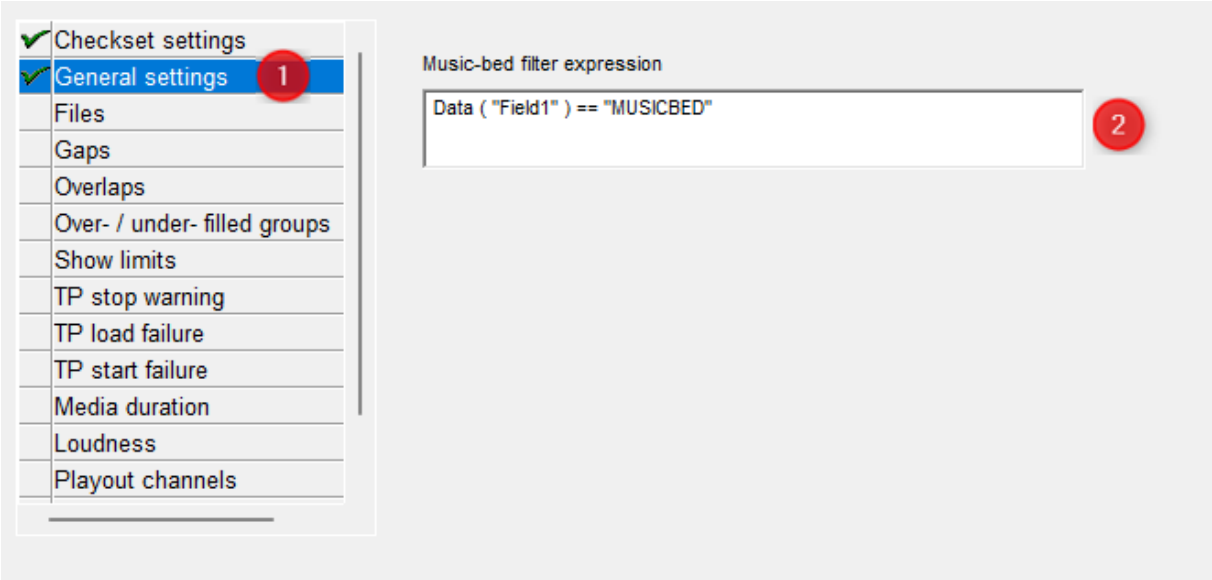
The show, which is checked is given by in the request message. The user in DigAIRange chooses one show for which the request is sent. This can be any show – in the future but also in the past.

Addition and deletion of checksets

13) New checksets can be created by clicking on the '+' button on the top of the tabbed checkset dialog
14) Checksets can be deleted by clicking on the "Delete checkset" button at the bottom left corner of the tabbed checkset dialog. The current selected checkset is deleted from the tab control. If this was the only checkset present, it will be deleted and a default checkset will be created and automatically reinserted to the tab control. In case more checksets are present, the previous checkset is automatically selected and its properties are loaded.
15) Changes made in the configuration dialog are saved when clicked on "OK".
16) Changes made in the configuration dialog are ignored when clicked on "Cancel".

7.16.5 Configuration of the ConsistencyCheck general settings

1) "General settings" item is always checked (same as "Checkset settings").
2) Macro filter expression for identifying music-bed elements can be entered. This is used for modifying the behavior of the Gaps and Overlaps checks: when "Report a gap inside a music-bed element" and "Report overlaps inside a music-bed element" checks respectively are enabled, filtered elements are ignored during those checks, i.e., Gaps and Overlaps inside such elements are reported.



7.16.6 Configuration of the individual ConsistencyCheck dialogs

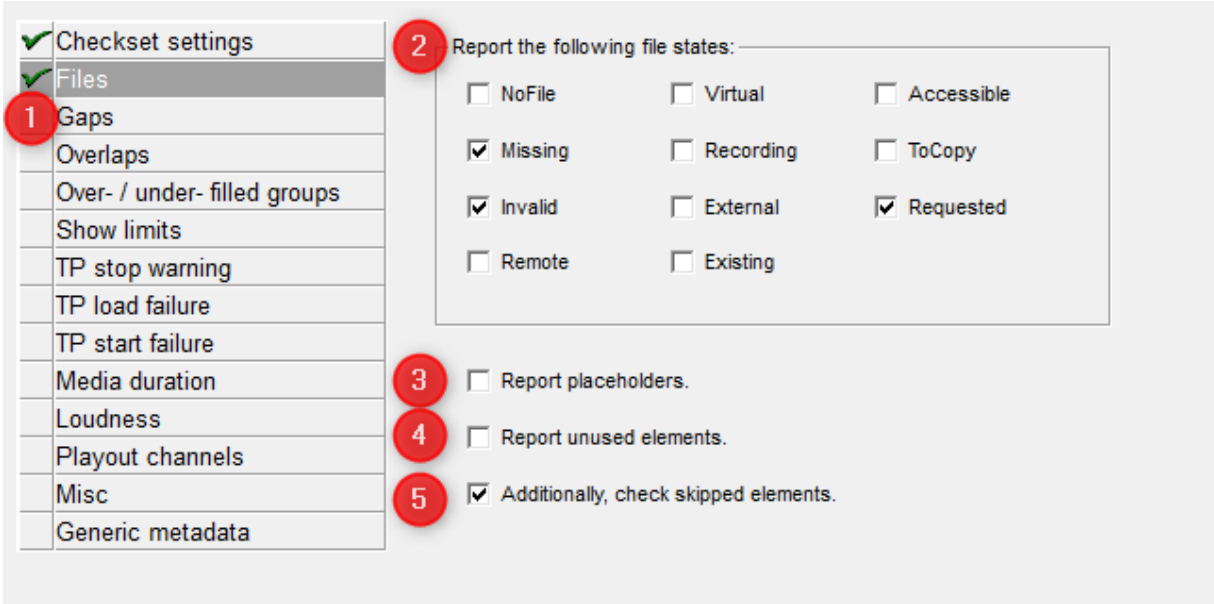
***** In the descriptions below, the case "unused element" is fulfilled if either Time_StartType="Unused" or Time_StartMode="Unused" or Reserve="Unused".

7.16.6.1 Files

This check warns against missing or corrupt media files by evaluating only the metadata that is set by "File check" task in BUS. Because it's important once again: this task does not check any files! It only evaluates the metadata fo elements. The file analysis itself must be done by FileCheck task.

One additional hint: most of the file states refer only to the main media file but not to an optional, additional presentation text. This is only different for the state "Mirrored", which is only assigned by BUS FileCheck if a given presentation text has been copied, too.

If you are unsure about the meaning of the file states, please have a look at chapter 3.2.5: FileState for an explanation of the individual file states.



With respect to the figure above, files consistency check can be configured as explained below.

(1) Selection of files configuration dialog

(2) The user can choose to report the file states as error conditions by selecting the check boxes available here. If the file state of the element is found to be "Invalid", it leads to an error state of "Error: Invalid", and all other file states mentioned lead to an error state of "Error: Missing". By default, the file states “Missing”, “Invalid” and “Requested” are selected.

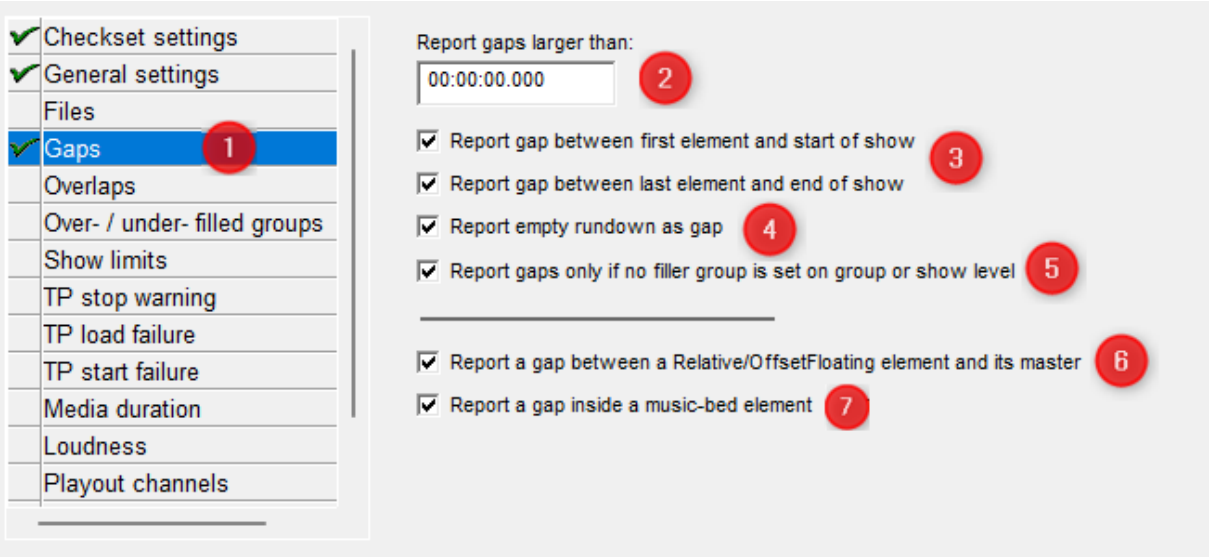
(3) This checkbox option tells the check whether intentional blank elements i.e., placeholders (decided by SendState), are to be reported or not.

(4) It is also possible to configure the check to report unused elements. The check compares the metadata fields Reserve, StartMode and StartType of the element to filter out such elements.

(5) This option refers to: future elements, skipped by the user. Elements in the past, which have been skipped automatically by TurboPlayer are already coverd by the general option for a checkset to treat past elements.

7.16.6.2 Gaps of elements

Gaps are empty places in a rundown without a planned element, which might result in on-air silence. Calculations are based on mark times, i.e., if the stop time of the current element corresponding to Mark-out is earlier than the start time corresponding to Mark-in of the next element, there is a gap which should be reported. Show borders are also taken into consideration for the checks. Placeholders, overlaid, skipped, info and unused* elements are ignored by the check.



With respect to the figure above, consistency check dialog for gaps can be configured as explained below.

(1) Selection of gaps configuration dialog

(2) Here the gaps time should be configured, which when exceeded, should be reported as a gap error by consistency check, in other words, the maximum tolerated gap time is configured here.

(3) Activate these two options if you want to get a gap between the start of a show and the first element in a show and/or the end of the last element in a show and the end of the show reported.

(4) By default, a completely empty rundown is not reported as a gap. If this should happen, activate the option. Hint: elements, which are typically not being played, like: placeholders, unused* elements, info elements or skipped elements do not count.

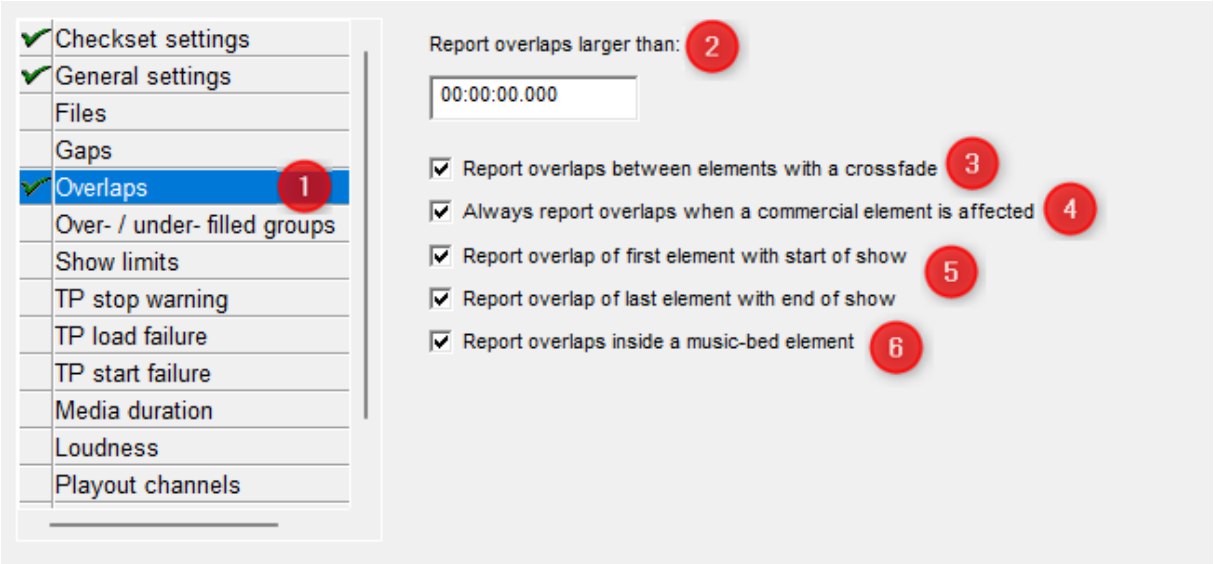
(5) This option tells the check if gaps are to be reported only if there is no filler group defined for the area (either group or show) of the gap. When this option is not checked, gaps are reported irrespective of fillers for shows and groups.

(6) This option checks for invalid metadata. An element with a start mode “Relative” and start type “OffsetFloating” is intended to start relative to its preceding “master element”. The difference between the start of both elements is given by the field “Time_StartOffset”. This offset must be in a range so that the relative element starts during the playtime of its master elements. If the offset is too small (or negative) or too big, then the relative element would start before the master element starts or after the master element has already stopped. This cannot be handled by TurboPlayer. Hint: the minimum time of field (2) is not applied for this check.

(7) This option enables music-bed filter from general settings. Elements which are filtered out are ignored during this check. This feature makes it possible to detect gaps inside music-beds.

7.16.6.3 Overlaps of elements

Overlapping elements in the show can be reported using this check. Calculations are based on mark times, i.e., if the stop time of the current element corresponding to mark-out is later than the start time corresponding to mark-in of the next element, there is an overlap which should be reported. Show borders are also taken into consideration for the checks. Placeholders, overlaid, skipped, info and unused* elements are ignored by the check.



With respect to the figure above, consistency check dialog for overlaps can be configured as explained below.

(1) Selection of overlaps configuration dialog

(2) Here the overlaps time should be configured, which when exceeded, should be reported as an overlap error by consistency check, in other words, the maximum tolerated overlap time is configured here.

(3) By default, the check recognizes overlaps, which were created by making a crossfade between two elements. (Technical background: this means that the second element must have start attributes Floating / Sequenced or Manual and the overlap time must be identical to the sum of the link-values.) Such an overlap is not reported. If you want to get a report, nevertheless, set this option.

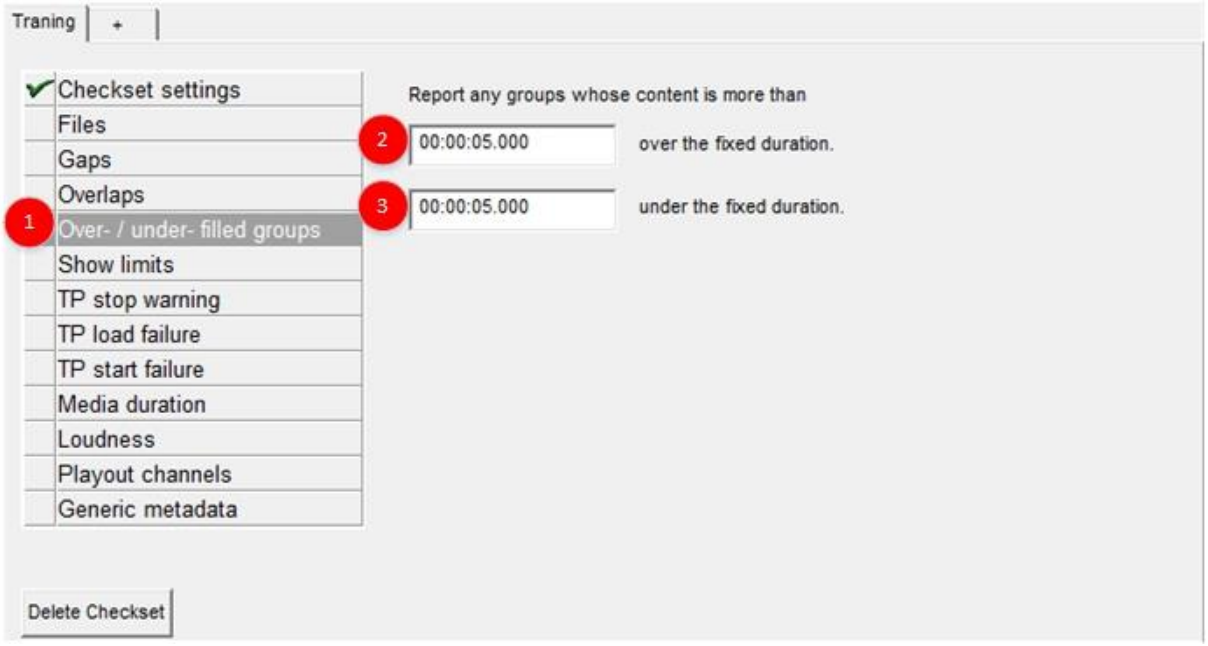
(4) This is only a specialization of the previous option. Even if a regular crossfade should not be reported, it might be desirable to get a report if one of the elements is a commercial element, because commercials should normally be played without any crossfades.

(5) Activate these two options if you want to get an overlap between the start of a show and the first element in a show and/or the end of the last element in a show and the end of the show reported.

(6) This option enables music-bed filter from general settings. Elements which are filtered out are ignored during this check. This feature makes it possible to detect overlaps inside music-beds and ignore overlaps of music-beds with other elements.

7.16.6.4 Over- / under- filled groups with a fixed / planned duration

This check watches for groups, which have the attribute "Time_FixedDuration" (shown as "planned duration" in DigAIRange) set and checks, whether the playout duration of all elements in such a group is exactly the duration of the group itself. The maximum allowed total over/under filling time is configurable (two different options). Placeholders, overlaid, skipped, info and unused* elements are ignored by the check. The planned duration of the group is compared with the total duration of all the elements in the group based on link times.



With respect to the figure above, consistency check dialog for over-/under-filled groups can be configured as explained below.

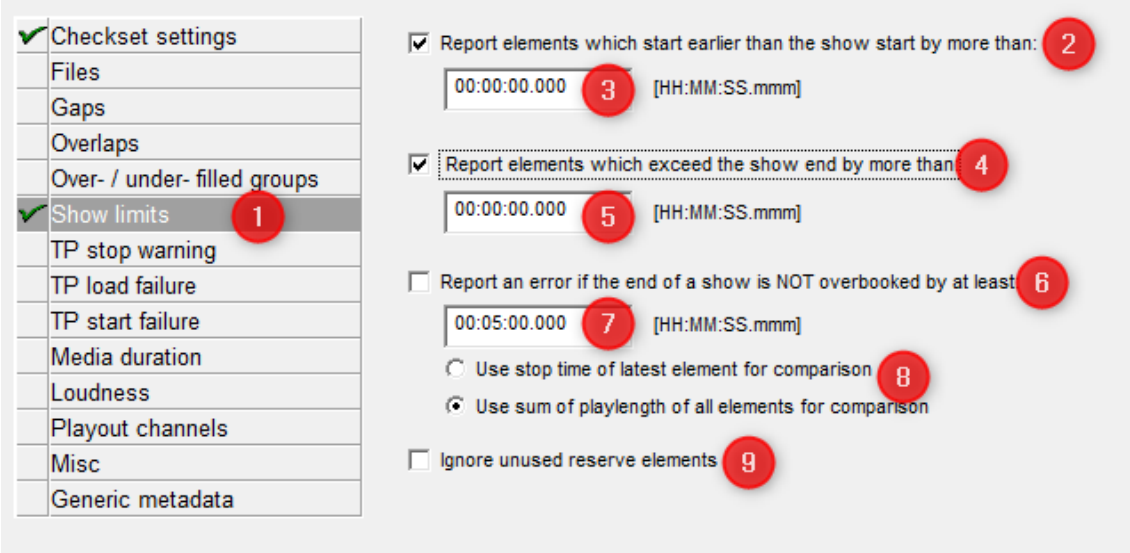
(1) Selection of over-/under-filled groups configuration dialog

(2) Here, a tolerable time duration limit should be configured, such that, when the group duration exceeds the fixed duration by this limit, an over-filled group error should be reported by consistency check.

(3) Here, a tolerable time duration limit should be configured, such that, when the group duration is lesser than the fixed duration by this limit, an under-filled group error should be reported by consistency check.

7.16.6.5 Show limits

If an element exceeds the show limits (show start or show end) by a configurable limit, the consistency check can report an error using the checks available here. Overlaid, skipped, info and unused elements are ignored by the checks. Calculations are based on mark times.



With respect to the figure above, consistency check dialog for show limits can be configured as explained below.

(1) Selection of show limits configuration dialog: Two different checks can be selected here, i.e., “an element is before show start” and “show end exceeded”.

(2) Select check for “Report elements which start earlier than the scheduled start (of the show) by more than”: the start time of an element corresponding to mark-in is before the beginning of the show.

(3) Here, a time duration should be configured, such that an error should be reported by consistency check indicating that the element is not within the show limits when, the element starts earlier than the scheduled start by this duration.

(4) Select check for “Report elements which exceed the show end by more than”: the stop time of an element corresponding to mark-out is after the end of the show.

(5) Here, a time duration should be configured, such that an error should be reported by consistency check indicating that the element is not within the show limits when, the element starts earlier than the scheduled start by this duration.

(6) In contrast to (4) you can check this option in order to make sure that a show is overbooked, meaning: filled with too much content so that it is sure that there is enough content to fill up the show and no gaps are to be expected during playout. Selecting this option only makes sense if you do NOT check option (4), of course.

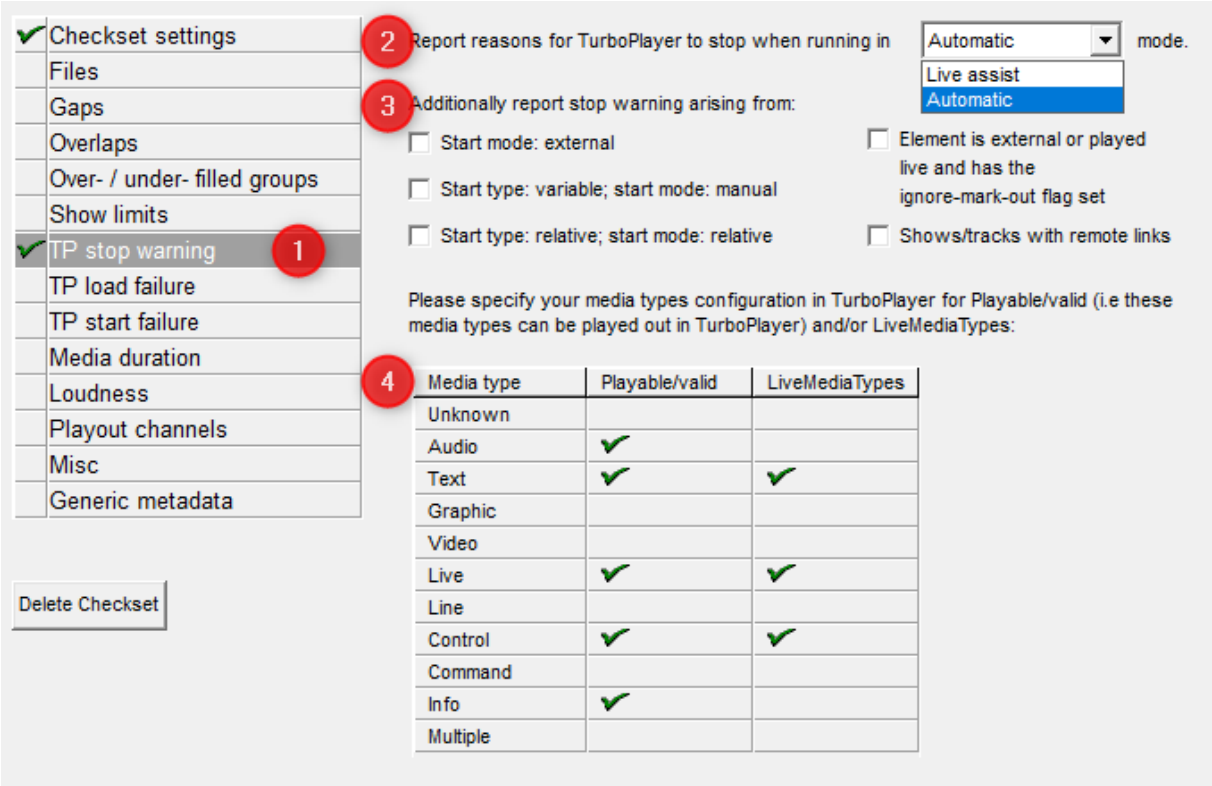
(7) Here, a time duration should be configured, how much time a show must be overfilled.

(8) Here you can select how BUS checks, whether a show is overbooked. In option (1) BUS simply looks at the stop time of the element with the latest found stop time and compares this against the end of the show. This is only useful if you have quite "regular" shows with all elements floating to the beginning. In this case BCS computes the times of the elements by simply adding up the play length of all elements. With option (2) BUS will compute the sum of the play length of all elements in the show and will check, whether the sum exceeds the duration of the show plus the configured overbooking time. (Play length means: the duration of elements between their link points plus a potential text duration, preferring real times if present.)

(9) Unused reserve elements can be ignored by selecting the checkbox.

7.16.6.6 TurboPlayer stop warning

The TurboPlayer stop warning checks whether the rundown contains any combination of elements and / or states which is likely to make TurboPlayer stop during playout (e.g., a manual start element). The TurboPlayer functionality is way too complex to check all possible stop conditions just by examining the rundown in advance. Therefore, the administrator has to set some parameters in order to equip this check at least with some conditions that could lead to a TurboPlayer stop. Info, placeholder, unused and skipped elements are ignored by the check.



With respect to the figure above, the consistency check dialog for TurboPlayer stop warning can be configured as explained below.

(1) Selection of TurboPlayer stop warning configuration dialog

(2) Select option for the mode in which TurboPlayer runs. Default selection is live assist mode. It is necessary to check this because TurboPlayer will stop at a manual element in live assist mode, whileas TurboPlayer will automatically start a manual element in automatic mode. In case you have some shows running in live-assist mode and some in automatic mode, you need to configure two checksets with different options and see how to achieve that a specific checkset only runs for the correct shows (e.g., by different start triggers, show filters or only running on external request).

(3) Select conditions which additionally may lead to a TurboPlayer stop such as some specific start type / start mode combinations, external elements played live and remote links on show or track level. In the table below, reasons why these options could lead to a stop condition can be found.

(4) By configuring the table for allowed media types, it is possible to track down wrong media types that could lead to a stop condition. By default, the Playable/valid types selected are: Audio, Text, Live, Control and Info. By default, the LiveMediaTypes selected are: Text, Live and Control.

The table below gives a good idea of configuration options leading to a stop condition.

Fields to be checked	Necessary parameters	Comments
StartMode=Manual	Active mode = Live Assist?	TP will stop at a manual element in live-assist mode.
StartMode=External	Is this start mode allowed?	TP might stop if no start signal is received.
StartMode=Manual + StartType=Variable	Is this combination allowed?	TP might stop in automatic mode, depending on parameter ActionOnVariableEnd.
FileState=Virtual + SendState=Planned/Cleared	Active mode = Live Assist?	TP will stop in live-assist mode.
StartType=Relative + StartMode=Relative	Is this combination allowed?	Relative starts might not be executed if no start trigger is received.
External or live and IgnoreMarkOut is set	Is this combination allowed? Which media type is played live?	These elements do not stop if they are not stopped manually. The next element will never play.
Remote links on show or track level	Is this combination allowed?	It depends on the TurboPlayer configuration whether remote shows/tracks can be played or not.
Wrong media type (audio, video, graphic, text, live)	Which media type is allowed?	A media type can only be played on specific TurboPlayer / MultiPlayers and certain configurations.

The most important issue which are not / cannot be checked in advance is: Internal element + cannot be loaded and external element + no mixer source available.

Explanations:

Internal means: MediaType = Audio, Video or Graphic and media type is not in the list of live media types; a media file is specified; Source = Digas or Harddisk.

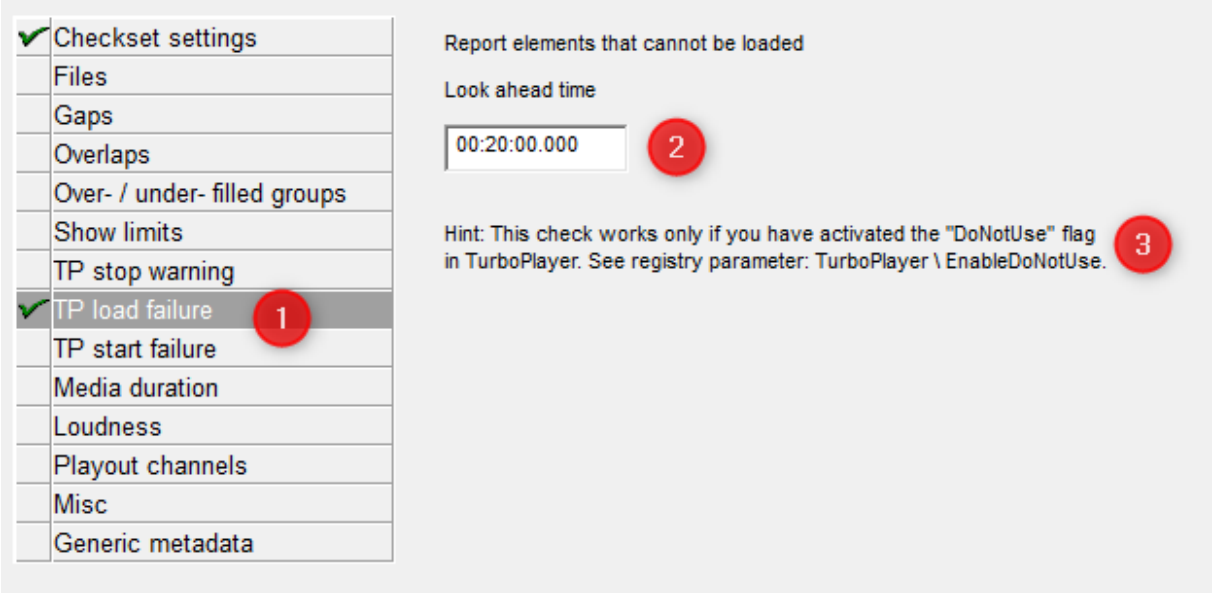
External means: MediaType = Line and media type is not in the list of live media types; Source is one of: CD, Microphone, Telephone, Tape, DAT, Minidisk, Satellite, Line; MixerSource is specified.

Live means: MediaType is in the list of live media types (typically Live, Control); Source = <empty>; MixerSource = <empty> or “*”.

Please see the chapter “Playout categories” in TurboPlayerTechManual for more information.

7.16.6.7 TurboPlayer load failure

This check evaluates the metadata flag "DoNotUse" set by TurboPlayer (assuming that it is set), whenever it cannot load an element.



With respect to the figure above, the consistency check dialog for TurboPlayer load failure can be configured as explained below.

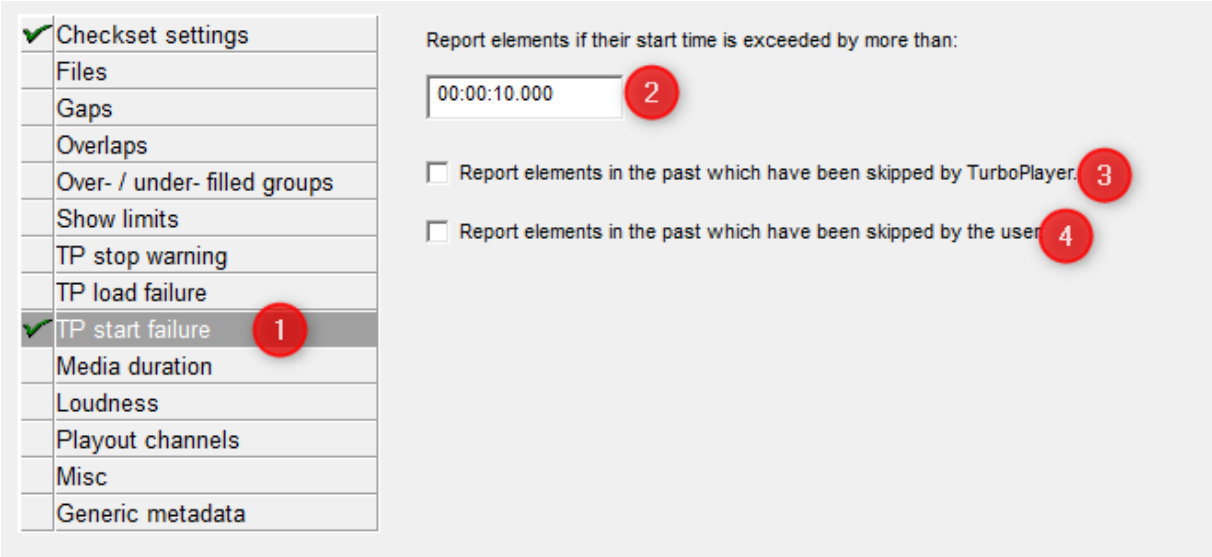
(1) Selection of TurboPlayer load failure configuration dialog

(2) Configure a look ahead time duration within which the check scans for elements in the rundown which has the DoNotUse flag set. Depending on this value, a failure is triggered by consistency check, i.e., if the time difference between <now> and the start of the element is smaller than the configured interval. For ex: if the value of this field is hh:mm:ss, the check will only be made hh:mm:ss before the start time of the element.

(3) The administrator must configure the flag “EnableDoNotUseFlag” under TurboPlayer settings. TurboPlayer can set this flag for elements, which cannot be preloaded by MultiPlayer. This has the effect that BCS does not take the flagged elements into account when computing the times in the rundown, meaning: the flagged element will get a duration 0 assigned. This flag is evaluated by the TurboPlayer load failure check.
(Warning: this flag must not be used if multiple TurboPlayers are working on the same rundown! The flag can only be used if a single TurboPlayer runs each rundown. Otherwise, there might be an endless update loop of 2 TurboPlayers, where the first one signals that an element cannot be loaded, whileas the second TP signals that it can be loaded and removes the flag.)

7.16.6.8 TurboPlayer start failure

This check should only be used together with a continuous start trigger for the checkset. If the scheduled start time (corresponding to mark time) of an element is elapsed (relative to current time) by a configurable time, and the send state of the element does not change to "Sending", an error is reported. For very short elements it is possible that the state directly becomes "Sent". Info, placeholder and unused* elements are ignored by the check.



With respect to the figure above, the consistency check dialog for TurboPlayer start failure can be configured as explained below.

(1) Selection of TurboPlayer start failure configuration dialog

(2) A time duration should be configured, such that, if the start time of the element exceeds this value, an error is reported by consistency check. By default, this field is set to 5 seconds. Pay attention to have a correct time synchronization of TurboPlayer and the computer on which BUS runs.

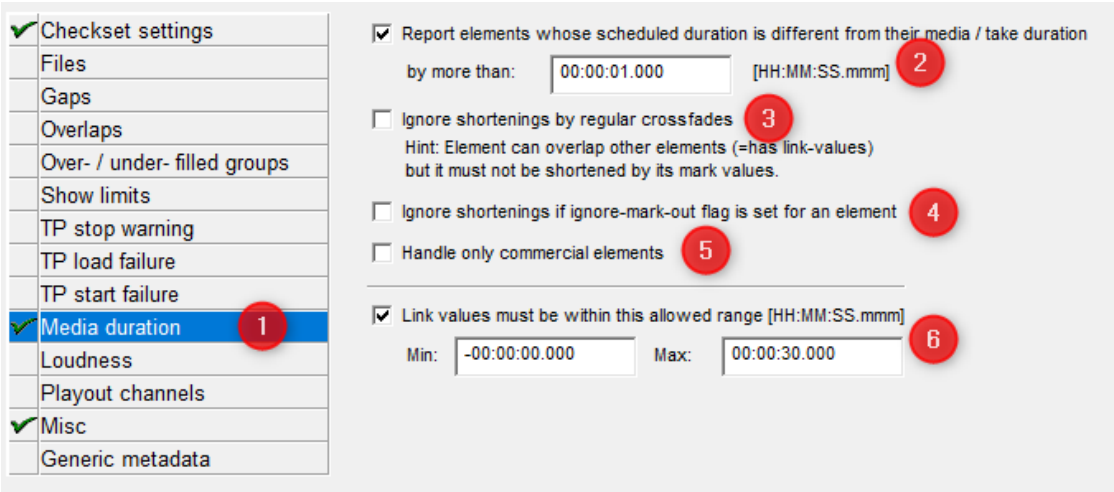
(3) Normally, the check does not report elements, which have been skipped by TurboPlayer (which happens e.g. for not-played elements above a time-started element). This option tells the consistency check to report a start failure for these elements. The configured time in (2) does apply for these elements, too.

(4) Normally, the check does not report elements, which have been skipped by a user. This option tells the consistency check to report a start failure for these elements. The configured time in (2) does apply for these elements, too.

7.16.6.9 Media / take duration

A consistency check can be performed to check for a mismatch of the element's scheduled duration with respect to its media / take duration. The check works only on elements with a given media duration > 0, which are only audio and video elements. In principle, it checks, whether an element has been shortened by its fade values and will therefore not be completely played out.

Only scheduled elements (SendState is "Planned" or "Cleared") and only elements with a file state of "Existing, "Accessible" or "Mirrored" are checked. Info elements are ignored by the check. This check evaluates only the metadata as it is already being set by FileCheck task. Therefore, a running FileCheck task is mandatory.



With respect to the figure above, the consistency check dialog for media duration mismatch can be configured as explained below.

(1) Selection of media duration mismatch configuration dialog

(2) A tolerable time duration can be configured, such that, if the scheduled duration of the elements differs from the media duration by more than this value, a media duration mismatch error is reported by consistency check.

(3) If you set this option, the check ignores a regular crossfade between two elements. (Technical background: this means that the second element must have start attributes Floating / Sequenced or Manual and the overlap time must be identical to the sum of the link-values.) So, this means: it is o.k. if both elements of a crossfade overlap (= they have link values greater than 0). But it is not o.k. and will be reported if one of the elements is shortened (= has a mark-in, which is not 0 and/or a mark-out, which is not at the end of the take).

(4) If this option is set, the check will not report the case that an element is shortened by having a mark-out value, which is not at the end of the take and in addition the ignore-mark-out flag is set for the element. In this case the check can assume that the element will not stop at the mark-out, but instead play till end of take.

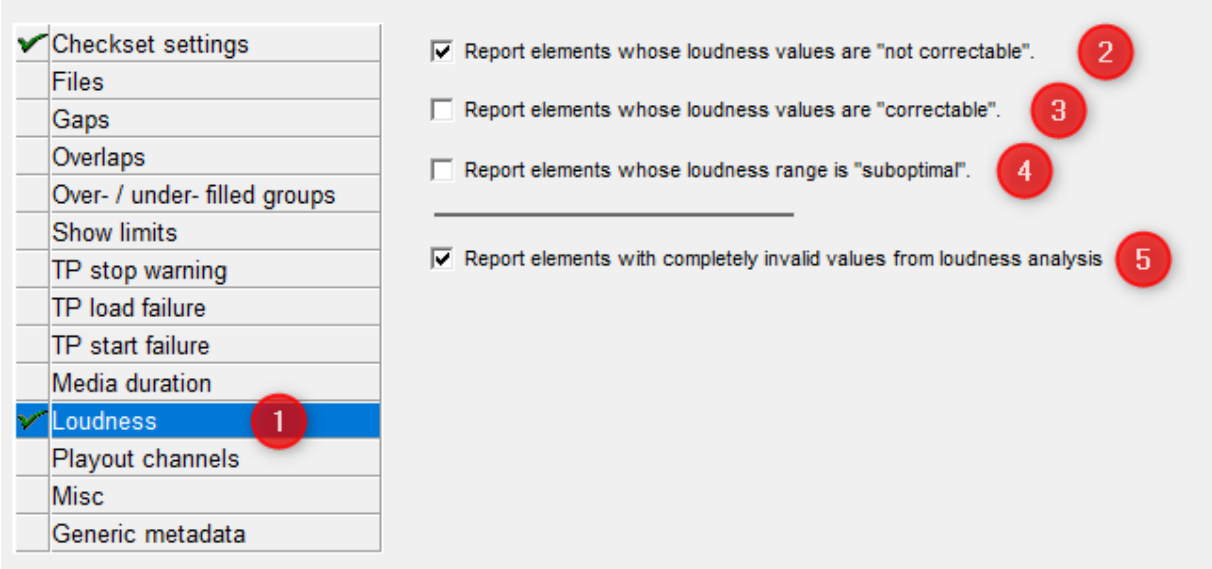
(5) If this option is set the check will evaluate only elements with class Commercial ("Advert"). This can be useful if your commercials must never be shortened, and you want to explicitly check for it.

(6) This check is independent of the checks above. You can let BUS check the link values of elements, whether they are in an allowed range. You can configure the valid range with the two edit boxes. BUS checks both values, the link-in and the link-out, against the limits.
Important: the value 0 is always a valid link-value and it is also the default value. Therefore, you cannot exclude 0. The mimimum value always refers to a negative link value. If you omit the minus sign, BUS automatically assumes a negative value. Negative link values mean that a gap between the elements is created.
The maximum value must always be a positive value. Positive link values are the regular case of two crossfaded elements. Have in mind that a crossfade created by OnAIR TrackMixer will typically have a link-out value with half of the crossfade time for the first element and the same link-in value for the second element. Or in other words: OTM puts the link point into the middle of the mark points of both elements.
Very high positive link values are typically a sign for mis-using the attributes. Sometimes the same scheduling can be solved differently, e.g. with an overlaid element with a relative start.

7.16.6.10 **Loudness**

This check reports loudness states of elements as error according to the options configured. This check evaluates only the metadata as it is already being set by the loudness analysis in File check task (or DPE). Therefore, a running FileCheck task is mandatory. The loudness set for the configured service from the registry is read and evaluated.

Placeholders, unused* elements, skipped elements, info elements and elements with flag "Audio_IgnoreLoudness" are ignored by this check. Only audio and video elements with one of the file states: Existing, Accessible or Mirrored are checked.



With respect to the figure above, the consistency check dialog for loudness can be configured as explained below.

(1) Selection of loudness configuration dialog

(2) Select option if elements whose loudness values are “not correctable” are to be reported.

(3) Select option if elements whose loudness values are “correctable” are to be reported.

(4) Select option if elements whose loudness values are “suboptimal” are to be reported.

(5) This option activates a check for invalid loudness values in the metadata of elements. So far, this check will only verify that the loudness values ILK, LRA, MAXSL and MAXML are not zero. This should never happen and cannot be the result of a correct loudness analysis (except perhaps for something like an artificial audio file, like a pure sinus waveform). Hint: an LRA value of zero is normal for elements with a take duration of less than 4 seconds. Therefore, this combination of values is not reported.

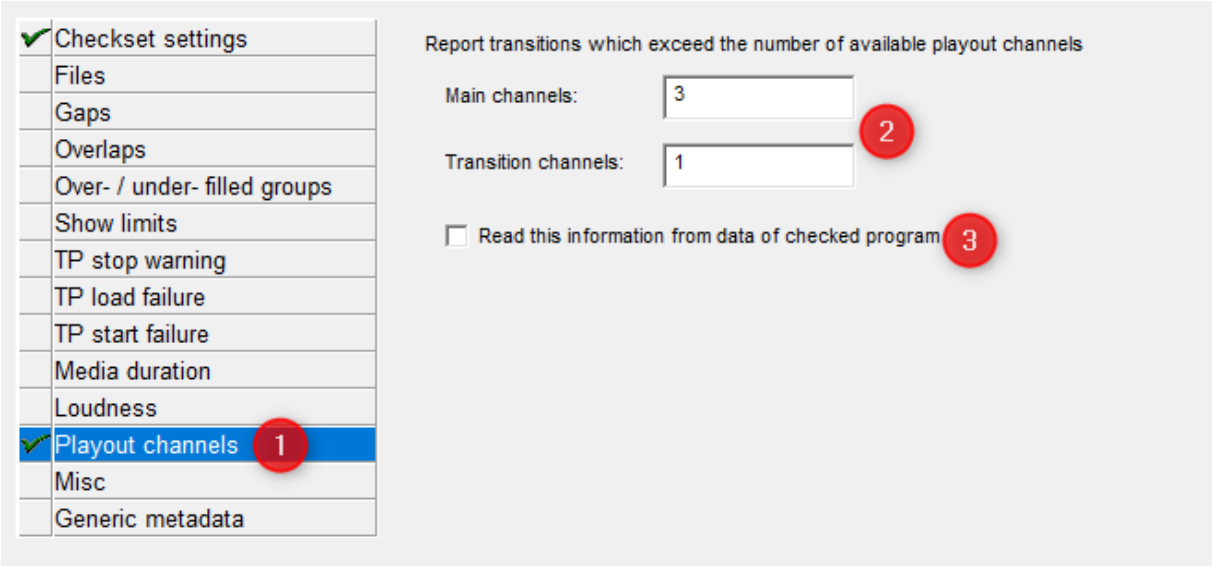
General information:
So far in DigaSystem/BroadcastSystem the only possible correction is a playout level adjustment if the loudness value is too low or too high, but there is no possibility to adjust the dynamic of the audio, like with a compressor. Therefore, if only the level needs to be adjusted, the state is "correctable". If in this situation the loudness range is slightly too big, the state "suboptimal" is assigned. (The limit is given in parameter "MaximumLoudnessRangeOptimal".) If either the true-peak level is too high, os the loudness range is much too high or the maximum shortterm loudness is too high, the state "not correctable" is assigned.

All the settings for the desired values and allowed ranges can be done within a loudness set in the registry below Common\Loudness\<set name>. The set, which is used for a specific service, is specified in the registry value Digas\Planserver\<Server name>\<Service name>\AssignedLoudnessSet.

7.16.6.11 Playout channels

This check verifies if the number of available TurboPlayer channels for playout is sufficient to play the rundown / all sequences as intended. Therefor it must analyze how many elements are running at each time. If this number exceeds a configurable channel limit, a warning is generated.

Placeholders, unused* elements, skipped elements, info elements and past/sent elements are ignored by this check. Aside of that, only audio elements with one of these file states: Existing, Accessible, Mirrored and Recording are checked. Elements with IgnoreMarkOut flag are treated as playing till the end of the take, otherwise it is assumed that they play from mark-in to mark-out. The special TurboPlayer features of class-channel or position-channel assignment are ignored.



With respect to the figure above, the consistency check dialog for playout channels can be configured as explained below.

(1) Selection of playout channels configuration dialog

(2) Configure the channel limit as it is given by your TurboPlayer / MultiPlayer setup. The main channels are used for all elements, which are not flagged as transition elements. This flag is normally set automatically by OnAIR TrackMixer if you have specified transition channels in the program/service settings. Elements with a transition flag are played on dedicated / special transition channels. Therefore, you have two independent channel limits.

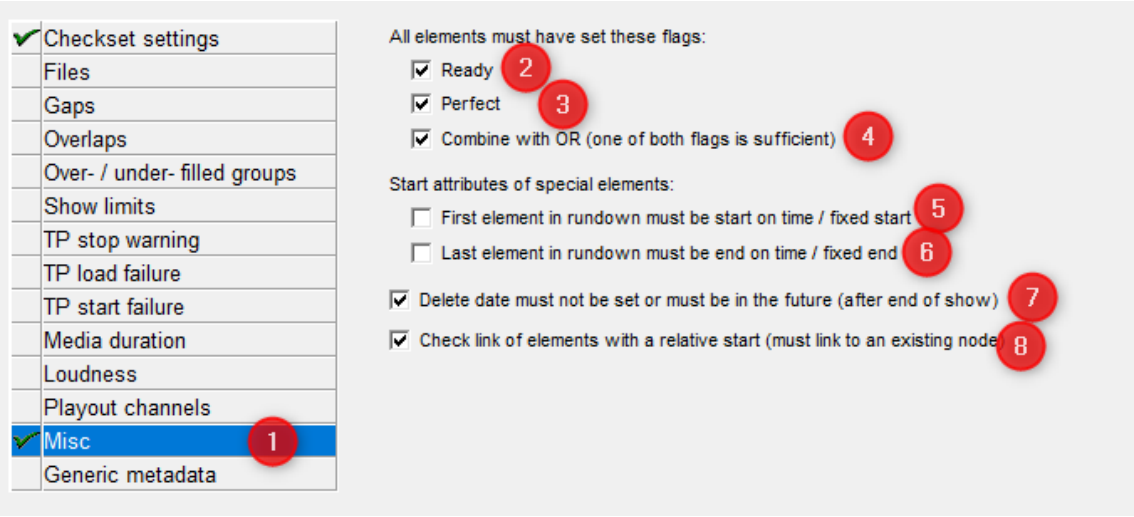
Known limitation: so far this check can only be used in a reasonable way for a single track of a show. It is not possible to configure the available channels for more than one track. If you use multiple tracks (e.g., for regionalization) you would have to setup multiple checksets (one for each track) at the moment. In case each track has the same number of main / transition channels available you can use one checkset for all tracks, of course.

(3) If you set this option, the task will read the number of available channels from the program/service data stored in the BCS tree. In this case it does not matter, which numbers you configure in the two edit boxes above.

Hint: here you have the same problem as in (2) if you operate with multiple tracks. Even worse: the number of available channels cannot be stored for each track in the program/service settings.

7.16.6.12 **Misc**

This check assembles multiple more or less specific checks, which do not fit well into any of the other categories. Placeholders, unused* elements and info elements are ignored by this check.



With respect to the figure above, the consistency check dialog for misc checks can be configured as explained below.

(1) Selection of misc configuration dialog

(2) Checks, whether all elements have the Ready flag set. Typically, this flag is already set in the DigaSystem database and it signals that an element is ready for broadcasting.

(3) Checks, whether all elements have the "Perfect" flag set. This flag exists independently from the Ready flag and it depends on the customer, what the exact meaning of it is. Therefore, the name of the flag can be configured in the DigaSystem registry in value DBM\ NameOfPerfectFlag. This name is displayed in the dialog. Therefore, you might see a different name than "Perfect".

(4) If you let the task check for both flags, Ready and Perfect, you can activate this option to achieve that it is sufficient, if only one of both flags is set. Otherwise, both flags must be set.

(5) Checks, whether the first element for playout in the rundown has start attributes start on time / fixed start.

(6) Checks, whether the last element for playout in the rundown has start attributes end on time / fixed end.

(7) Checks the field "DeleteDate" of elements. This field must either be empty or if a date is given, it must be in the future, meaning: the delete date must be bigger than the end date of the show.
Though the delete date is not evaluated within BroadcastSystem, it is very well evaluated in DigaSystem. This check should prevent the situation that an element from the database is scheduled in a rundown but no mirror files have been created yet. If the original database element is then deleted too early, the file references in the rundown will dangle and the element is no longer playable.

(8) If you have elements with a relative start in the rundowns, you can use this option to make BUS check, whether the referenced node of these relative starts really do exist.

7.16.6.13 Generic metadata

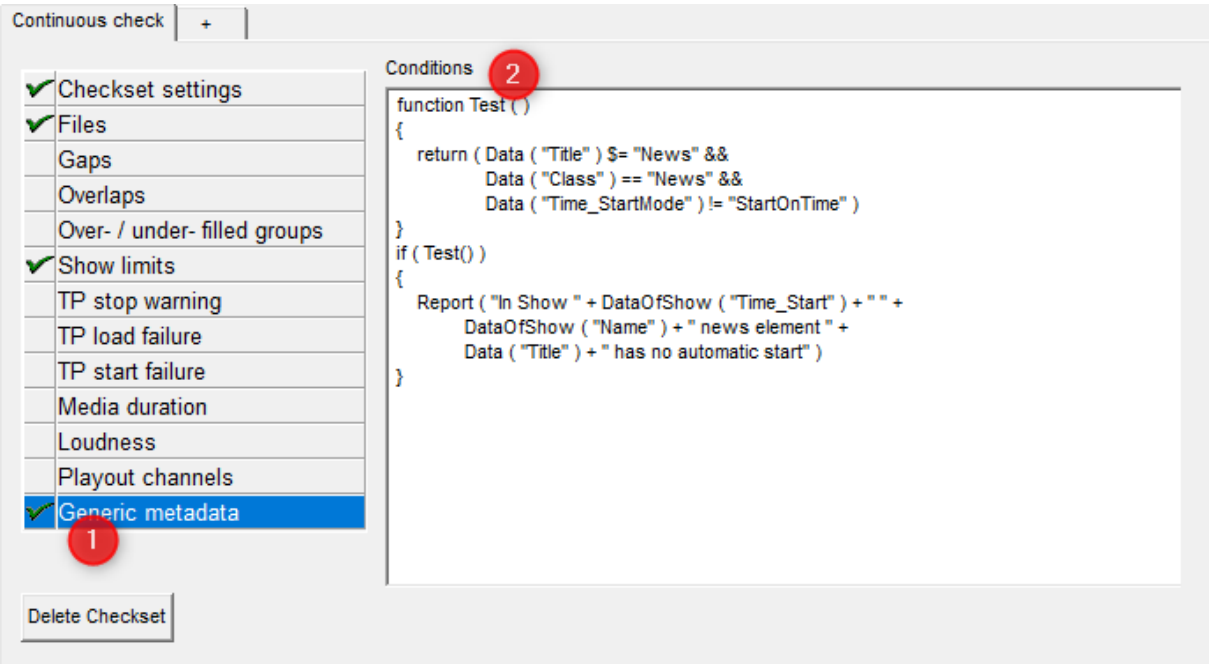
This check verifies any element and show metadata based of expressions. See chapter “Macros and filter expressions” above for details about the syntax. The macro you specify will be called recursively for every rundown node (group, story, element) within the track/show, which is checked. For each of these macro executions the currently checked node is then what is described as the “current node” in chapter 7.1.10 about “Macros and filter expressions” above.

You can write one or multiple if-statements to decided, whether a node must be reported as inconsistency. If a node should be reported as inconsistency, you have to use the procedure “Report”. It expects a single or multiple string parameters. If you specify multiple string parameters, all of them will simply be concatenated. The text within this parameter is the report, which is sent to the ConsistencyXMLWriter taks, which then will write all reports to the output XML file.

An example, which tests, whether elements with class News and the word "News" somewhere in the title have the start mode "StartOnTime":

```
if ( Tagname ( ) == "Element" &&
     Data ( "Title" ) $= "News" &&
     Data ( "Class" ) == "News" &&
     Data ( "Time_StartMode" ) != "StartOnTime" )
{
    Report ( "In Show " + DataOfShow ( "Time_Start" ) + " " +
             DataOfShow ( "Name" ) + " news element " +
             Data ( "Title" ) + " has no automatic start" )
}
```

Since version 6.0 of BUS you can also use defined macro functions to make the code more readable. The screenshot shows the example above while using a function Test() for testing each node.



With respect to the figure above, the consistency check dialog for generic metadata can be configured as explained below.

1) Selection of generic metadata configuration dialog.
2) Enter macro source code here with report about any inconsistences.

Here is a more complex example of a macro, which checks for the existence of a specific control element in track 0 for switching TurboPlayer to automatic mode as first or second element:

```
// Define a function executed at the beginning before parsing
// all nodes of a rundown (=track).
// The function must be called "$Begin" with a '$' at the beginning!
// We only define a global variable to store the result from the parsing.
function $Begin ( )
{
    $FoundSwitchingElement = 0
}

// Define a function executed at the end after parsing all nodes.
// The function must be called "$End" with a '$' at the beginning!
// Here we evaluate the global variable and do a report if
// the element we were looking for was not found.
// The data of the show and of the track can be evaluated in the $Begin()
// and $End() function, too, but not any other data of a specific node.
function $End ( )
{
    if ( ! $FoundSwitchingElement && DataOfTrack ( "Number" ) == 0 )
    {
        Report ( "In Show " + DataOfShow ( "Time_Start" ) + " " +
                 DataOfShow ( "Name" ) + " / track 0 no control " +
                 "element for activating the automatic mode " +
                 "was found as first or second element." )
    }
}

// The next part is called for every node in the rundown.
// We try to identify the control element we are looking for.
// It should be the first or second element in the rundown.
// If it is present, we set the global variable.
// The special state variables %TheNode and %TheElement can be used
// for enumerating all nodes/elements in a rundown.
// These variables are automatically incremented for each call
// of this macro, starting with 1.
if ( Tagname ( ) == "Element" &&
     ( %TheElement == 1 || %TheElement == 2 ) &&
     Data ( "Class" ) == "Control" &&
     Data ( "Title" ) == "Switch to automatic" &&
     Data ( "Control_Operation1" ) == "SwitchToAutomatic" )
{
    $FoundSwitchingElement = 1
}
```

7.17 Consistency XML Writer Task

This task is a helper task for the consistency check task and it is used to collect and assemble reports of ConsistencyCheck tasks and write them as an XML file into the file system. Writing of the XML file is carried out by a separate thread inside BUS to optimize the access times of the XML file which avoids / reduces problems with the simultaneous access of this file. A new BUS task type which hosts the background writer thread can be created. The writer thread being used is configurable in the consistency check task settings. This allows the administrator to adjust the XML files being written.

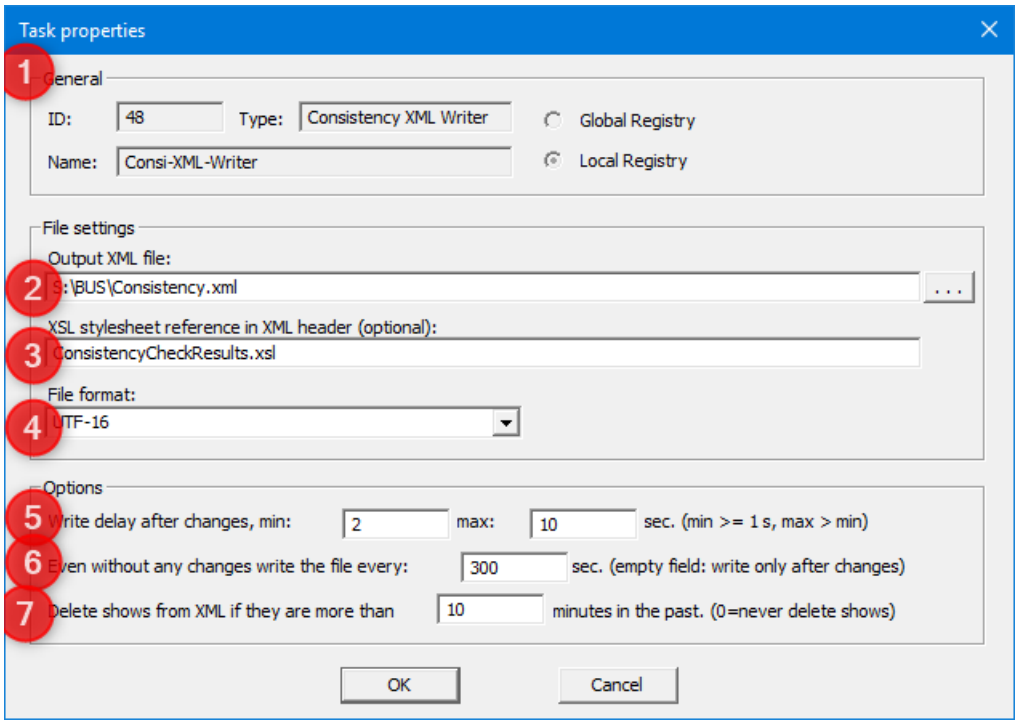
Consistency XML writer task can combine the results of multiple checks / checksets / tasks and BUS instances into a single file. Therefore, the XML is structured in a hierarchy: BCS / Program (=service) / Show / Track and each inconsistency report gets a unique “CheckID” which has the format: computer name \ BUS instance name \ task number \ checkset number. This allows the writer task to identify all reports and cleanup all old reports of one checkset if it receives new reports of the same checkset. Therefore, the administrator is completely free how to configure the writer task. Anything from “each checkset should have its own writer and its own XML file” up to “the results of all checks – even from different computers - should be written into a single XML file” is possible.

Example of a Consistency check result XML file


```xml
<?xml version="1.0" encoding="utf-16"?>
<?xml-stylesheet type="text/xsl" href="ConsistencyCheckResults.xsl"?>
<ConsistencyCheckResults>
 <BCS Name="DESKTOP_SUS">
  <Program Name="Studio_SUS">
   <Show Name="Bayern 1 am Nachmittag" Time_Start="2016-03-23 13:00:00.000"
    Time_Stop="2016-03-23 17:00:00.000"
    ID="00000000/00000064/000007E0/00000003/00000017/00000066" Consistency="BAD">
    <Track Number="0" Name="Bayern 1" Consistency="BAD">
     <Inconsistency>
      <CheckID>DESKTOP_SUS\BUS\18\1</CheckID>
      <Type>StopWarning</Type>
      <ID>00000000/00000064/000007E0/00000003/00000017/00000066/00000002/00000064</ID>
      <Title>Nachrichten</Title>
      <Time_Start>2016-03-23 13:00:00.000</Time_Start>
      <Time_Stop>2016-03-23 13:03:00.000</Time_Stop>
      <Time_Duration>180000</Time_Duration>
      <Description>TurboPlayer stop warning for Element "13:00:00
       Nachrichten" of "Bayern 1 (0)" : it has start mode "Manual", is an external
       element and has the flag IgnoreMarkOut set</Description>
     </Inconsistency>
     <Inconsistency>
      <CheckID>DESKTOP_SUS\BUS\18\1</CheckID>
      <Type>Gap</Type>
      <ID>00000000/00000064/000007E0/00000003/00000017/00000066/00000002/00000065</ID>
      <Title>Wetterbett MIT Nachrichtenstinger nach Regional Topthema</Title>
      <Second_ID></Second_ID>
      <Second_Title></Second_Title>
      <Time_Start>2016-03-23 13:03:00.000</Time_Start>
      <Time_Stop>2016-03-23 13:07:21.144</Time_Stop>
      <Time_Duration>261144</Time_Duration>
      <Description>Gaps found between elements "2016-03-23 13:03:00.000 Wetterbett
       MIT Nachrichtenstinger nach Regional Topthema" and "2016-03-23 13:07:51.144 B1
       Vormittag/Mittag Openerbett 3 D" of "Bayern 1 (0)"</Description>
     </Inconsistency>
    </Track>
    <Track Number="101" Name="Stack Mod 1" Consistency="GOOD"></Track>
   </Show>
  </Program>
 </BCS>
</ConsistencyCheckResults>
```


Please note that all checked tracks, shows, programs and BCS are listed in the file. On show and track level the attribute “Consistency” will be set to “GOOD” if no inconsistencies were reported for the node and “BAD” if there are inconsistency reports.

Configuration of the XML Writer Task Dialog



With respect to the figure above, the Consistency XML Writer dialog can be configured as explained below.

(1) These are the general settings each BUS task has, like the task name. These settings can only be set when a new task is being created. Changing them is not possible.

(2) The “Output XML file” is the filename for the XML file with all checked BCS servers, programs, shows and tracks and all reported inconsistencies.

(3) Here an optional XSL stylesheet file can be configured. The task writes a reference into the XML header (see the example above). This allows to open the file with a browser and to get a nice formatting.

(4) Here the binary encoding of the XML file can be configured. Possible options are ANSI, UTF-8 and UTF-16. For the UTF formats the file can be written with or without byte order mark (BOM) in the first bytes of the file.

(5) The writer task does not change the XML file with every new report it receives from a consistency check task. In order to avoid a too frequent access to the XML file, the writer task collects some reports and writes the XML file only from time to time. Here you can configure the minimum and the maximum wait time of the writer, meaning: if a single report is received, the XML file will be written after the minimum time. If more new reports are received during the wait period, the wait period starts anew. But if the maximum time after the first report has elapsed, the XML file is being written in any case.

(6) Here you can configure whether the task should write the file regularly every n seconds, even if no new reports of the consistency check task were received. Normally, there is no need to do this, because if there is no new information, why should the file be written anew? The reason is in the next setting below. Only the XML writer task is responsible for deleting old shows from the XML. If no new reports are received from the consistency check task for some time, it might happen that old shows stay in the XML during this time. If this is no problem, there is no need to write the file regularly. But if the presence of past shows in the XML disturbs your users, you should activate a regular writing even if there are not any changes in the current shows.

(7) Old reports are automatically removed from the XML file by the writer task. This does also apply for reports which have been written by other tasks or other BUS instances. The single criterion is the age of the report. If it is more than the configured time in the past it will be removed. (Please see setting (6), too.)

7.18 Radio AD-Server Task

DigaSystem RadioAD-Server is "last minute commercial" planning system based on Adremes API. The purpose of the RadioAD-Server task is to permit that last minute blocks of commercials may be initially planned (not necessarily with its real content there) and later have the content inserted based on MotiveID of previously scheduled blocks (items) on BCS side. At the moment not all functions of SAST API are implemented, but only the essential ones for the successful operation.

Also to mention is that RadioAD-Server task only works with a licensing dll, which is provided by David. If you don't have the dll you will be able to change settings but not to run the the task. Together with the dll three more files will be provided, which refer to specific library code used by the SAST SDK.

DigaSystem doesn't delete or create new blocks in a broadcast list in this task, all changes in broadcast list are only possible with already existing blocks. If the supposedly scheduled element is not found, nothing is done and this block is ignored. Every block (item) is supposed to have his own unique id, which in this is case is the MotiveId.

Requisites for the task are the already typical broadcast server data, as seen in the figure above from Settings Dialog of the task, as well as some other new fields that will be described shortly.

First of all, the task processing will make use of a table in database. The list of available tables will be displayed in "Database for planned items". A table MUST be chosen, this is where the final audio element will be stored and referenced in broadcast list. For each change in broadcast list the database will be checked for the database entry with the same id (MotiveId). If the entry is found, it will be then updated (later with new audio, through the MediaAvailable event), otherwise a new entry with the same MotiveId will be created.

Next comes the track number. All changes in broadcast list are possible in one track only per task. If more tracks are desired, more tasks should be created and configured with a different track number. The task will search for the element in broadcast list whose data and hour are inside the limits of a given show in this configured track. If there are overlapping shows, the first one which comprehends this time interval will be picked up.

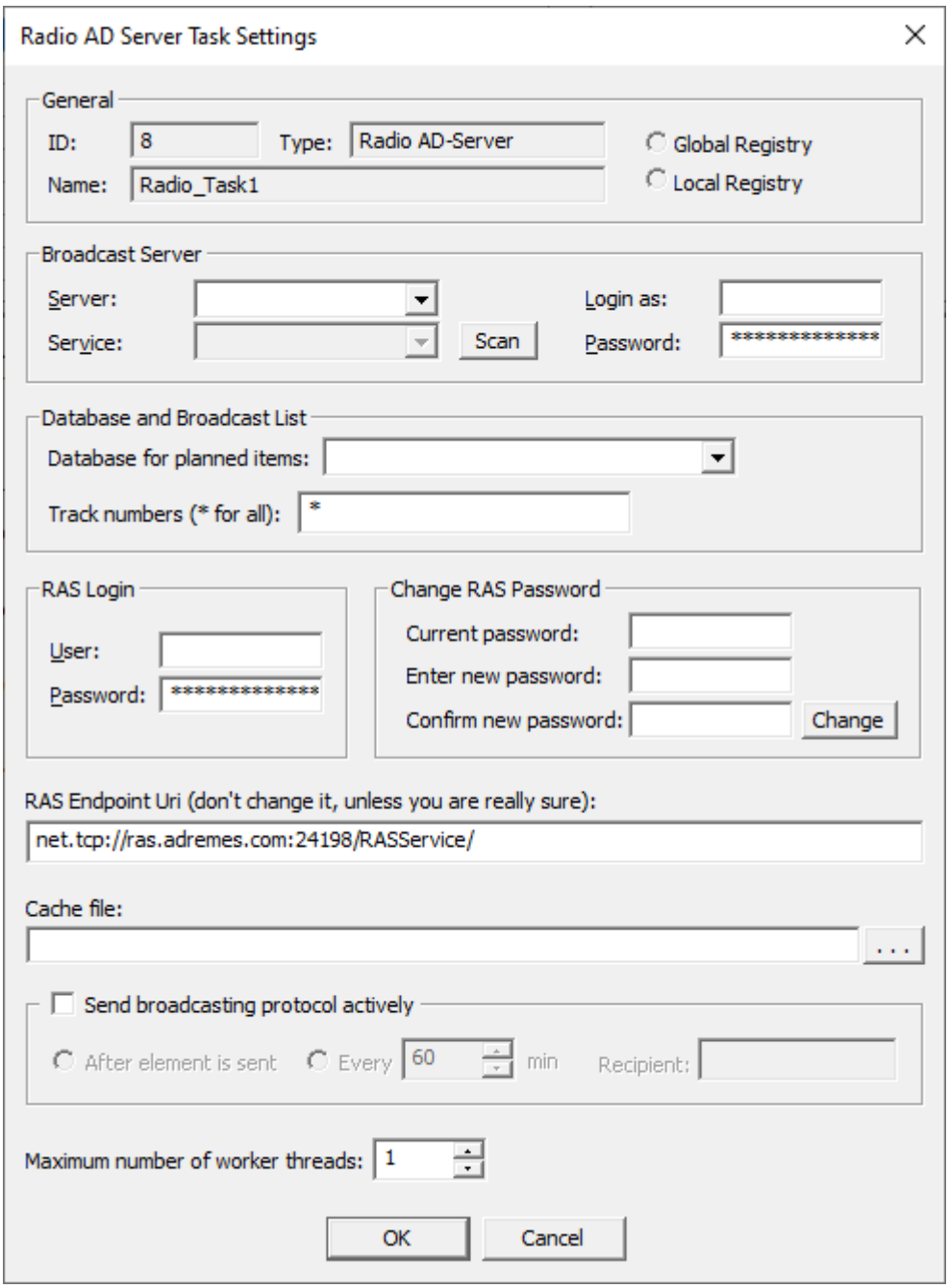
Than we have to configure the data for access the RAS system. In the RAS section of the settings dialog shown above we have a "Login as" field representing the login user used to log in RAS, as well as the password field for it. This data should be made available to you by Adremes and NOT by David. However it is essential to be filled. It is important to notice that at the moment you click the "Ok" button this acess data will be checked. That means, there will happen a try to connect with RAS system with the supplied Login and Password. If something goes wrong, it will not be possible to save anything set for this task. The set up only succeeds if the access data are correct. The connection will be established and finished, just to check the validity of the provided data.

Also in the RAS section we finally have the fields where we don't need to fill anything. The settings in "Change RAS password" block allow the password for the set user to be changed. In order for it to work, the user must be filled in the "Login as" field in "RAS" block on the left and the password data must be filled on the right as indicated in the dialog. Simply type the original password once and the new password twice and click on the "Change" button. Notice that there are many criteria which must be fulfilled for the password change so succeed. They are:

- It must have length 8-16 characters.
- It must have at least one uppercase & at least one lowercase characters.
- It must have at least one digit.
- It must have at least one special character.

If the new password fulfills all these criteria, the change succeeds and the user is notified. Else, an error message is popped up indicating it.

Still concerning the RAS configuration, there is the possibility to change the Endpoint Uri to which the conection is established. This field comes with a predefined value, which should only be changed in case Adremes or David Systems tells you to do so. Else, leave this field untouched with its default value.


Radio AD Server Task Settings
General
ID:
8
Type:
Radio AD-Server
Global Registry
Name:
Radio_Task1
Local Registry
Broadcast Server
Server:
Login as:
Service:
Scan
Password:
Database and Broadcast List
Database for planned items:
Track numbers (* for all):
*
RAS Login
User:
Password:
Change RAS Password
Current password:
Enter new password:
Confirm new password:
Change
RAS Endpoint Uri (don't change it, unless you are really sure):
net.tcp://ras.adremes.com:24198/RASService/
Cache file:
...
Send broadcasting protocol actively
After element is sent
Every
60
min
Recipient:
Maximum number of worker threads:
1
OK
Cancel

It is also mandatory to set where the cache file will be written. This is the file used to store all requests received from the cloud and to be processed by the task. When the task starts, this file will always be checked (in case it exists) for possibly pending changes to be processed (this is also good if for some reason the task aborts, the requests wouldn't be lost). It is important to notice that it request will be sent again and again as long as the system doesn't acknowledge its receipt. BUS only acknowledges the receipt after writing it to the cache file, which means that no request will be lost no matter what happens. This file tends to be empty in normal execution of BUS, since the requests are received and processed very quickly, so don't expected to monitor the contents at this file, else in error scenarios.
For the sending of broadcasting protocols this can happen on request or the user can choose to send it actively after elements are sent or every X minutes. For actively sending broadcasting protocols, the recipient of the protocol must also be provided. If this option is unchecked, broadcasting protocols will be sent only on request.

Last but not least, we may configure the maximum number of worker threads that may be used at a time by this task. This is directly related to the number of elements that may be processed at the same time. The higher this number is, the more elements will be processed at a time, which may be good if a good performant system is available. In this case changes are supposed to be processed quicker the higher this number is. It is important to notice that it is useless to set high values in this setting if the system is not able to cope with so many threads at a time or if not many cores are available. This number may be set empirically or simply left as 1, which means that each change will be processed at a time (also a safer scenario).

After setting the desired changes, click the Ok button and if all data is valid, the dialog is closed. The task is ready to be started.

Concerning the task processing, every time a change in broadcast list is required, a check will be performed to see whether this entry is already created in database. If it is not, it will be created with some essential data only. When receiving the media for a given planned broadcast list item, the same check will be performed and the same will be done. After receiving the media, the item is updated in broadcast list and the element in database will be referenced in the DbRef field.

7.19 Index task

Preliminary hint: there is another chapter below, 9 Full-text search within BroadcastSystem, which describes the search part of the system and a short description of how to start with ElasticSearch.

The index task is part of the new full-text search system based on ElasticSearch which replaces the old system based on dtSearch. This new solution is available since version 6.2 of BUS. The minimum required version of BCS is 6.2, too. The index task generates a full-text index for a whole BCS tree and all data in it and then updates it with any changes by monitoring all BCS notifications. Full-text search for clients is currently realized via ActiveX control BcsSearch.ocx for DigAIRange and BCSSearchService for any Web/custom clients to run queries on ElasticSearch via REST and Web Sockets so that you can play around with your own frontends (e.g. a web page).

In order to run the index task, you need the ElasticIndex.dll / ElasticIndex64.dll. These dlls should be located in the same directory as BUS / BUS64.

Only one BCS (but multiple services) can be indexed per task. If you want to index the content of multiple BCS servers, you need one index task per BCS. The index task can be very memory-consuming. For the data of 20 years of a full-time service this can easily be many 100 Mbytes. Therefore, it is recommended to run the index task as single task on an extra instance of BUS – at least for the 32-bit variant. You can also use the 64-bit variant of BUS. Then the memory is no longer a topic.

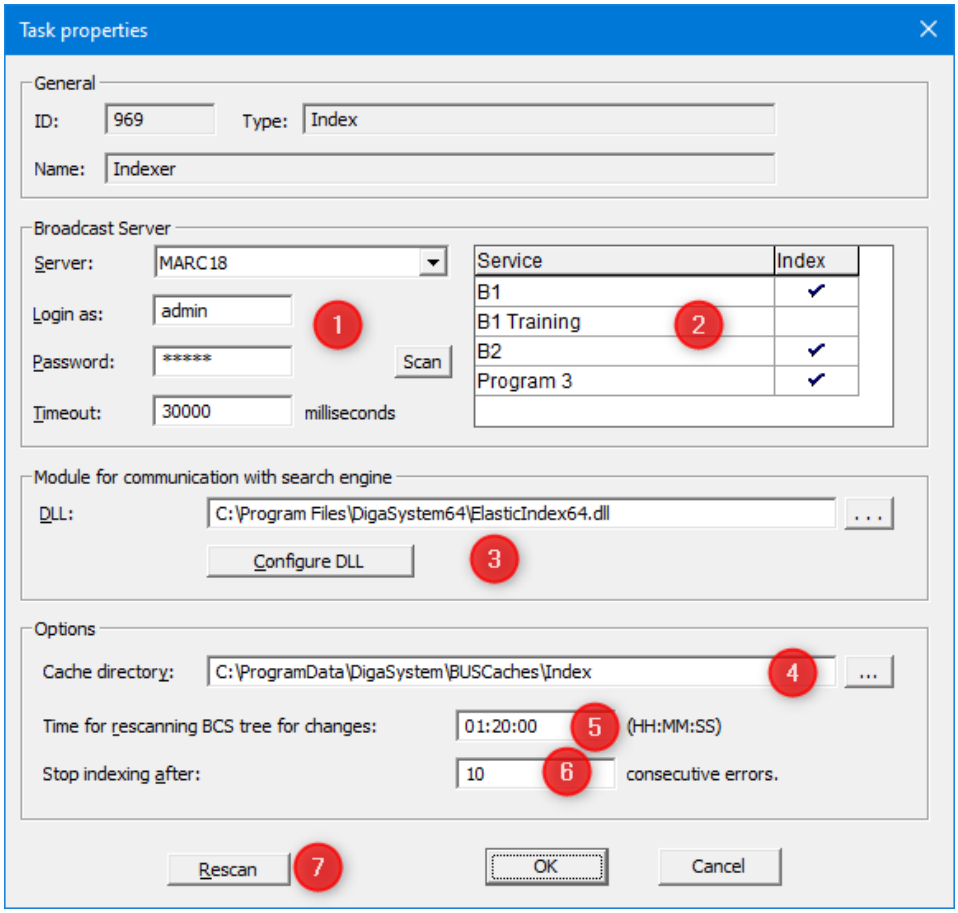
7.19.1 Index task stages

The index task runs in three stages:

1) **BCS crawling**. This means: the task queries the last change timestamp of all rundown nodes of the BCS tree to evaluate, which rundown needs to be indexed. (For checking what the rundown nodes are, please see chapter 2.1XML Data Structure / Overview.) This crawling needs the help of BCS, because BCS must provide a function, which evaluates the last change timestamp without permanently loading the corresponding node into memory, because this would make explode the memory consumption. The special function for querying the last timestamp is only available in BCS version 6.2 or newer.
   In the very first run the index task will treat all found rundowns as new. The information about the last change timestamp will then be stored in a binary file in the configured cache directory. In further runs the index task will only index new or changed rundowns. The crawling is the first stage when the task starts, and you can configure a time when the task should additionally do it each day (best in the night). The crawling done once per night is only some sort of “full run” for the unlikely case that data or a notification is lost. The crawling when the task starts allows to stop and restart the task, because the task will then catch up with all the changes, which were made while the task was not running. Don’t worry if the first crawling takes a very long time. Further crawling is optimized and is faster.
2) **Fully indexing**. This is the second stage when the task has finished crawling and knows, which rundown nodes need to be indexed. The task will handle rundown after rundown, starting with some around the current date and some special ones, like the jingles. In this stage all content of a rundown will be indexed, independent of the necessity for each individual node in the rundown. This will result in a slightly higher workload for ElasticSearch, but it’s no problem with multiple entries, because the indexing of a node will always overwrite any existing data for the same node. In addition, the task queries all existing nodes of the rundown on both sides – BCS and ElasticSearch – and removes the data for nodes in ElasticSearch, which do no longer exist on BCS side. Be warned that the very first full indexing is very time-consuming. You can expect something like 4 hours for a year of rundown data.
3) **Monitoring**. This is the third and the permanent stage for the index task. The task enters this stage after finishing stage 2 and does now handle all BCS notifications. For most notifications all the required data for updating the index is part of the data, which is sent by BCS. There is only an exception for deleting very big nodes, like deleting a full month or a year. Therefore, the task does not need to query any data from BCS for most notifications, which reduces the workload for BCS.
   Notifications, which were received during stage 1 or 2 are cached and are treated, too, when stage 3 starts. The caching is done in the configured cache directory within XML files. As the very first indexing can take a very long time (many hours or even days) no BCS notifications are handled during this time. When the first indexing has been finished, the task will start with phase (1) once again to catch up with all the changes, which happened since the start of the first indexing. For this and all further crawling/fully indexing the BCS notifications are cached as described.

7.19.2 Configuration

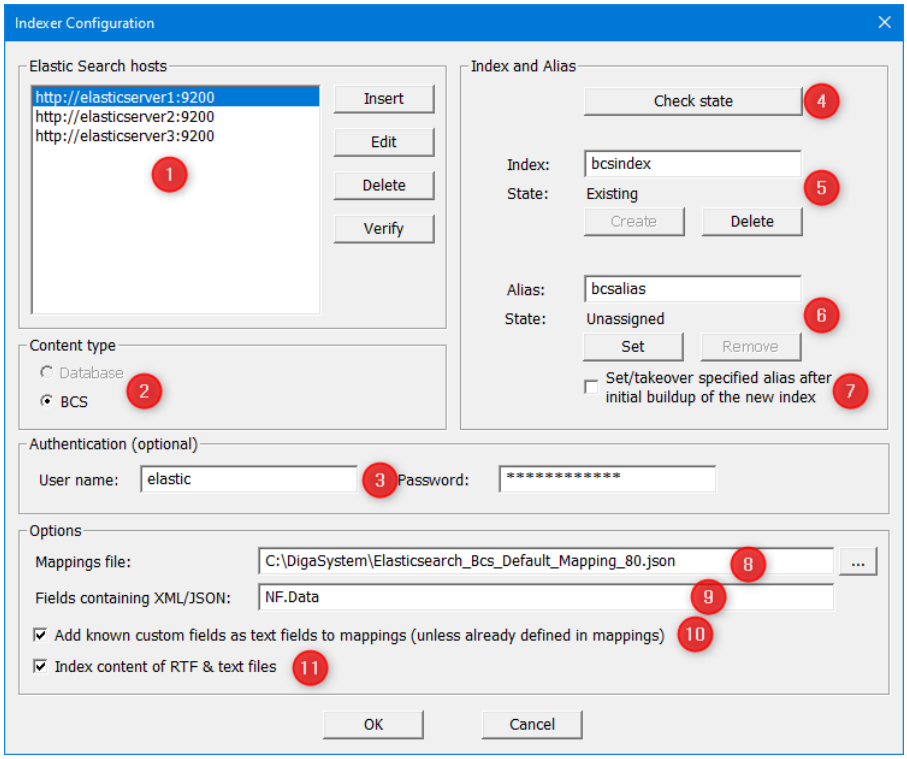
If you open the settings dialog of the task this window appears:



1) In the section “BroadcastServer” you can select one of the configured BCS servers to be used for indexing. Always provide credentials in “Login as” and “Password”. As timeout for the BCS connection something like 20 or 30 seconds should be o.k. If you press the button “Scan”, the list of programs/services is read from BCS and displayed in the box on the right-hand side.
2) Here you can select, which programs/services should be indexed. Global nodes, like the global jingles or the general pools are always indexed.
3) The index task does not communicate directly with ElasticSearch. This is done with the help of a “communication module”. The idea behind it is to be able to replace the communication module when a different search engine should be used in the future, without the need to rewrite the index task itself. Nevertheless, so far only one communication module for ElasticSearch does exist. It is the ElasticIndex.dll / ElasticIndex64.dll. Enter the filename for this dll. If you press the “Configure DLL” button, the configuration dialog of this dll appears.
4) The BUS index task needs a cache directory for storing the information about the last change timestamp of rundown nodes and for caching BCS notifications. This directory should not be used for anything else.
5) Here you can configure the daily time when the index task performs a BCS crawling and – if necessary – a full reindexing of changed rundowns. You should select a time when BCS is not busy – typically at a time in the night.
6) When the search engine/ElasticSearch continuously reports errors, it normally does not make sense to retry indexing again and again. Therefore, the task will stop the retries after some consecutive errors and will switch to state “incapable”. You can configure after how many consecutive errors this should happen. If you enter 0 the task will never switch to “incapable”.
7) The button “Rescan” will delete the file with the information about the last change timestamp of all rundown nodes. When the task starts the next time, it will act as being started for the very first time and will fully reindex the whole BCS content. Hint: this is not the same as removing the data in the full-text index. If you want to remove everything form the full-text index, you need to delete the index within the configuration dialog of the ElasticIndex.dll. **Important:** In order to fully rebuild an index, you have to do both actions!

7.19.2.1 Communication module

When clicking on “Configure DLL” button in the task dialog this dialog appears for the ElasticIndex.dll:



1) Here you can configure the URL for the ElasticSearch server. Use the protocols “http” or “https” accordingly as prefix of the host name. The default listen port of ElasticSearch is 9200. Normally, you enter only the URL of a single server. The dialog allows to configure a list of servers. This can be used for connecting to a cluster of ElasticSearch instances. The dll will randomly pick one of the addresses from the list in order to connect to a cluster. If this server is not available, another one from the list is tried and so on. This serves a load-balancing for the multiple nodes in a cluster but also gives safety against server failure. Hint: though each host/node within an ElasticSearch cluster can serve different roles and not all of them might be appropriate for doing a specific operation (like searching, indexing, etc.), it is possible to send a request to any of the nodes. They will relay the request to a node in the cluster, which has the correct role. This function is called "coordinating node". Every node is implicitly a coordinating node. Therefore, you can list all nodes of the cluster and do not have to filter the list for a specific role.
   If you press the “Verify” button, the dll tries to establish a connection like described above and tries to read the version number of the ElasticSearch server (cluster).
2) The content type is always “BCS”, nothing else is supported in this context.
3) It is possible to specify credentials for the login to ElasticSearch. This is optional and it is independent of whether you are using http or https, but it must match the requirements of your ElasticSearch server, of course. Hint: the dll performs a basic authentication if credentials are given.
4) Whenever you want to make any changes to the configured index and/or alias, you first need to press the button “Check state”. The dll then reads out information about all existing indexes and aliases from ElasticSearch and displays a small dialog with this information. In addition, the state of the buttons below (create/delete/set/remove) is updated, so that only sensible operations are available.
5) In this part of the dialog you can configure an index name, see the state of the index and have buttons to either create a new index or delete an existing one. Creating an index does not put any content into the index. It only creates the managing structures for holding data and – very important – defines rules how the content should be treated. Most of these rules can be configured in the mappings file (8). Deleting an index means to remove everything stored within and for the index. So be careful, you will lose your existing full-text data!
   The dll always uses the index name for write operations, meaning: writing full-text information into ElasticSearch. For reading the alias name is always used, see below. You should choose an index name, which is somehow related to the BCS, for which you are indexing the content.
   Important: when you want to recreate an existing index, you always have to do three actions: delete the existing index, create a new index and then press the “Rescan” button in the config dialog of BUS Index

task so that the task knows that the index is new and must be filled from scratch! Otherwise, you will end up with a new and empty index, but the index task will not fill it up again.

6) In this part of the dialog you can configure an alias name, see the state of the alias and have buttons to either set a new alias for an index or to remove an existing one. There is not much action going on when you assign an alias name to an index. But the alias name is important nevertheless, because the dll always uses the alias name for every read access to the index (meaning: a search request). This allows to recreate an index, while users are still using an old one, see the next paragraph below.

7) This option is used in conjunction with the re-creation of an index. See the next paragraph below. You can check this option instead of pressing the “Set” button to assign the configured alias name to the index. This operation is then caught up later when BUS Index task has completed the initial buildup of the content of the index.

8) Here you can specify the mappings file. This information is sent to ElasticSearch when the new index is being created. The file must be formatted in JSON and it contains some options and, most important, a list of most of the fields in BroadcastSystem and rules how each field must be treated. A set of these configuration files is delivered together with the ElasticIndex.dll – different ones for database content and for BCS content and one for each major version of ElasticSearch. If you run version 8.x of ElasticSearch you should use the mappings file “Elasticsearch_Bcs_Mapping_80.json”.

   Modifying the content of the mappings files is only recommended if you really know what you are doing! Always start with the provided files. A case when it becomes necessary to modify the used file is: you try to index your BCS content and you get error reports from ElasticSearch that nodes cannot be indexed because ElasticSearch cannot interpret the content of one or more fields correctly. E.g. ElasticSearch could be confused about dates or times it cannot interpret and you could then define the corresponding field to be a pure text field. Before you do changes, please check the documentation of mappings for ElasticSearch:
   https://www.elastic.co/guide/en/elasticsearch/reference/8.14/mapping.html

   The mappings file is only used during creation of an index. Modifying the mappings for an existing index is not supported by the ElasticIndex.dll. (Hint: ElasticSearch supports it to some extent, but if you want to try it you have to do it by directly sending the necessary commands to ElasticSearch via http and a corresponding client.) So, whenever you make changes to the mappings file, you need to recreate the index and then fully rebuild it !

   There is one important case when you need to edit the mappings file: if you use custom fields, which need to be indexed. By default, ElasticSearch stores the data of all fields, but they are not searchable in a full-text search. For the latter, you need to add the custom field to the mappings file and make ElasticSearch copy the content of the field to the virtual “__all” field. You can look for the entry for field “Title”, copy it to the end of the mappings file to the appropriate hierarchy in the JSON and then modify the field name to the name of the custom field. Attention: in BCS-JSON custom fields normally have two parts, which are separated by two underscores, like: “BR__SpecialField”.

   You might also encounter other errors being reported by ElasticSearch when you try to index your data for the first time. E.g. if you have a lot of custom fields, it might happen ElasticSearch reports that the number of supported fields is exceeded. This happens, because ElasticSearch will add a new field to the internal mappings whenever it encounters a new and unknown field name. The number of supported fields is configurable. You can have a look into the mapping files and see the line “"index.mapping.total_fields.limit": 2500” at the beginning. Anyhow, the value 2500, which we provide, is already a very big number and you should not increase it thoughtlessly. One thing you can do in that case is to use a test client for making REST calls and execute the command "GET /<our BCS index name>/_mapping”. ElasticSearch will return the internal mapping, which was built up while indexing all nodes and you can evaluate how many fields you have and how they are handled by ElasticSearch. From that you can learn how to construct a better mappings file.

   Be warned that the creators of ElasticSearch sometimes make fundamental changes even with a new minor version, which require an adjustment of the mappings or, worse, new code for handling things differently. E.g. in version 7 they had the “good” idea to remove the capability for doing a full-text search – which is a very astonishing change for a full-text search engine. Normally, such a step would have made this search engine unusable. Fortunately, we found a workaround. You might see the instruction "copy_to": "__all" for many fields in the mappings file for version 7 and 8. The idea behind this is to copy the content of most fields into a special virtual field, called “__all”. This increases the amount of data, which must be stored in the index, of course, but it makes it possible to search within this field when a full-text search is wanted. When this paragraph was written (2024-09-01) tests with ElasticSearch 8.6 – 8.13 were made. For newer versions it cannot be guaranteed that they will be suitable.

9) Here you can specify fieldnames as a comma-separated list of fields, which can contain data, which is formatted as XML or JSON. This might be the case if someone tries to store a hierarchy of additional fields in a single custom field. (This is, by the way, a bad idea. Individual values should be stored in individual fields in BCS and if you want to create a hierarchy then you can use e.g. the dot ‘.’ to separate the levels.) The problem with this type of data is that it might be impossible to build up a valid JSON for the overall

node data. ElasticSearch will then reject to index such a node. If you specify such a field here, the dll will remove all formatting characters, like '<', '>', '{' or '}' from the data and replace these characters by a space. The resulting string should be indexable and should contain all useful words from the original data.

10) As described above in (8), you normally have to add the custom fields you are using as text fields to the mappings file in order to prevent errors from ElasticSearch when indexing nodes. The dll can add known custom fields automatically to the mappings when an index is being created and when you have checked this option. The only drawback is that in DigaSystem custom fields are only defined in the database and in BCS mode the dll does not have access to any database. Therefore, the dll can only read out custom fields, which have been defined in the registry. The dll evaluates the two places:
Common\CustomFields
DBM\CustomSearch
In addition the dll adds the name of custom fields, which have been defined by DAVID. These are the fields "DAVID__NOSTOP", "DAVID__DistributionEndpoints", "EPS__Consistency", "SYSID__SID", "SPEECHTOTEXT__PLAIN", "LOUDNESS__..." and "MUSIC__...".

11) By default, the content of RTF and text files is not added to the index. If you check this option, the dll will try to read the text of specified presentation files and add it in the field "Presentation" to the JSON, which is sent to ElasticSearch.

7.19.2.2 Seamless re-creation of an index

Initially filling a new index can take a very long time. You can expect many hours and for really big BCS trees more than a day. This imposes the problem what to do when you want to re-create an index – for whatever reason. If you first delete an existing index and then create a new one and start filling it, the users will not be able to run searches (or at least only on a very small fraction of the already indexed content). ElasticSearch and DigaBroadcastSystem have a solution, which allows to re-create an index while the users can go on using an existing index, followed by a seamless switchover.

The feature of ElasticSearch, which is being used here, is the alias for an index. The ElasticIndex.dll always uses the index name when doing write operations (indexing content) and the alias name when doing read operations (search requests). So, the idea is to continue using an existing index with its alias name for searching and at the same time build up a new index in the background and to re-assign the alias name to the new index once the initial filling of the new index is completed. You have to do the following steps:

1) You have one BUS Index task, which you keep running to continuously update the existing index. This is the index, which is being used to handle search requests.
2) You configure a second BUS Index task. It should be configured to index the same content as the first task.
3) In the settings dialog of the ElasticIndex.dll you specify a different index name but the same alias name as for the first task. You do NOT assign this alias to the new index. Instead, you check the option "Set/takeover specified alias after initial buildup of the new index". This delays the switching of the alias until the new index has completely been filled and it is ready to be used.
4) You start the second Index task and let it run in parallel to the first one – probably for a very long time. When the task has finished the initial indexing of all the BCS content, it will trigger the takeover of the alias name, meaning: the alias is unassigned from the old index and assigned to the new index. From this moment on, all search requests will go to the new index.
5) As the old Index task is still running but no one is going to use the old index anymore, you should stop this task and probably you will go to the settings and delete the old index.

8 DIGAIRANGE

This chapter is not a complete manual of DigAIRange, it describes only the parts which can be used for the administration of the Broadcast System! You should check out the user's manual first.

Please note that from DigAIRange version 5.2 onwards, there are additional prerequisites for a correct installation, described at the end of the Section 1.2.4.

8.1 Command line arguments (case insensitive)

DigAIRange 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 "DigAIRange". You can specify the configuration after a space, a colon or an equal sign, e.g. -c=DigAIRange_B3
-mi	<none>	Multi-instance switch. Must be used in order to start more than a single instance of DigAIRange.
-nsw	<none>	No startup window. Suppresses the startup window.

8.2 Configurations

When DigAIRange starts a configuration can be chosen. A configuration is the whole set of parameters and settings which are loaded from the DigaSystem registry. Each configuration is stored within a single top-level registry key (also the configuration name). The default key is "DigAIRange". It is recommended to name further keys "DigAIRange_xyz" so they stay together. It is not possible to have a subkey of DigAIRange in multiple configurations, e.g. DigAIRange\Import1 and DigAIRange\Import2. 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:

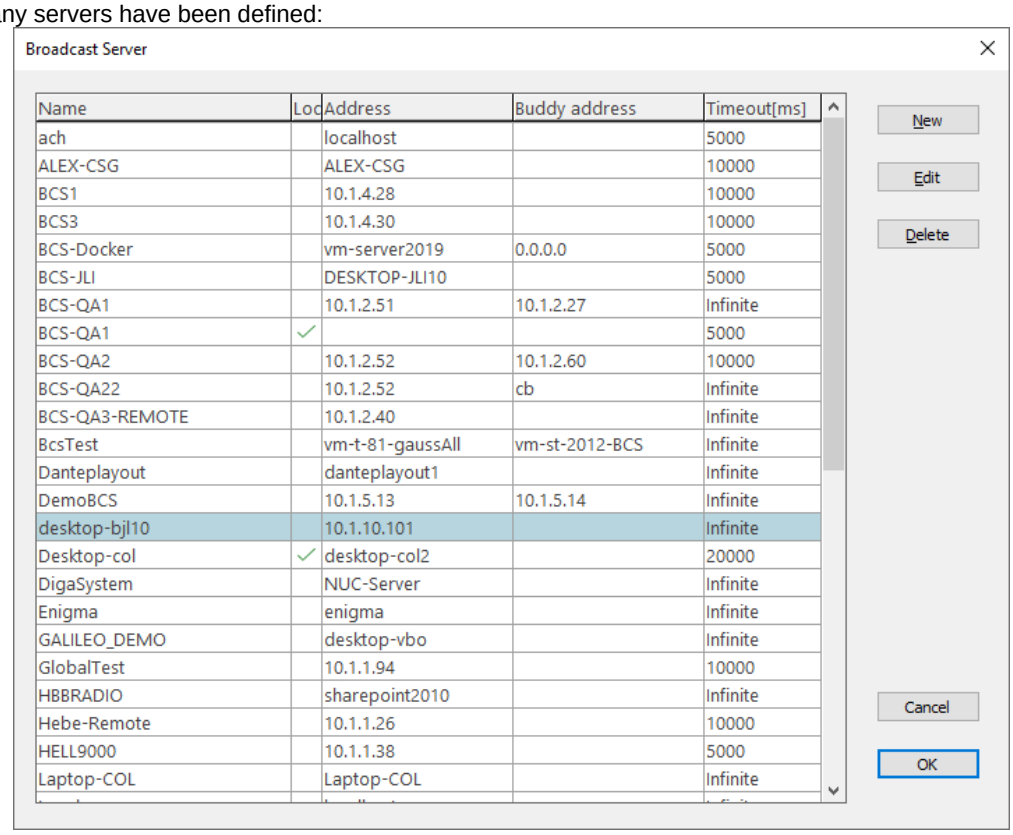
- If there is only the "DigAIRange" key, this configuration is started.
- You can specifiy a command line parameter c to select a configuration. Examples: "-c DigAIRange_B3" or "/C:DigAIRange_Training". Of course, a top-level key with the specified name must exist, otherwise DigAIRange would load default settings.
- DigAIRange 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 selectable configuration in the key " DigAIRange \Configurations". (Hint: This is the single parameter which cannot be moved to a different top-level key, because DigAIRange must have something to start with.) In this key you can list the configurations as sub-keys, for example:
    - DigAIRange = Standard settings
    - DigAIRange _Training = Special settings for training purposes
    - DigAIRange _Youth = Settings for the youth service

  Below each sub-key you have settings for the name, which is displayed to the user in the selection dialog, filters, which limit the availability to certain computers and values, which define default configurations. The filters for computers can help if you want to maintain the configurations within the global registry. For more information about the possible parameters, please see the parameter documentation in DigParam.rtf.

Be aware that a key name for a configuration replaces the key name "DigAIRange" for every registry access of DigAIRange: for reading and writing, for the global, the local and the user registry. It DOES NOT have an effect for other key names (e.g. "BroadcastSystem") or other modules, like dlls or ocx, loaded by DigAIRange. For other modules this is only a theoretical topic, because all used modules, like MultiRec, CFM, OTM, import/export modules, do not read from or write to the registry themselves. This is always done via host process, which is DigAIRange in this case.

8.3 Server Settings

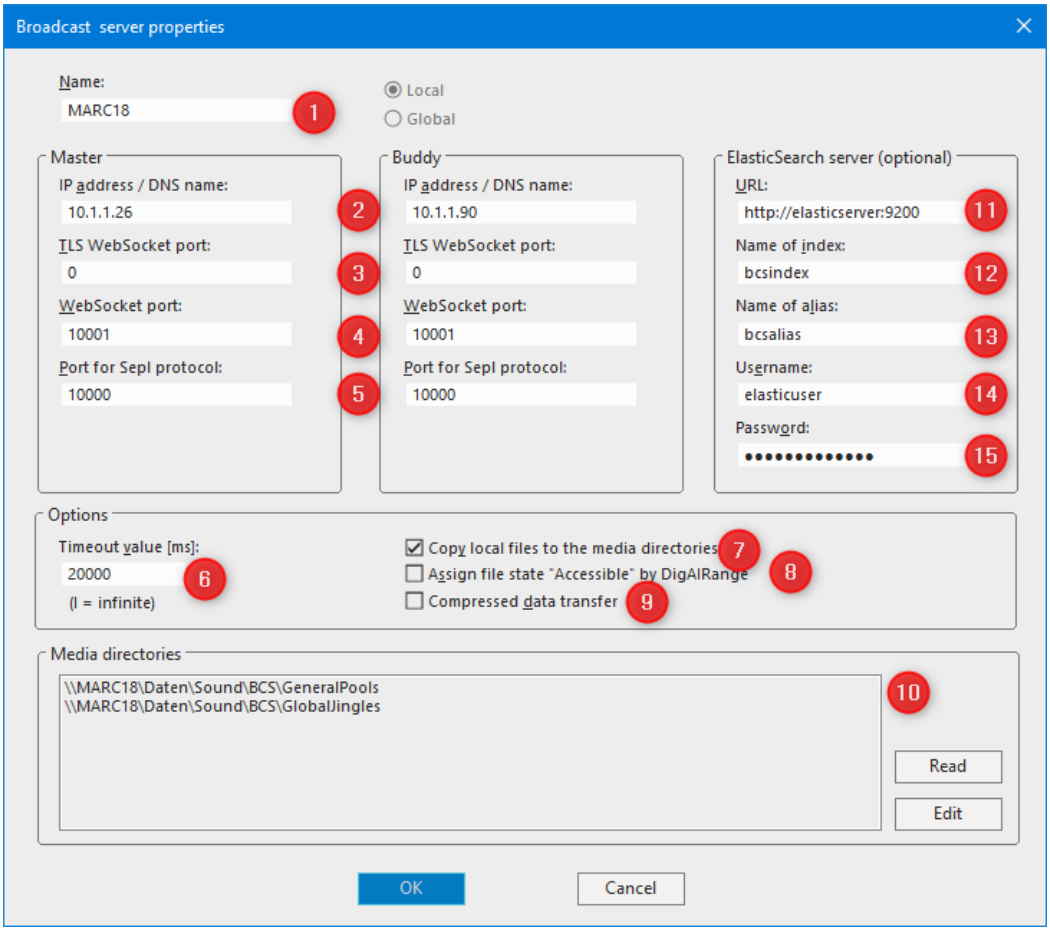
When you start DigAIRange for the first time you have to create registry settings for each Broadcast Server in your system. This can be done with the Admin program, but there is a dialog for this in DigAIRange too. Call the menu Program / Server properties which is only available if you are logged in as "ADMIN". A dialog with the list of all defined servers appears. At the first time this list will be empty, of course, but here is an example how it could look like if many servers have been defined:



Name	Loc	Address	Buddy address	Timeout[ms]
ach		localhost		5000
ALEX-CSG		ALEX-CSG		10000
BCS1		10.1.4.28		10000
BCS3		10.1.4.30		10000
BCS-Docker		vm-server2019	0.0.0.0	5000
BCS-JLI		DESKTOP-JLI10		5000
BCS-QA1		10.1.2.51	10.1.2.27	Infinite
BCS-QA1	✓			5000
BCS-QA2		10.1.2.52	10.1.2.60	10000
BCS-QA22		10.1.2.52	cb	Infinite
BCS-QA3-REMOTE		10.1.2.40		Infinite
BcsTest		vm-t-81-gaussAll	vm-st-2012-BCS	Infinite
Danteplayout		danteplayout1		Infinite
DemoBCS		10.1.5.13	10.1.5.14	Infinite
desktop-bjl10		10.1.10.101		Infinite
Desktop-col	✓	desktop-col2		20000
DigaSystem		NUC-Server		Infinite
Enigma		enigma		Infinite
GALILEO_DEMO		desktop-vbo		Infinite
GlobalTest		10.1.1.94		10000
HBBRADIO		sharepoint2010		Infinite
Hebe-Remote		10.1.1.26		10000
HELL9000		10.1.1.38		5000
Laptop-COL		Laptop-COL		Infinite

The first column shows the name of the server as it is displayed within DigAIRange. This need not be equal to the DNS (or NetBIOS) name of the server. The second column is checked if the server definition is within the local DigaSystem registry. The next two columns are the addresses of master and buddy server. If there is no buddy server, the column is empty. The last column lists the timeout value for TCP/IP connections to the server. With the buttons you can create new registry entries and edit or delete existing ones.

For editing a single entry, the **next dialog** (see below) is used: (Hint: the screenshot shows the dialog of a DigAIRange version 6.2 or newer. Older versions have a reduced dialog without WebSocket parameters.)



1) The name and the registry location can only be set if you create a new entry, otherwise the fields are disabled.

   The IP settings for the BCS servers is seperated into "Master" and "Buddy", which can form a cluster. If you have only a single BCS and no cluster, fill the "Master" part and leave "Buddy" empty.

2) The address field can contain either an IPv4-address in format "nnn.nnn.nnn.nnn" or a DNS name like "localhost". If you use WebSocket communication only, it is also possible to enter an IPv6 address.

   The three port settings have a fixed priority. If you enter a TLS WebSocket port, it will be used first. Only if this value is empty (0), the WebSocket port is used. Only if this value is empty (0), the SeplProtocol port is used. The clients can fallback from master to buddy if the master is not available and vice-versa. But there is no fallback to other protocols. Therefore, use only a protocol, which is offered by the BCS the client is addressing.

3) The TCP port for a communication via WebSockets and secured by TLS.

4) The TCP port for a communication via WebSockets in plain text.

5) The TCP port for a communication via SeplProtocol (this is the old one and the only one supported by pre-version 6 modules). By default, the port number is 10000.

   The options group contains four parameters:

6) The timeout value is specified in milliseconds and applies to the TCP/IP communication with this server. If a server does not respond within the specified time an error message appears and the connection is declared as disturbed. The message is displayed to inform the user about the communication or server problem, but DigAIRange will try to reconnect automatically to the server (or its buddy) with the next operation.

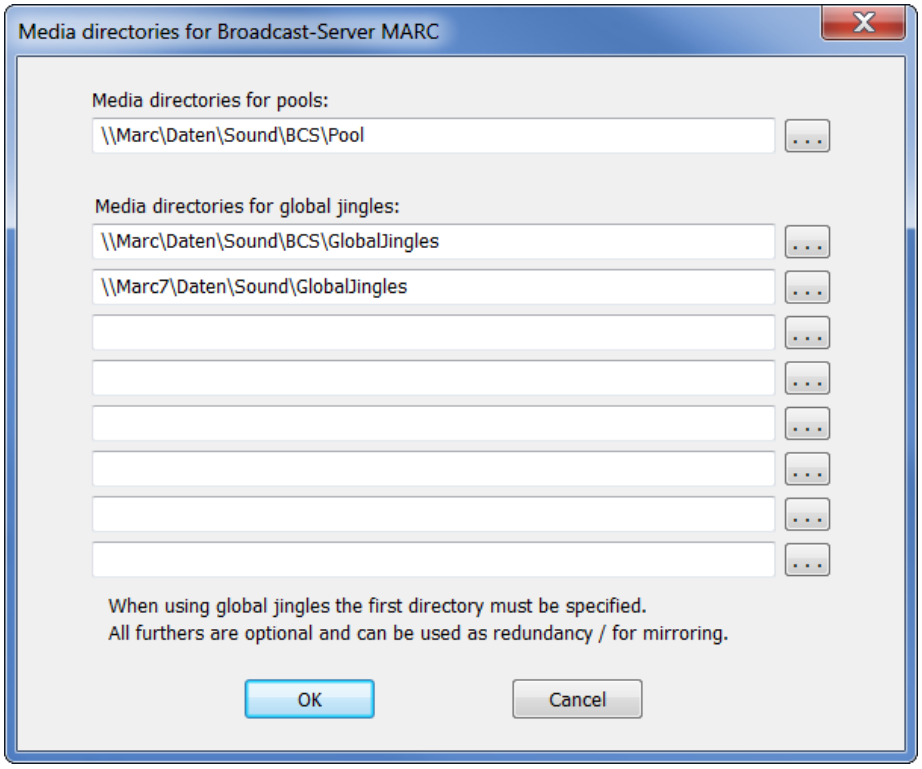
7) The two top checkboxes are only of interest if you have a standalone computer, e.g. a laptop. If you have read the chapter about the file management, you know that DigAIRange will make copies of media files sometimes. One case is when new schedule elements are created which reference a file on the local computer. As such a file normally is not visible from other computers, it is automatically copied to one of the media directories. In a multi-computer system these media directories are on a fileserver and this operation is useful. But on a standalone computer it is simply unnecessary, so you should uncheck the first checkbox.

8) The second checkbox is similar. In a multi-computer environment, you have always a Broadcast Utility Server running which checks the media files. This is probably not true on a standalone computer. But without BUS the media files are not checked, and the highest possible file state is "Existing" which is displayed with a big question mark in DigAIRange. If you find this disturbing or if your OnAir client needs the file state to be "Accessible", check the second checkbox. Attention: this does not mean, DigAIRange will perform a file check, it simply sets this file state independent of the file content !

9) The option "Compressed data transfer" will enable a compressing algorithm for the BCS communication. This is always useful in case the client has a slow connection to the BCS. But as it also requires a lot of processor time (especially on the BCS when many clients are using it) check out if this option really improves anything in a fast LAN environment. This option is available for the old SeplProtocol but also for a WebSocket-based communication.

10) The "Media directories" group is for defining the media directory for the general and personal pools and the global jingles. As explained in the chapter about file management these nodes are defined outside of a service. Therefore, you cannot use the manage dialog for a service to define the corresponding directory. You have to connect to the server to read or change this setting which is stored in an XML node for the whole server. Perform an admin login and set the directories as you like.

    If you press the "Edit" button of the media directories this dialog appears:



    The parameters in “ElasticSearch server” are all optional and are only needed if you have an ElasticSearch server fort the full-text search running. (Please see chapter 9 for more information about full-text search.)

11) This is the URL for connecting to ElasticSearch. Specify the protocol (http or https), the dns name of the server and a port number (9200 is the default for ElasticSearch). E.g: http://elasticserver:9200

12) Specify the name of the index, in which the BCS data is indexed.

13) Like before, but here the name of the alias is needed.

14) If you have configured ElasticSearch to require an authentication, you need to specify a valid username for ElasticSearch in this field. (Hint: this is not a DigaSystem user. ElasticSearch has its own management of usernames and passwords.)

15) If an authentication is required by ElasticSearch specify the password in this field.

8.4 User login

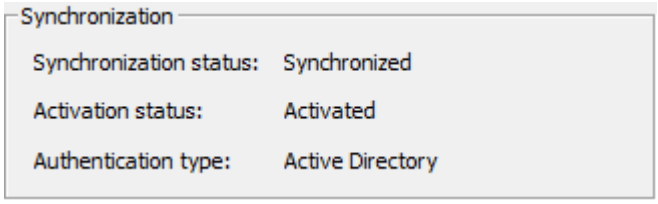
Technically, we need to distinguish two things: first, DigAIRange is a DigaSystem desktop application and therefore performs a login at the DigaSystem user management implemented by DigaSQL.dll. Second, DigAIRange can connect to multiple different servers/services, which are: BCS client, BCS remote GUI or BUS remote GUI. For all these connections, a login to the server/service is required.

8.4.1 DigaSystem login

During startup, DigAIRange will perform the login at the DigaSQL.dll. This can happen in four different ways:

1) Login dialog: DigAIRange will display the user a dialog to enter his credentials – the name and the password. The user must be known as a valid DigaSystem user as you can manage them in Admin.exe. This type of login is also done for a user change, if the user presses the login button in the main toolbar.
2) Shared login: Most DigaSystem desktop programs can share the user. This is applied, if such a program is started and there is already another one running, which made a DigaSystem login. For example, if you first start DBM.exe and perform a login, then start DigAIRange, DigAIRange will automatically share the same user. If the shared user is changed later in one of the affected programs, all others are automatically changing the user, too.
3) DigaSystem auto login: In the main settings dialog of DigAIRange on tab “Local” you can enter the name of a DigaSystem user, which should be logged in during startup. This user must have the right for an auto-login, which is the action right “AutoLogin”. A password check is not done in this case. A shared login is preferred.
4) Windows auto login: In the main settings dialog of DigAIRange on tab “Local” you can select the radio button “Windows”. (Hint: the user name is the user currently logged on to the Windows session and is only displayed for information, not to be changed.) During startup, DigAIRange checks, whether the current Windows user exists in the DigaSystem user list, too, and if yes, this user is logged in. Again, this user must have the right for an auto-login, which is the action right “AutoLogin”. A password check is not done in this case. A shared login is preferred.

It is also possible to login with the credentials of an ActiveDirectory user. Therefore, the AD users must have been replicated / synchronized with the DigaSystem users. In Admin.exe the properties of the user will look like this:



For an ActiveDirectory user, login types (1) or (4) apply. With method (1) DigaSQL will relay the credentials to Windows, which then authenticates the user via ActiveDirectory. With method (4) no password is checked and therefore the login can succeed in the same way as for a local user account.

8.4.2 Login to BCS and BUS

A BCS client needs to perform a login. The classical way is that DigAIRange sends the credentials (user name and password) to BCS. DigAIRange always uses the credentials of its DigaSystem login. With login method (1) DigAIRange knows the password. For the other login methods, the password can be queried from DigaSQL. (Hint: DigaSQL allows this only for the current user, not for others.)

When WebSockets are used for the communication with BCS then an HMAC (key-hash bases message authentication) method is used for the login. With this method the password is never transferred via network.

For an ActiveDirectory user the situation is different. In this case, the DigaSQL.dll does not store the password and via ActiveDirectory it cannot be queried. BCS supports a special login method for AD users. This method uses a HMAC method, too, where the key is a combination of user information and the password of user “ADMIN”. This method of login has the following requirements:

- DigAIRange version 5.7.776.0 or newer,
- BCS version 5.7.364.0 or newer,
- DigaSQL.dll version 3.14.3468.0 or newer on DigAIRange and BCS side,
- the BCS connection parameters in the DigaSystem registry in key “Digas\PlanServer\...” must not specify a certain user and password,
- user logged on to Windows must be an ActiveDirectory user being synchronized to DigaSystem,
- client computer and BCS computer must both be members of the same AD domain,

- password of user “ADMIN” must be identical for DigAIRange and BCS computer.

DigAIRange will first try to log in as the AD user but if this fails, a login dialog will be displayed and a regular login with username and password is done. Whenever DigAIRange has queried the password from the current user, a regular/classical login is preferred afterwards. Therefore, the AD login applies only, if DigAIRange was started with a shared login or a Windows auto-login and the user did not enter his password yet.

In addition to the mentioned login methods, it is possible to specify a user name and a password in the DigaSystem registry for each BCS. This is done in the key “Digas\PlanServer\<BCS name>” in values “User” and “Password”. If these parameters are present, they are preferred against an individual user login.

The login method "HMAC" requires like the "AD" method that the password for the user "ADMIN" is identical on the client computer and on BCS. Typically, this is guaranteed if both computers use the same DigaSystem registry (Global.par/Rights.par). If this is not the case, the client will not be able to login with the HMAC method. Therefore, it is possible to specify a different login method for a BCS connection in the registry parameter "Digas\PlanServer\<server name>\LoginMethod". It can be: "HMAC", "AD", "RC5" and "AUTO". The type "RC5" means the old/classical method of BCS clients to crypt the password with the RC5 algorithm and a hard-coded key. This method works even if the registries and/or the ADMIN passwords are not identical on both sides. The default value is "AUTO", which means: for a WebSocket connection "HMAC" is used by default, otherwise "RC5" and for an ActiveDirectory-synced user an AD login is tried first. If this does not work a fallback to one of the other methods is done.

The same possibilities for a login apply when DigAIRange connects as a remote BCS-GUI to a BCS, too.

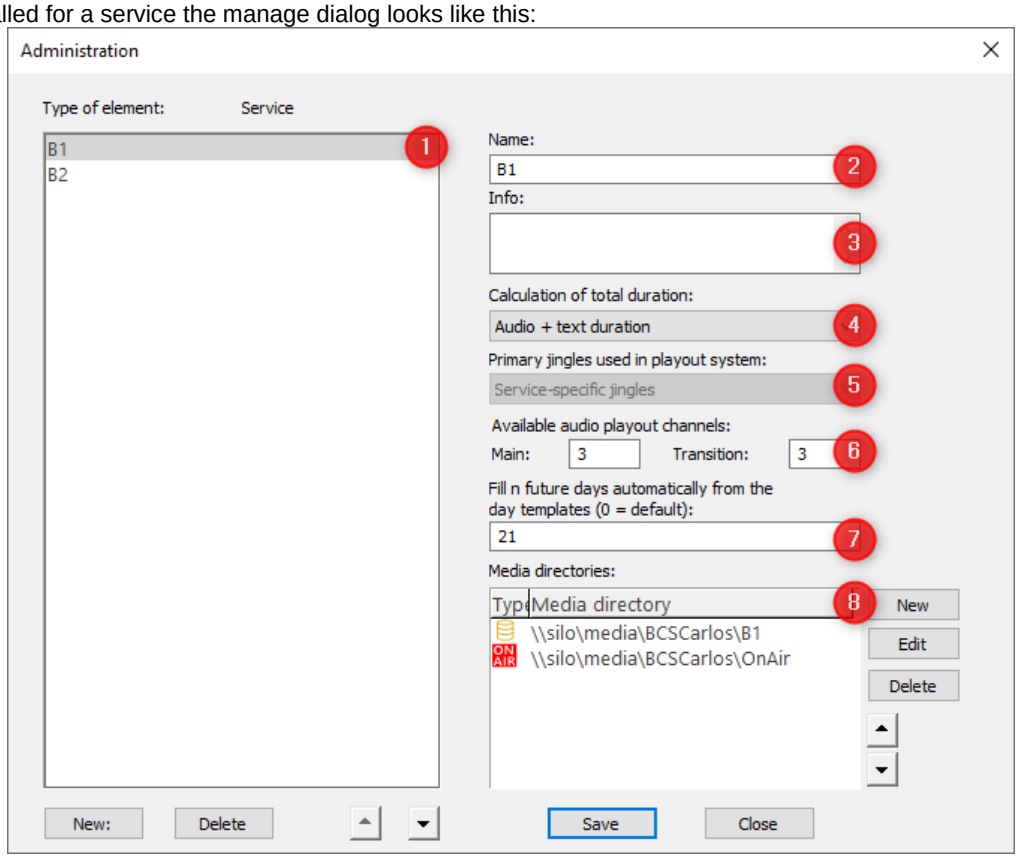
If DigAIRange connects as a remote BUS-GUI to a BUS, the current DigaSystem user is used. The method used for login can be configured, too. In this case this is done in parameter "Digas\BUS\<BUS name>\LoginMethod". The default value "AUTO" will be resolved to "AD" when the current user is synced with ActiveDirectory and "HMAC" otherwise. Like for a BCS connection, it is possible to set the parameter to "RC5" if BUS and the GUI are running in different AD domains and/or use different DigaSystem registries. In the latter case a login is only possible if the password is known to DigAIRange (an AD login is not possible).

8.5 Manage Dialog

The manage dialog is available for many items in the tree view. You can select the main menu item Tree / Manage or the Manage item in the context menu of the tree. Calling this dialog is only possible if you are either "ADMIN" or if you have the manage right for the service. The second statement is only true for all sub items of the service. Items which are not inside a service (like the import, export or a server) can only be managed by the supervisor. The fields of the dialog depend on the type of the tree item.

8.5.1 Manage Services

When called for a service the manage dialog looks like this:



1) On the left side you can see the list of defined services. With the buttons below the list you can create new services and delete existing ones. Moving the elements of the list is available for elements of other type only. A service must not be moved and the Broadcast Server will prevent this operation. If you select an element of the list its properties are displayed on the right-hand side.

2) The first field is the name of the service as it is displayed to the user. This name must also be used to define the right objects (see the chapter about the rights for more information). DigAIRange will automatically create or rename the right objects when you enter a new service name.

3) The info field is not used so far, you can enter what you like.

4) In the next combo box, you can define the way the total duration is calculated by the BroadcastServer for this service. You can select between “Only audio duration” or “Audio + text duration”.

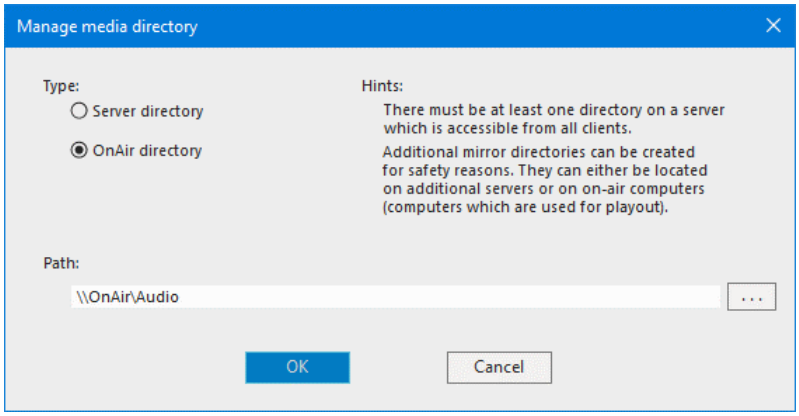
5) Here you choose which jingle node is used as primary jingles, the global jingles node or the jingles within the service. "Primary" means two things: First, TurboPlayer will load only jingles and jingle groups from this node into its "Jingles" window. Other windows (stack rundowns) can be filled with all nodes but the "Jingles" window has a special look&feel optimized for jingles. Second, a jingle group can be loaded automatically when a new group or show in the main rundown becomes the next one to be played. This is only possible with the primary jingles.

6) Here you define how many channels are available in your on-air studios for the playout of main rundown elements and for transition elements. The main rundown elements are the regular elements, like music or commercials. Transition elements are short elements during a transition / crossfade of main rundown elements. This can be: voice-over elements, jingles, drop-ins,… These transition elements can be mixed down on a single output line / fader on the mixer console. The two values are evaluated when the user schedules transitions with

OnAIR-TrackMixer: the user can be warned if he schedules too many elements which cannot be played due to an insufficient number of channels. If you do not have transitions channels leave the values empty. If your studios have a different number of transition channels use the values for which you want to get a warning during scheduling.

7) In DigaBroadcastSystem you can define day templates which are automatically inserted into a "real" day whenever a user requests the day for the first time. This is the default behavior, which is chosen by the default setting 0. If you enter a positive number BCS will always copy the appropriate day template for <n> days into the future. (Hint: A more specific usage limitation of the day templates can be found in the manage dialog for these templates.)

8) The media directories are explained in detail in the chapter about file management. One of the media directories must always be specified otherwise the broadcast system will not work properly. You can define up to 9 additional directories. This is done to have a redundancy and therefore the directories should be located on different fileservers. If you press the “New” or “Edit” button this dialog appears:



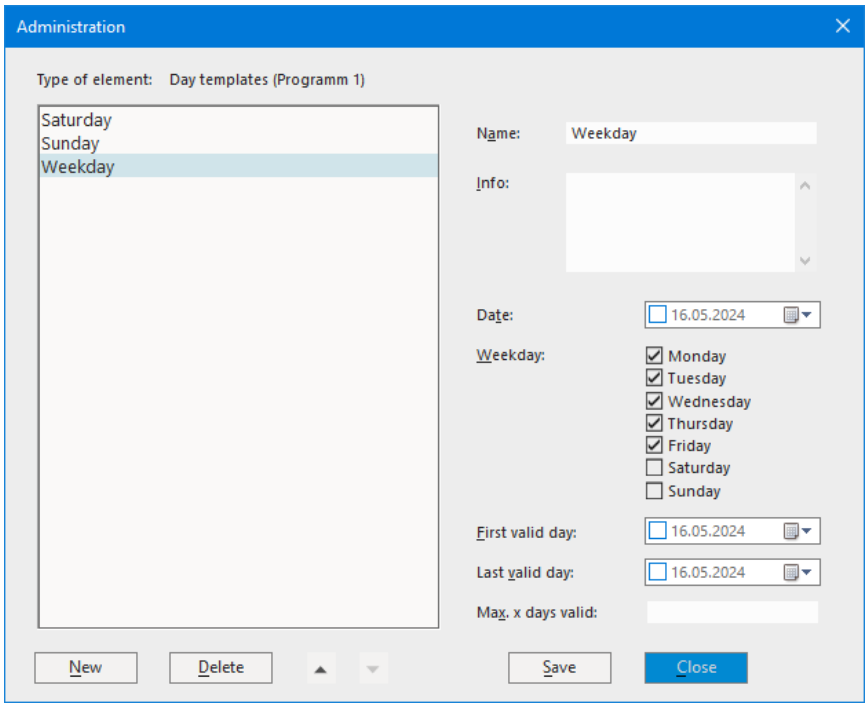
One hint about media directories: in principle all media directories are treated equally. Exception: if a directory is marked as OnAir directory it is treated different, see the chapter about file management for more information. Beside this way to define media directories, there is also the possibility to assign one additional media directory to each show. This can be useful if your shows are sent from individual studios typically and you have a lot of on-air media directories as a consequence.

If you use the manage dialog to create a new service, DigAIRange will ask you, whether you want to create all right objects for the new service, too. If you choose yes, you will find the newly created objects in DigaSystem Admin in “Objects\Showplan” and you can grant any rights to your users. Please see the chapter 1.5 about right objects for more information about which objects and rights exist.

Attention: BCS has an internal handling of right objects, which was optimized in BCS version 5.7.365.0. It is implemented as a caching of all existing right objects. If you create a new service it can happen that a client already tries to access this service and that the objects existing at this moment are evaluated by BCS. If DigAIRange (or you yourself) creates the missing objects later, BCS will not have any knowledge about them. In this case, you can call the BCS function “Reload settings” from the main menu to make BCS scan for new objects. This avoids a restart of BCS.

8.5.2 Manage Day Templates

The next thing to do when installing a new system is to create templates for days. The idea of a template is that you do not have to define a show structure for all days in the future. Instead, you define only some templates which will be used automatically: When a client tries to open a day, which has never been opened before, the Broadcast Server checks whether there is a template which is valid for this day and inserts it before answering to the client. Such a template should contain shows, groups and placeholders to reflect the format of your service. Now let's look at the dialog to discuss its parameters:



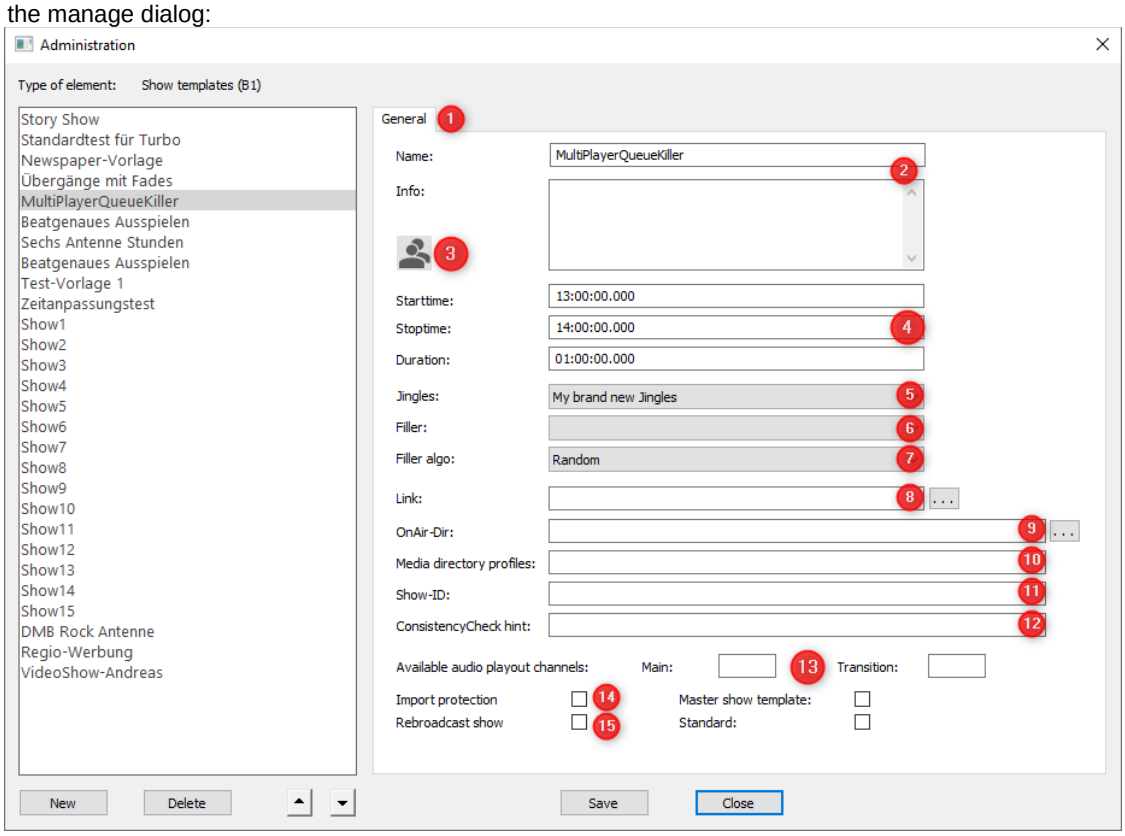
There are multiple options to define for which days a day-template is valid. The first way is to specify an exact date, e.g. for Christmas. The usefulness of this function is limited because such a template will be used only once and therefore you can also fill the day directly. The second way is to specify one or multiple weekdays when the template should be used, e.g. every Monday. Use only one of both ways because an exact date has a higher priority than a weekday.

With the two fields: first and last valid day you can define a date range in which the template will be used. It is possible to use only one of both values (or even none) which means the template is not limited in this direction. If you do not limit the validness in this way, you can make use of the last field too. It restricts the usage of the template x days into the future. This field does not mean: x days from the day when you enter the value in the dialog but instead it means: x days from the moment when someone tries to open a day. For example, in the dialog above the value is 14 days. Therefore, a user can always open days up to 14 days in the future. But if he tries to open a day which is even more in the future, the day will simply be empty and he cannot put any new elements into it (of course only if he is a normal user without manage right).

The name field is simply the name of the template and you can enter whatever you like, but of course, it is never wrong to use a sensible name like the weekday itself. The info field is not used so far.

8.5.3 Manage Shows, Show Templates and Day Templates

If your focus in the tree is on a day or a day template or a show template you can manage the list of defined shows with the manage dialog:

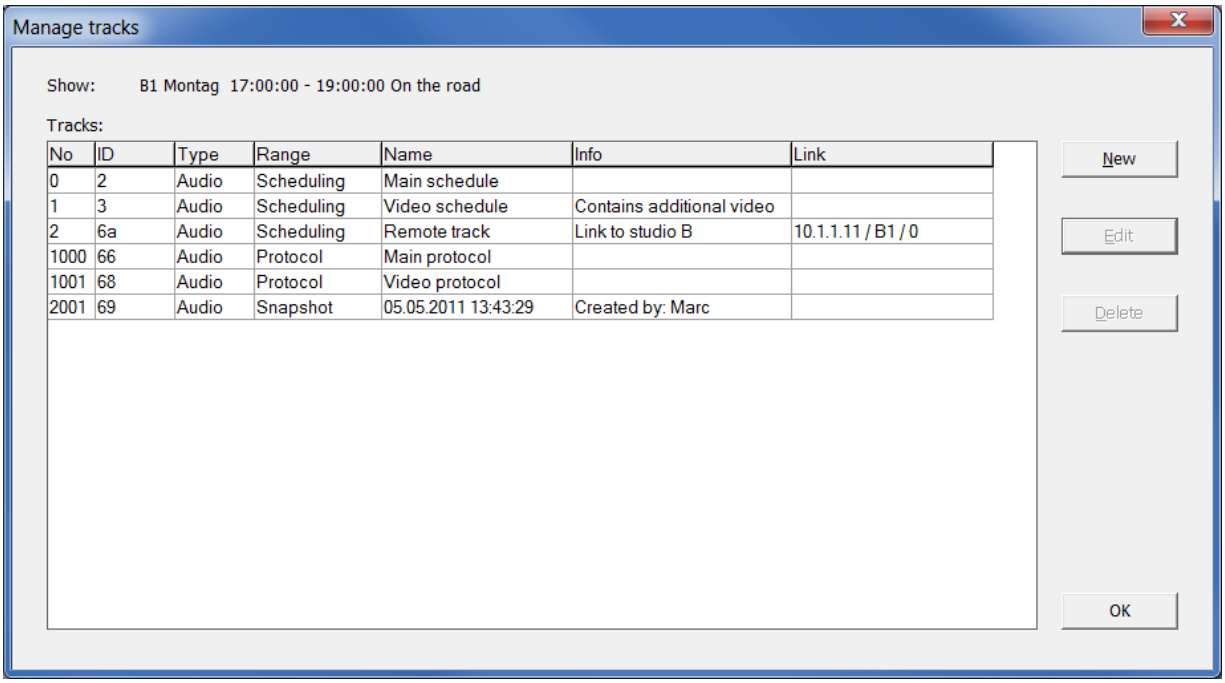


1) Multiple tabs are only available if you have show user masks installed and activated for the current user. For the installation, see “DigaContentTabTutorial.doc” and “DigParam.rtf”, key “DigAIRange\ShowUserMasks”. The additional tabs allow editing custom-specific metadata for a show. If the manage dialog is displayed for a non-show node, the additional tabs are not available, only the general tab.
2) Like above, name and info can be filled as you like.
3) The button with the two heads opens the team dialog of the selected show. In this dialog you can define persons working with the show, like: host, speaker, chief, … Within this dialog there is also another button to change the access control list (ACL) for the show, see below.
4) Each show has a start time, stop time and duration. Only two of the values must be entered, the third is being calculated. You must pay attention with shows, which last across midnight. If you want to enter such a time, you must include the date! Simply write it before the time in German, English or international format. It is possible to have multiple shows for a particular time. This is irrelevant for planning, but the OnAir client will not be able to switch to the next show automatically if there are multiple shows, you have to load them manually.
   The order of the show list is normally given by the start time. Therefore, you should not change the order with the arrow buttons – unless the shows are not yet ordered by start time.

5) With the jingles field you can select one of the jingle groups, which have been defined for the service. This is the predefined jingle group, which will be loaded by the OnAir client together with the show.
6) The filler group is one of the jingle groups, which has been filled with fillers. The fillers can be automatically used by TurboPlayer to fill gaps.
7) The filler algorithm can be Standard or Random. In Standard mode, TurboPlayer will use the filler, which has not been used for the longest time ago. In Random mode TurboPlayer will pick one of the fitting fillers randomly.
8) The link field is only in use if you have a remote show assigned to the show. The field is read-only. To change it press the button to the right of it. The remote show dialog appears (see next chapter).
9) The OnAir directory is an additional media directory you can define per show. This can be useful if your shows are sent from individual studios typically and you have many on-air media directories as a consequence. Hint: only BUS evaluates this field when mirroring files. The media files are not copied directly into the specified directory, an additional level with the date of the show is inserted first.
10) In the field “Media directory profiles” you can enter a comma-separated list of names. Each of them can be a media directory profile, which is evaluated by BUS FileCheck task – see chapter 7.3 for more information.
11) The Show-ID field allows to specify a unique ID for the show. In principle, you can enter whatever you like but we recommend to include date and time to make it unique. This field is needed if a show should be used in multiple shows, meaning: rebroadcasting an original show. The field is evaluated by the Rebroadcast task of BUS – as the checkbox below.
12) The field “ConsistencyCheck hint” is a free-text field. You can enter any text you like. The field is intended to be evaluated by the ConsistencyCheck task of BUS. There you can write generic filter expressions, which define whether a show must be checked or not. In these filter expressions you can evaluate this field. For more information see chapter 7.16.
13) For the available playout channels, see the identical fields in the “Manage services” part above. Values defined for a show override the values of the service.
14) Import protection is a flag, which can be set to prevent that the automatic import in BUS writes into the show. This is only relevant if BUS is configured to perform an alignment the show structure and if the setting “Ignore protected shows” has been activated.
15) Rebroadcast show is a flag, which can be set to mark a show as a possible destination for an automatic scheduling of a rebroadcasted rundown. The scheduling is done by BUS task “Rebroadcast”.

8.5.4 Manage tracks

If you have loaded a show into one of the show windows of DigAIRange you can call the "Manage tracks" dialog – either by calling the corresponding menu command in the tree menu or the context menu of the tree view, or you can right-click on the rightmost button (a plus icon) in the header bar of the show window. This dialog appears:

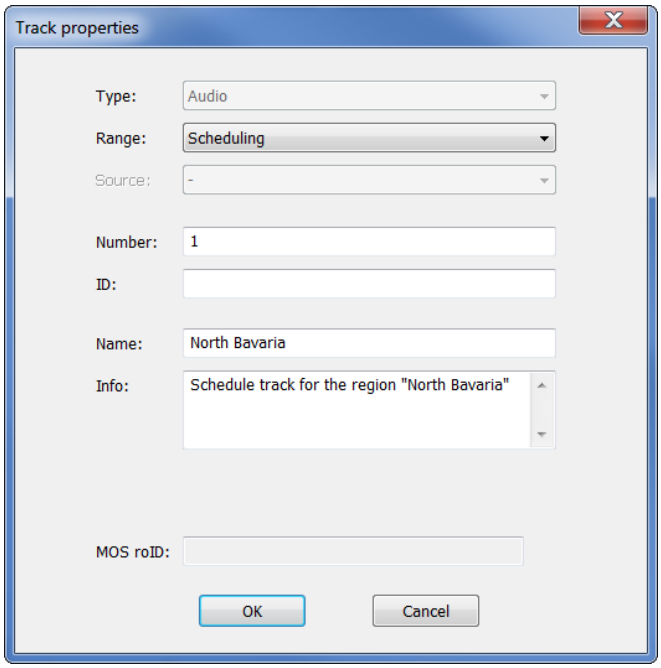


No	ID	Type	Range	Name	Info	Link
0	2	Audio	Scheduling	Main schedule		
1	3	Audio	Scheduling	Video schedule	Contains additional video	
2	6a	Audio	Scheduling	Remote track	Link to studio B	10.1.1.11 / B1 / 0
1000	66	Audio	Protocol	Main protocol		
1001	68	Audio	Protocol	Video protocol		
2001	69	Audio	Snapshot	05.05.2011 13:43:29	Created by: Marc	

This is an example of the show "On the road" as you can see on top. The show contains 6 tracks. The track properties are displayed in the list:

- No — Each track has a number. In principle the number is free, but there are certain numbers or ranges of numbers which have s specific meaning. The track with the number 0 normally is always present and is the main scheduling track. The track with the number 1000 is the standard protocol track. The allowed number ranges can bee seen below in "Range". Within a range you are free to select a number you want.
- ID — This is the last number of the full ID which is used to identify each node within the BCS tree. Normally you need not specify a value here becaue BCS will assign one automatically. But in case you want to use relatively linked elements on a distributed system with remote tracks it might become necessary to assign identical track IDs for all BCS locations.
- Type — This field was intended to distinguish different media types. But as it is possible now to mix all media types on a single track, this field has become obsolete and therefore the type is fixed to "Audio" (=the first value which was implemented).
- Range — One of:
  - Scheduling (No = [0-999])
  - Protocol (No = [1000-1999])
  - Snapshot (No = [2000-2999])
- Name — This can be any name which will be presented to the user. If you do not specify a name an automatically generated name will be displayed.
- Info — Some additional comments. Not used by the system.
- Link — In case a track is linked to a track on a remote BCS you will see the target of the link here. It is the address, the service and the remote track number.

For each track you can call the properties dialog. It allows to edit the information described above:

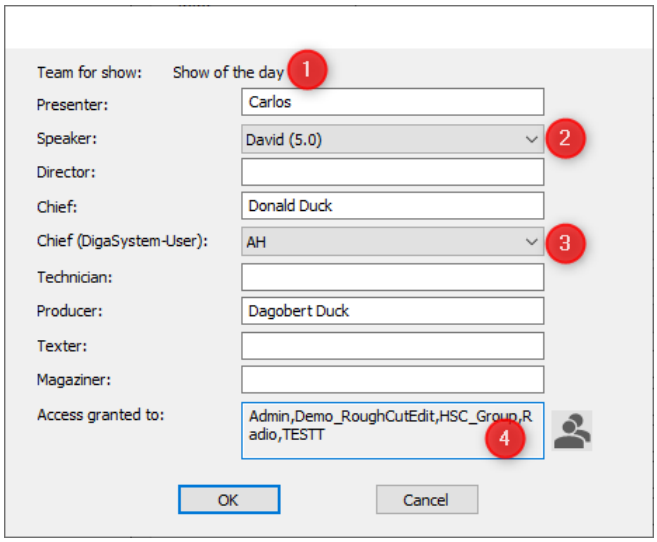


The MOS roID (rundown ID) cannot be edited. It is only for information and the field is only filled if a track was created by DigaMOS.

8.5.5 Team dialog

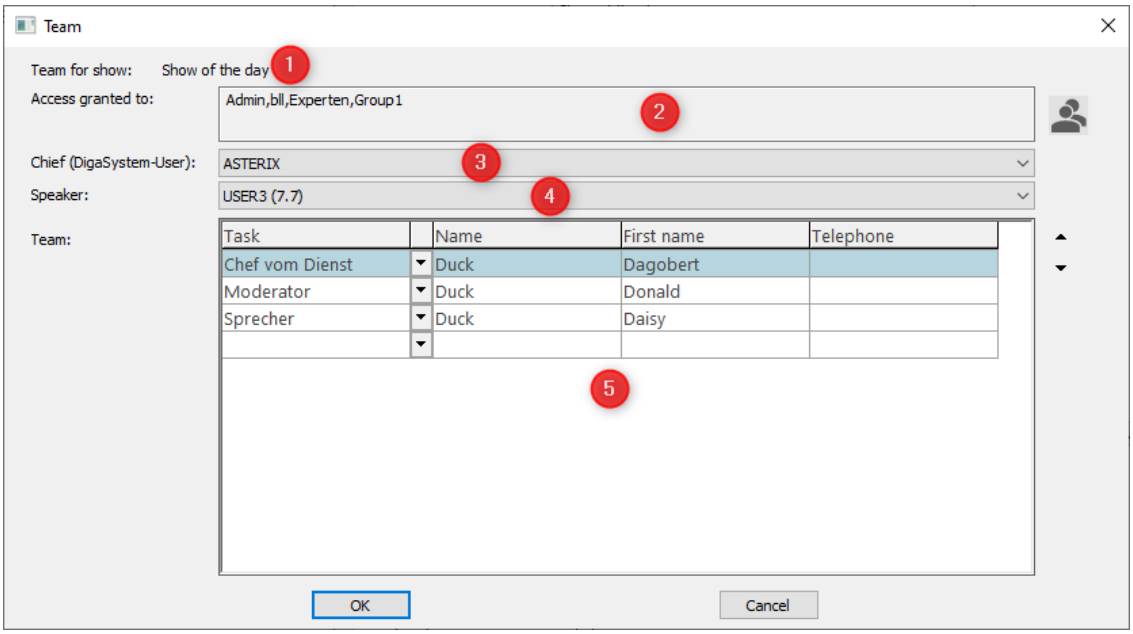
If you click on the button with the two heads in the manage dialog for a show, the team dialog opens. It allows to set the team members for a show.

The appearance of the dialog is different layouts, depending on the feature “free team list” (Option “Free teamlist” in the tab “View” of the general settings dialog of DigAIRange). If the free team list is not active, the dialog looks like this:



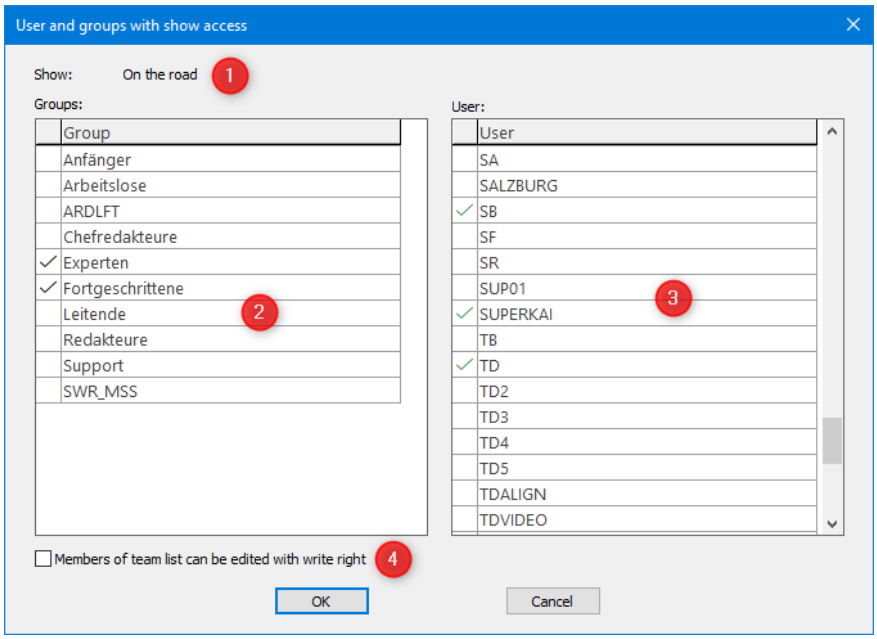
1. The name of the show you are editing.
2. The speaker – a DigaSystem user. Text length computation is based on this information.
3. The chief for the show – a DigaSystem user. This user has manage rights for the show.
4. The access control list (ACL). If names are listed, it’s the DigaSystem user ID. If a number is listed, it is a group number. To edit the ACL press the button with the two heads and the ACL dialog will open.
5. All other fields are free and do not have any specific meaning for the system.

With free team list being active, the dialog looks like this:



1. The name of the show you are editing.
2. The access control list (ACL). If names are listed, it's the DigaSystem user ID. If a number is listed, it is a group number. To edit the ACL press the button with the two heads and the ACL dialog will open.
3. The chief for the show – a DigaSystem user. This user has manage rights for the show.
4. The speaker – a DigaSystem user. Text length computation is based on this information.
5. A free list of tasks and persons.

8.5.6 Access control list (ACL)

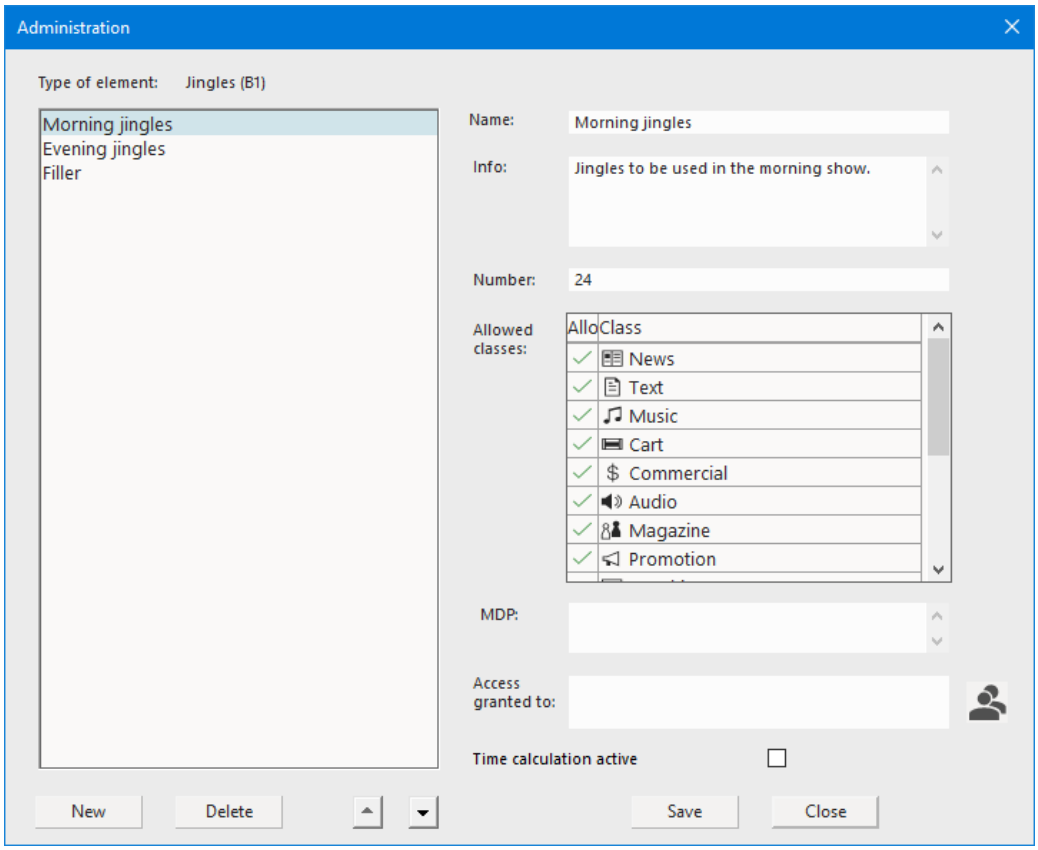


1. The name of the show (or group, or jingle group) you are editing.
2. The groups, which have access.
3. The users, who have access.

For more information about ACLs see chapter 1.5.3. In general: if an ACL is empty, everyone has access, otherwise only the listed groups and users. Being listed in the ACL gives you only a general write access. Other specific rights (e.g. class rights) still might be necessary.

8.5.7 Manage other node types

The manage dialog is available for nearly every node in the tree view, but it looks differently, depending on the node type. Here is an example for the jingles:



The “Number” field is used to limit the maximum number of jingles in this group. “Allowed classes” restricts the allowed classes in the jingle group.

The field "MDP" is for listing the names of media directory profiles. (See BUS FileCheck task above for more information about media directory profiles.)

The field “Access granted to” defines an access control list for the jingle group. To edit the list, click on the button with the two heads.

“Time calculation active” is a field which can be found in nearly every pool-like node. If active, the list is treated as a rundown with the usual time calculation of the BroadcastServer. If this option is deactivated, no time calculation is performed for this node. Instead it is possible to sort the list within DigAIRange by clicking on the column header.

8.5.8 Manage Remote Shows

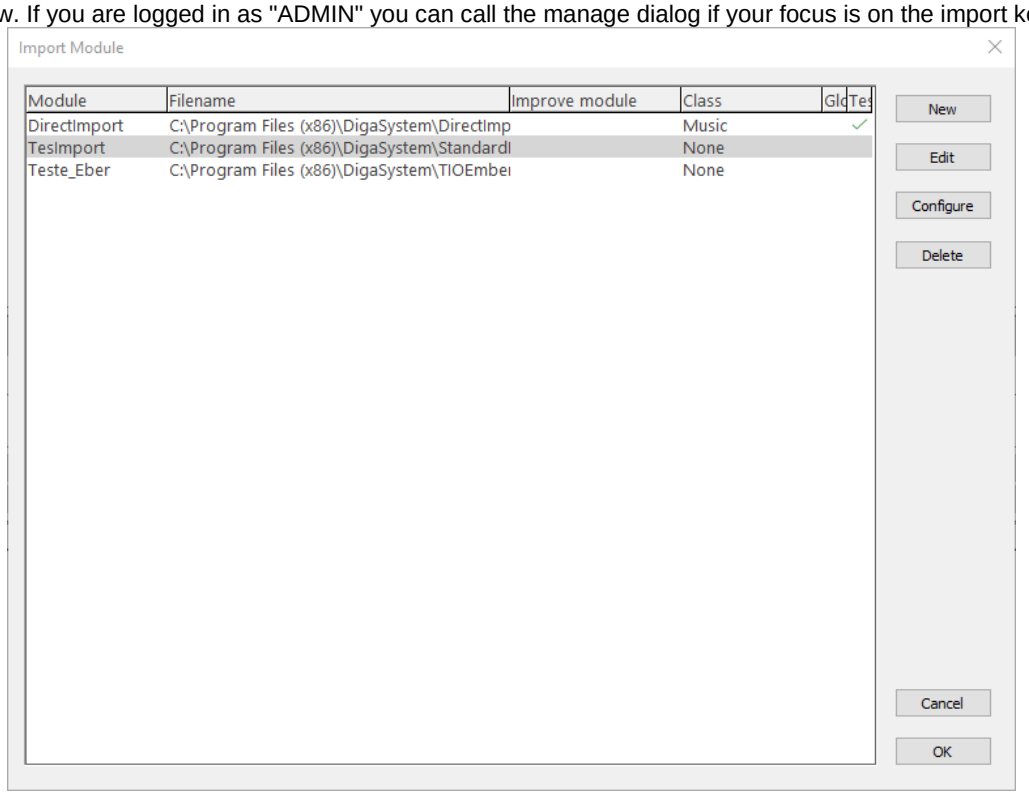
From the manage dialog of a show and a track you can call the remote-link dialog. In former versions of this document the dialog was explained here, but now you can find an extra main chapter "Regionalization and Remoting" below.

8.6 Other administration dialogs

8.6.1 Manage Import Modules

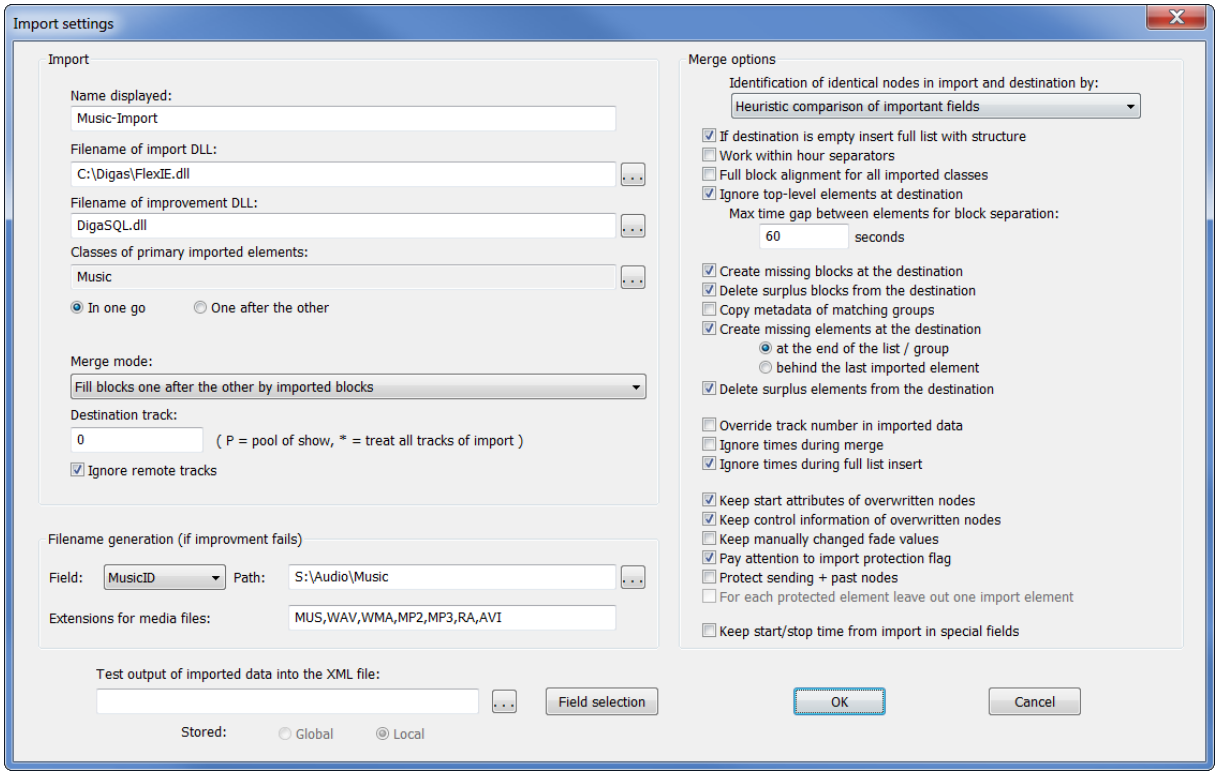
Hint: it is important to read the chapter about the import first to understand how the import works and what the options do.

Each import DLL together with a parameter set can be stored in DigAIRange below the "Import" key in the tree view. If you are logged in as "ADMIN" you can call the manage dialog if your focus is on the import key:



The first column is the name as it is displayed in the tree. The second column is the filename of the import module. In the example above you can see some DLLs multiple times but each of the lines has a different parameter set. The next column is the improvement module. The class column shows the classes which are imported by the module. The next column has a check if the parameters are stored in the global DigaSystem registry. The last column lists whether the import does a debug output of the imported XML data.

If you press the edit button or double-click on a line the edit dialog appears:

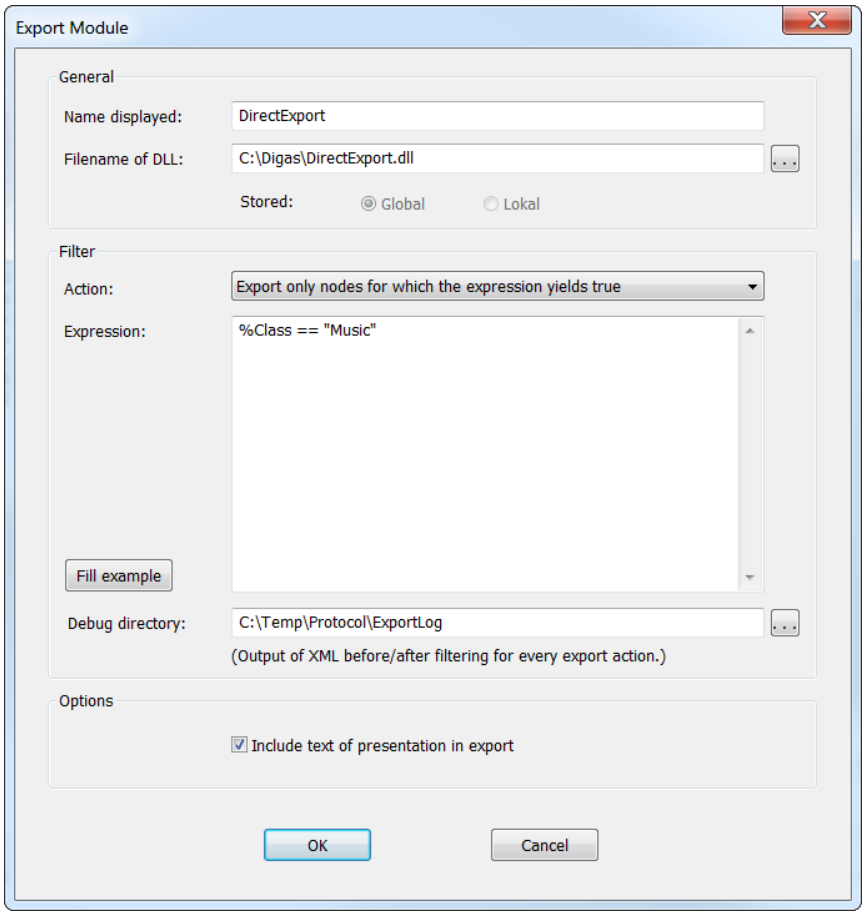


This is a dialog DigAIRange displays which is equal for each import module. In contrast, the similar button "Configure" opens the configuration dialog of the corresponding import DLL which is different for each DLL, of course. (These dialogs are described in the general chapter about the import.) The parameters from both dialogs are stored together in the DigaSystem registry below the "DigAIRange\Import" key.

If you have read the chapter 5.1 "The Import Function in DigAIRange and BUS", nearly all fields should be clear. One hint: all the options (the check- and dialog boxes and the primary class) apply only to the semi-automatic import which merges imported show and the BCS show. If a user simply opens an import in an extra window they do not have any meaning. The "test output" field is used for debugging purpose only. Here you can specify a filename where DigAIRange writes every XML string it receives from the import module. Normally the field should be empty.

8.6.2 Manage Export Modules

Managing the export modules is similar to managing import modules but more simple because there are no merge options to set. The first dialog with the list of all export modules is not even shown here. The DLL specific configuration dialog is described in the general chapter about the export. But we can discuss the export module dialog of DigAIRange briefly:



As you can see, there are not many fields to explain. The displayed name and the DLL filename should be clear.

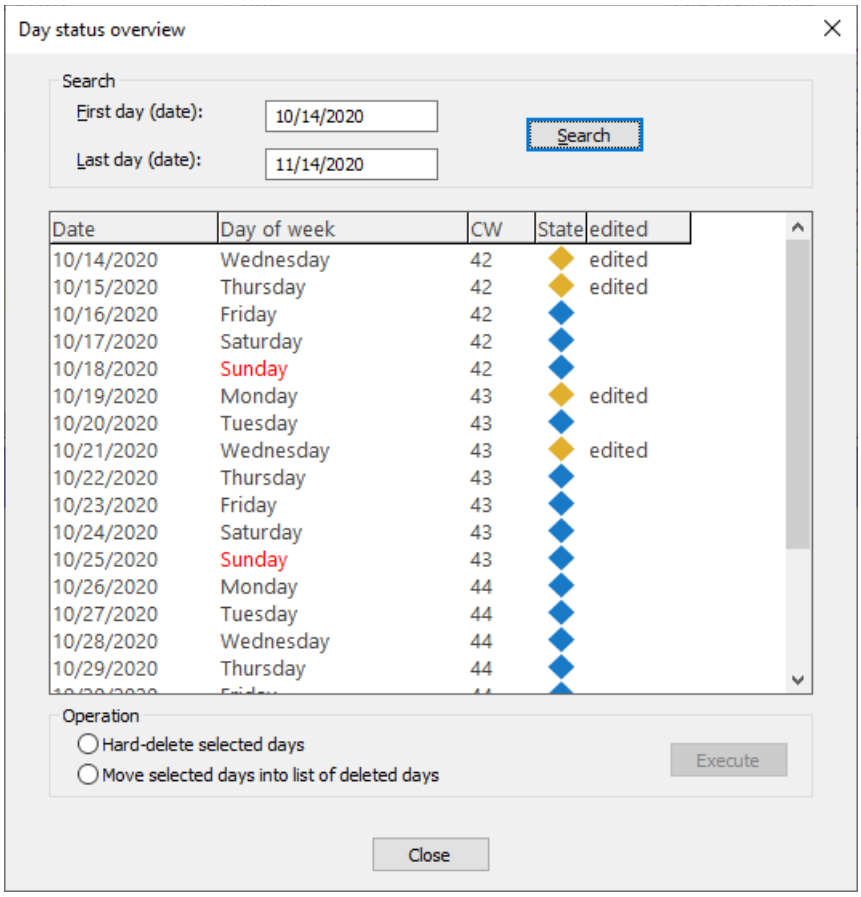
The filter expression allows to filter the nodes before exporting them. Please see the chapter about BUS / Export task and BUS / Filter expressions above for more information.

Beginning with DigAIRange version 6.0 you can also define your own macro functions in order to make the code more readable. If you do that, first define all functions you need and then define a last function with name "filter" (!), which must return either true (1) or false (0). Do not mix functions and declaring only a single expression.

The checkbox “Include text of presentation” must be checked if you want to include the text of the RTF file for each entry in the generated XML data. This is a time consuming operation because all files must be found, read and included in the XML string, so only use this option if you really need it.

8.6.3 Day status

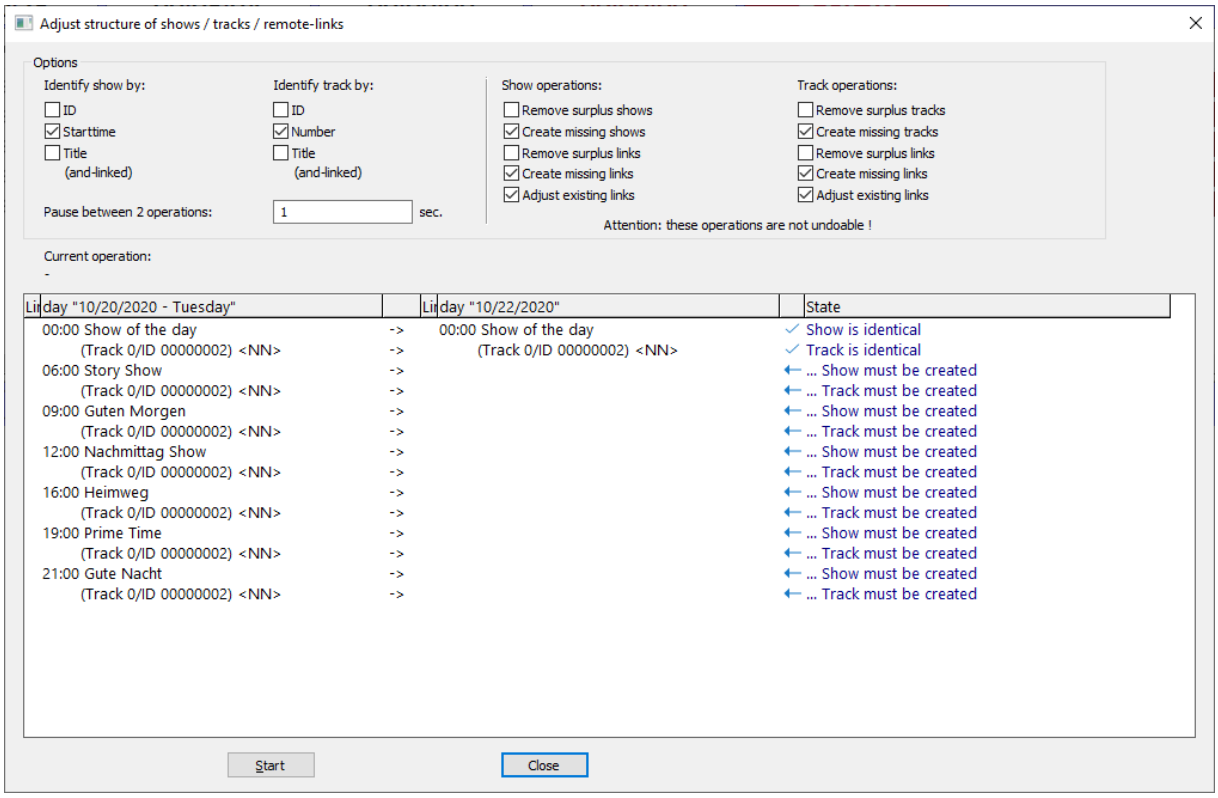
Since version 1.6 of DigAIRange there is a dialog to query the day status from the BCS:



The dialog is useful if you have to change a template and want to know where the template has already been used and/or changed. You can select all days which have been edited already and copy them to the list of deleted days. All days which have been opened already but which are unchanged can be deleted really. As the deleted days are empty afterwards, the BroadcastServer will automatically use the new template when the day is opened for the next time. Restoring the changed entries must be done manually, unfortunately, as this is an operation which is nearly impossible to be done automatically. So you can open each edited soft-deleted day and see which shows have been edited (can be identified if you define different colors for the tree view). Next you drag the individual elements which have been created manually from the deleted into the new shows.

8.6.4 Synchronize structure

Since version 1.6 of DigAIRange there is a dialog which allows to synchronize the structure of shows / tracks / remote-links:



The dialog appears when you drop a day onto a day or a show onto a show in the tree view and select "Adjust structure" in the following option dialog. This feature allows to copy parts of the information of the dragged node to the destination node. The big list view shows the result of an analysis of the source and the destination node. Some possible results are: a show is missing in the destination, a show is surplus at the destination, tracks are missing, remote links are missing or different.

With the options in the upper part of the dialog you select how identical shows/tracks are identified and which operations should be performed. Whenever you click on one of the options the analysis is done again and the list view is updated, so you can see what the operations will be before really executing them. The synchronizing operations start when you click on the "start" button. Afterwards the list view shows the results of the individual operations.

8.7 Special Settings

Nearly all parameters of DigAIRange are available from program dialogs. There are only a few special parameters which must be set in the DigaSystem registry directly. Here they are:

8.7.1 Service specific parameters

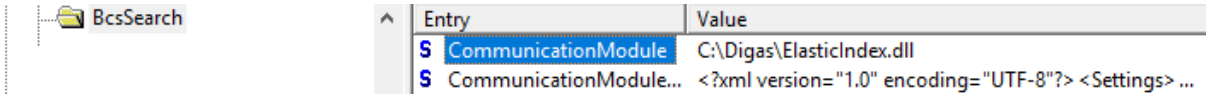
Some of the parameters are defined per service. They can be found in the key “Digas\PlanServer\<Server_Name>\<ServiceName>\DigAIRange”. An example as values are the three colours which are used to signal changed or completed shows. For a full description of these parameters see the technical document “DigaSystem – Parameters”

8.7.2 YearInterval

In the key “Digas\PlanServer\<Server_Name>” you can find the value “YearInterval”. This value defines how many years are visible in “Date” node of the DigAIRange tree. You can specify two values, one in the past and one in the future. For example: “-1,3” means one year back and three years in advance.

8.7.3 Full-text search module (BCSSearch.ocx)

DigAIRange can display the ActiveX control “BCSSearch.ocx” in its lower multi-view window. As usual, this AxtiveX must be registered with regsvr32.exe. The configuration for this module can be found in the top-level registry key “BCSSearch”. You need to configure the filename for the “communication module”. This module is responsible to make calls to the full-text search server (ElasticSearch). So far, only the 32-bit variant of the ElasticIndex.dll can be used:



The second parameter “CommunicationModuleConfiguration” can be used to specify all needed parameters for ElasticIndex.dll. You should not build the XML string yourself. Instead, navigate in Admin.exe to the registry settings of your BUS index task. Copy the content of the value “CommunicationModuleConfiguration”.

Using a single configuration file like this is only suitable if you have a single ElasticSearch server and a single index for the BCS content in it. If you have individual servers or indexes for each of your BCS servers, do not define this parameter. Instead, define the necessary communication parameters in the subkey “ElasticSearch” of each BCS key in the registry (->”Digas\PlanServer\<BCS name>”). This can then be different for each BCS, of course.

8.7.4 Print Playlist via Browser

In the local DigaSystem directory a subdirectory for XSLT stylesheets is needed.
When you have an MS Internet Explorer older than version 6 you need the files msxml3.dll, msxml3a.dll und msxml3r.dll within your windows/ system 32 directory.
In Admin /Global Settings you have to give in the path to the XSLT stylesheets:

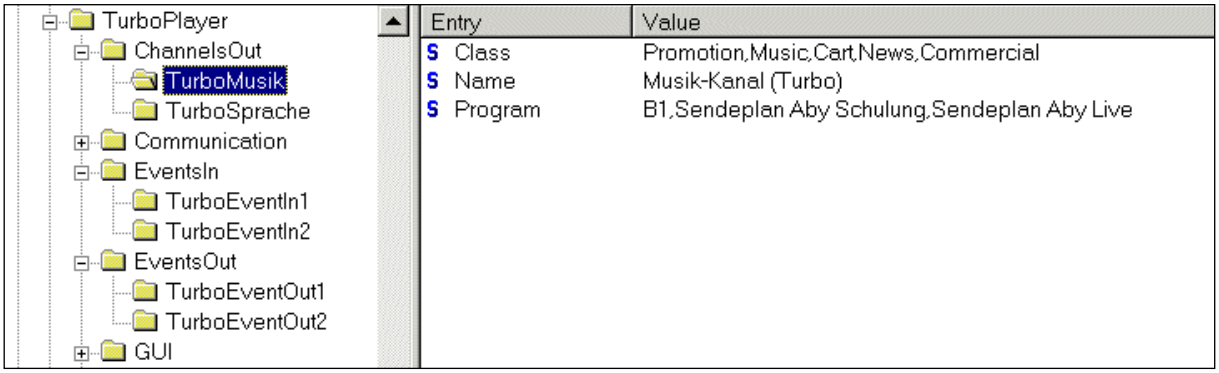
8.7.5 Audio Editors and DigAIRrange

For the communication between DigAIRrange and the editors, you need these keys:

And do not forget to check the value “Edigas\Flags\Create_Section=TRUE” !

8.7.6 Events and Channels

DigAIRange reads three keys from the settings of the on-air modules to display them in list of defined values. This applies to the in-events (used as external start event), the out-events (used to perform some external operations, typically by GPO) and out-channels (or better: the sound processing to be used). The settings are read from the two keys “DigaROC” and “TurboPlayer” and are added in the displayed lists, the subkeys are identical for both programs. Here is an example:



Typically you have a “Name” value which is the user-friendly name to be displayed and the “Program” value (program means service) where you define the services for which the key is valid. Other values are evaluated by the on-air applications only.

8.8 User masks

8.8.1 Step 1: requirements

In order to use a Custom Content Tab with DigAIRange the following conditions must be satisfied:

1. DigAIRange must be able to find a definition for the tab. This is done via a DigaSystem Registry entry.

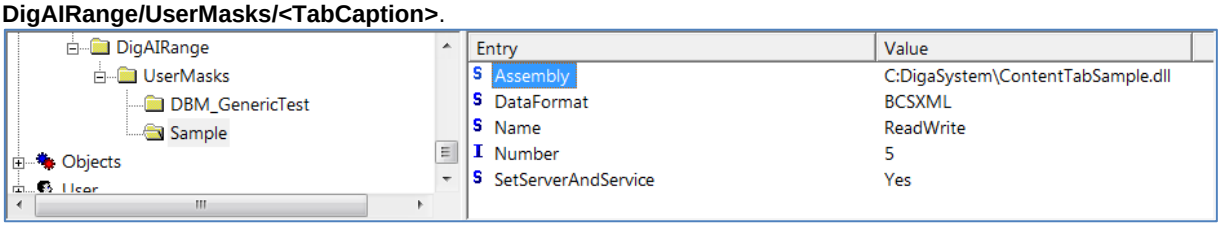
2. DigAIRange must be able to load **DigaContentTabHost.dll**, meaning it must be included in the search path for DLLs.

3. DigAIRange Version 5.2.651.0 or later must be used.

4. At the moment maximum 14 tabs can be defined.

8.8.2 Step 2: create DigaSystem Registry settings for a DigAIRange Custom Content Tab

DigAIRange Custom Content Tabs are defined in the DigaSystem registry (local or global) under the path **DigAIRange/UserMasks/<TabCaption>**.



Assembly	The tabs codebase.
DataFormat	Optional the data format to be used one of: BCSXML, DBX, DBE.
Name	Optional the tab name in case the dll contains more than one tab.
Number	The order in which the tab should appear. (1-14)
SetServerAndService	Optional, name value pairs which have meaning specific to the particular tab.

8.8.3 Step 3: configure the visible Content Tabs for a user

As the Content Tabs are defined locally or globally they are available to all users. Each user can define in his settings which of the configured tabs he want to use. Therefore he has to open the settings dialog of DigAIRange, select the “Mask” settings and choose the tabs:

Shown	Submask
✓	General
✓	General (group)
✓	News
✓	Music
✓	Commercial
✓	Presentation
✓	Control
✓	...more
✓	Classic
✓	BR
✓	Medium
✓	BR-GEMA
✓	BR-Word
✓	BR-Classic
✓	BR-Status
✓	Cart
✓	DBM_GenericTest_old

8.9 Print templates

DigAIRange allows printing directly onto a printer, writing an RTF file and calling WordPad afterwards, or writing an HTML file and calling the browser afterwards.

8.9.1 XSL templates

For printing via browser, you need stylesheets (XSL files) which specify the conversion of BCS XML to HTML. There is an extra document available (->"XSL for BCS") which describes the details of writing such a stylesheet. The stylesheets you have written are placed in a directory, preferable on a server so every client can use them. With the registry parameter "DigAIRange\XSLPath" you tell DigAIRange where it can find the stylesheets. If you want to have different templates for different services you can create subdirectories with the service name first and put all service-related templates into the corresponding directory. DigAIRange will then display a dialog box in which the user can select the desired service/subdirectory and below one of the stylesheets.

8.9.2 RTF templates

For printing via printer or RTF file RTF templates are available. These are simple RTF files which contain placeholders for some metadata fields which are replaced during the printing process. Here is an example:

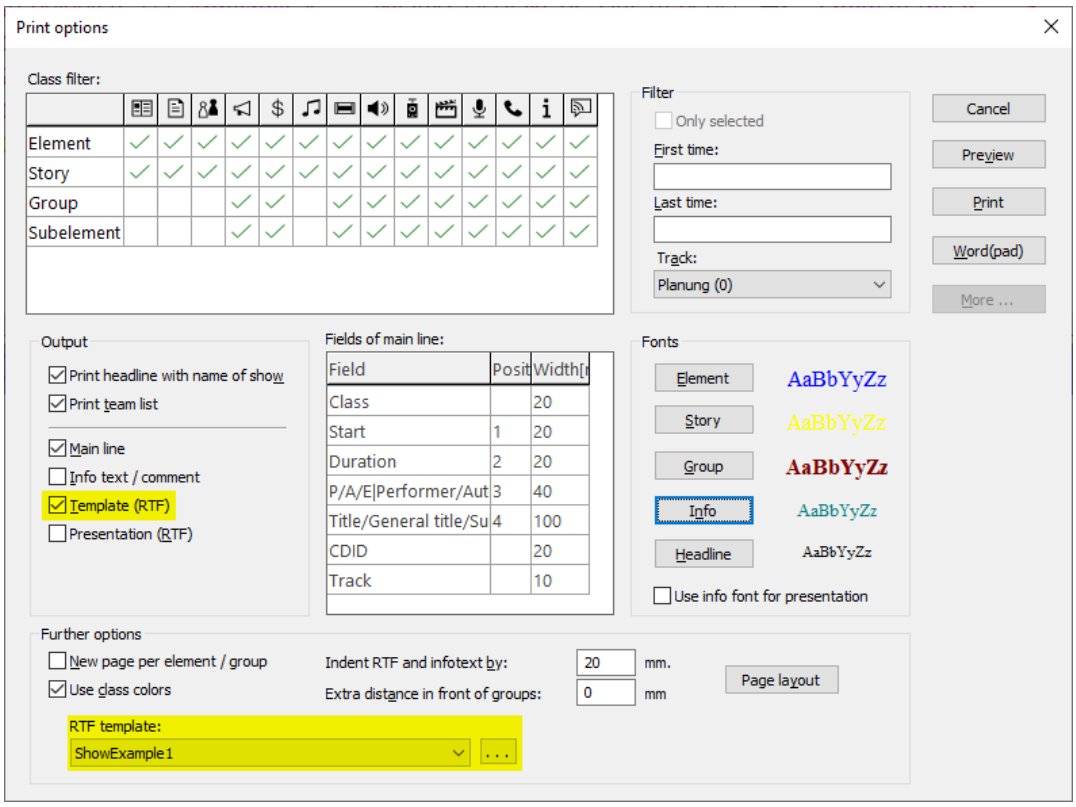
```
Item %ItemNumber% of show %ParentStartTime% "%ParentName%" on %ParentStartDate%:
%CombiPAE%: %CombiTitle%
Times: %StartTime% - %StopTime% [%TotalDuration%]
Info: %Info%
Presentation:
%Presentation%
```

As you can see, the placeholders have the format "%...%". You can use every metadata field which is available within BroadcastSystem, e.g. %Title% or %Time_Start% (see the list of all metadata fields above). As some of the stored information has a not very user-friendly format (e.g. duration in milliseconds) additional placeholder names have been defined to support a better printing. Here is a list of the additional names:

Name	Meaning	Comment
ItemNumber	An enumerating number for each item printed	Items are groups and elements.
StartDate	The start date	Dates and times are printed in local format. For times real times are preferred. Times are printed without milliseconds.
StartTime	The start time	
StopDate	The stop date	
StopTime	The stop time	
FirstStartDate	First start date	
FirstStartTime	First start time	
LastStartDate	Last start date	
LastStartTime	Last start time	
Duration	Audio (primary media) duration	For durations real values are preferred. Audio duration is between mark values. Durations are printed without milliseconds.
TextDuration	Text duration	
TotalDuration	Sum of durations	
TheStartMode	Start mode	The on-air mode/type is preferred.
TheStartType	Start type	
CombiTitle	One of: Title, Music_GenericTitle, Music_SubTitle	The first not empty field is printed.
CombiPAE	One of: Music_Performer, News_Author, News_Editor	
Presentation	The presentation with RTF formatting	
PresentationText	The presentation as pure text	

ParentName	Name	"Parent" is the node of the printed list, Show, JingleGroup, PreProduction, ...
ParentInfo	Info field	
ParentStartDate	Start date	
ParentStartTime	Start time	
ParentStopDate	Stop date	
ParentStopTime	Stop time	

You can use RTF templates within the print options dialog:

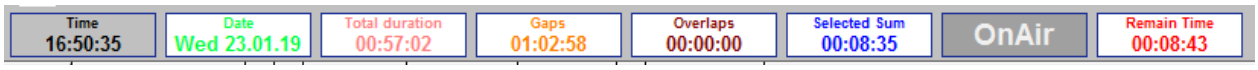


For RTF templates there is no dedicated directory. In the dialog the user can select any RTF file and afterwards the drop-down combo box lists all RTF files found in the directory of the selected file. The dialog stores all options in the user settings when it is being closed.

One additional remark: for creating RTF templates you should always use WordPad (or InfoPad, DigaText) as application but not Microsoft Word. If you use Word it is very easy to create an RTF file which is displayed correctly by the Windows rich edit control, but which cannot be used when replacing the placeholders. Reason is a bug in the Windows rich edit control which prevents a correct exchange of text in this case.

8.10 Time fields in DigAIRange

This document describes how DigAIRange computes the times which can be displayed in the time fields:



The following time fields can be displayed:

1. Local time
2. Date
3. Duration of show
4. Total playlength* of all elements of a show
5. Total duration of all texts (text elements + presentation text)
6. Sum of all visible gaps of a show
7. Sum of all visible overlaps of a show
8. Sum of duration* of all marked elements
9. Total playlength* of all marked elements
10. Total playlength* of all unmarked elements
11. Sum of media (non-text) duration* of all visible elements
12. Sum of text duration of all visible elements
13. Sum of media + text duration* of all visible elements
14. OnAir - Display
15. Remaining time of current OnAir element (till MarkOut)
16. Remaining time of current OnAir element (till LinkOut)

* Playlength = playout duration in rundown = duration between link points
Duration = duration between mark points (crossover times are counted twice)

8.10.1 Total duration

A concept used multiple times below is the total duration of an element. It is computed like this:

The duration mode of an element is important. It can be: audio only, text only, sum of audio and text. The special value Default should never occur in a BCS system, because if an element with duration mode Default is inserted into a BCS tree, BCS should apply the duration mode for the coresponding service.

For each element the total duration is computed:

If duration mode is audio only: if real times are present, the real duration (Time_RealDuration) is taken, otherwise the scheduled duration (Time_Duration).

If duration mode is text only: the text duration (Time_TextDuration) is taken. Hint: It is a big flaw in BroadcastSystem that an additional text cannot be "played" and does have neither a play state nor real times.

If duration mode is audio + text: both times, computed as described above, are added.

8.10.2 Link- and mark-values

The times as they are stored in BCS are all link-based. That means: start time = link-in point in timeline, stop time = link-out point in timeline. The duration as it is stored in BCS is the difference between the stored stop and the start, aka the link duration. If the duration between the two mark points must be computed, it is necessary to add the two link values to the duration. As an alternative it is also possible to compute mark-out - mark-in - though this only gives the scheduled duration, never the real duration.

Special things to have in mind (because the results can be very astonishing):

- Also, the real times are stored link-based.
- The computation above remains valid if a link value is negative. A strange result can be that the resulting duration (between the link points) becomes negative, though the element is played (between the mark points) for a positive duration, of course.
- The link and the mark values can each be empty. In this case the logic must pick the corresponding default value (e.g. Fade_LinkIn → Fade_DefaultLinkIn).
- The duration described above refers only to the audio (video) duration. Depending on the duration mode of an element the text duration might be added to the audio duration. This is even true, if an audio element was played and has now real times.
- On the other hand, we cannot "play" the additional presentation text (which is referenced by the additional text duration Time_TextDuration) and therefore there are no mark-, or link-values, or real times for this additional presentation text.

8.10.3 Computation info

Local time

This info is simply the local computer time, independent of any content.

Date

This info is simply the local computer date, independent of any content.

Duration of show

This info displays how long a loaded is defined: Time_Stop - Time_Start. For non-show nodes (Jingles, personal, general, show and day pools) a duration 00:00:00 is displayed.

Total playlength of all elements of a show

The computation is done like this: the playlength (= total duration as described above) for nearly all elements in a rundown is added. Elements, which have Time_StartType=Unused or which have Time_Overlaid=1 or which have Reserve=Unused or which don’t have a distribution endpoint with active time calculation are ignored. The function adds up the times for top-level elements, elements in a group or a story or elements in a story, which is part of a group. The times of stories and groups themselves are not added.

Total duration of all texts (text element + presentation text)

The computation is done like this: the text duration (Time_TextDuration) for nearly all elements in a rundown is added. Elements, which have Time_StartType=Unused or which have Time_Overlaid=1 or which have Reserve=Unused or which don’t have a distribution endpoint with active time calculation are ignored. The function adds up the times for top-level elements, elements in a group or a story or elements in a story, which is part of a group. The times of stories and groups themselves are not added.

Sum of all visible gaps of a show / Sum of all visible overlaps of a show

These two infos are computed in two steps. First, DigAIRange iterates through all visible elements of the rundown. It computes whether there is either a gap or an overlap between elements. If so, a special line is inserted in the view in between of these elements to make the gap or overlap visible. In the second step DigAIRange simply adds up the duration of all created gaps and of all overlaps and displays it in both info fields.

No gaps/overlaps are created for:

- A rundown, which can be sorted (has flag NoTimeAdjustment=1 and therefore is not sorted by time by BCS).
- In the past part of a schedule track of the current show if the setting DigAIRange\HideGaps=Current.
- In the past part of a schedule track of any show, which started in the past if the setting DigAIRange\HideGaps=Past.
- If the computed gap or overlap is smaller than the user parameter DigAIRange\ShowView\GapTolerance, which is by default 500ms. That means: no gap/overlap space is created in the rundown and the gap/overlap does not count for the corresponding sum.

Elements with TimeStartType=Unused or with Reserve=Unused or with Time_Overlaid=1 or which don't have a distribution endpoint with active time calculation are treated as if not existent in the rundown.

For the times of elements, the times as they are stored in BCS (the link-point times) are used. If a real time is present, it is preferred against the schedule time.

The rules for having a gap or an overlap are:

- Between two nodes with the same parent node simply the stop time of node 1 is compared with the start time of node 2, which might give a gap or an overlap.
- At the beginning of the rundown there might be a gap/overlap between the start time of the first top-level node (element/story/group) and the start time of the rundown (show).
- At the end of the rundown there might be a gap/overlap between the stop time of the last top-level node (element/story/group) and the stop time of the rundown (show).
- A story/group has a clearly visible beginning in the rundown: the header line. Therefore, it is clear that a gap/overlap before the story/group must be before this header. But as a story/group does not have a visible end, a gap/overlap behind a story/group is normally visible behind the last element in the story/group.
- There is one exception: a story/group normally has exactly the duration of the playlength of all children. But a story/group can be fully fixed in time (Time_StartType=Fixed) giving it a clear duration. The same applies for a group with attribute Time_FixedDuration=1. In this case DigAIRange draws a black line in the rundown, which signals the end of the story/group. In this case there might be two individual gaps/overlaps: one behind the last child of the story/group and this end, and a second one behind the end of the story/group and the start of the next node behind the story/group or between the end of the story/group and the end of the rundown (show).
- The first child of a story/group might have a different start time than its parent. In this case a gap/overlap exists between the story/group and its first child.
- If an element is a mute-start element (Time_MuteStart=1) and if the user setting DigAIRange\ShowView\ShowMuteGaps=FALSE (which is the default but it can be changed during runtime) than the computation uses the stop time of the mute-start element instead of the start time. If the user setting is TRUE than a mute-start element is treated like a regular one and its start time is used for the computation.

OnAir - Display

This info field always displays the text "OnAir". It is greyed if a show is not on-air and white color on red background (as standard settings) if a show is on-air. A show is treated as on-air if: either the current computer time is within the interval of the currently loaded show, or there is at least one node in the rundown with: SendState=Sending.

Sum of duration of all marked elements

DigAIRange computes this sum like the "Total playlength of all marked elements", but instead of using the total duration of each element, the total duration + both link values is taken. Hint: this yields the mark-duration, the duration between mark-in and mark-out (plus the optionally existing text duration, which is included in the total duration). For elements, which don't have a distribution endpoint with active time calculation, simply the alternative duration is used. For EndpointStories it depends on, whether individual children of the EndpointStory itself are selected or not. If either no child is selected or all children are selected then only the duration of the EndpointStory itself is added. If individual elements are selected, then only time of these elements is added.

Total playlength of all marked elements

For this info DigAIRange iterates among all elements, which are currently selected by the user. Top-level elements or elements in groups and/or stories are taken into account, but elements, which have Time_StartType=Unused or which have Time_Overlaid=1 or which have Reserve=Unused or which don't have a distribution endpoint with active time calculation are ignored. For EndpointStories it depends on, whether individual children of the EndpointStory itself are selected or not. If either no child is selected or all children are selected then only the duration of the EndpointStory itself is added. If individual elements are selected, then only time of these elements is added.

For all these elements, the total duration (possibly including the text duration) as it is described above is added.

Total playlength of all unmarked elements

Like before, but DigAIRange evaluates the elements, which are not marked.

Sum of media (non-text) duration of all visible elements

DigAIRange adds up the duration between the mark points for all visible elements. This includes elements which have Time_StartType=Unused or which have Time_Overlaid=1 or which have Reserve=Unused or which don't have a distribution endpoint with active time calculation.

Sum of text duration of all visible elements

Like before, but the text duration (of pure text elements and of media elements with additional presentation text) is added up.

Sum of media + text duration of all visible elements

The sum of the two previous counters.

Remaining time of current OnAir element (till LinkOut)

This field is only active if there is an active on-air element in the rundown. This would be the last element in the rundown, which has SendState="Sending". Of this element the value "Time_RealStop" is evaluated. The time difference between <now> and this stop time is computed. Afterwards 500 ms are added (to avoid rounding issues) and the time is rounded to full seconds, which are then displayed.

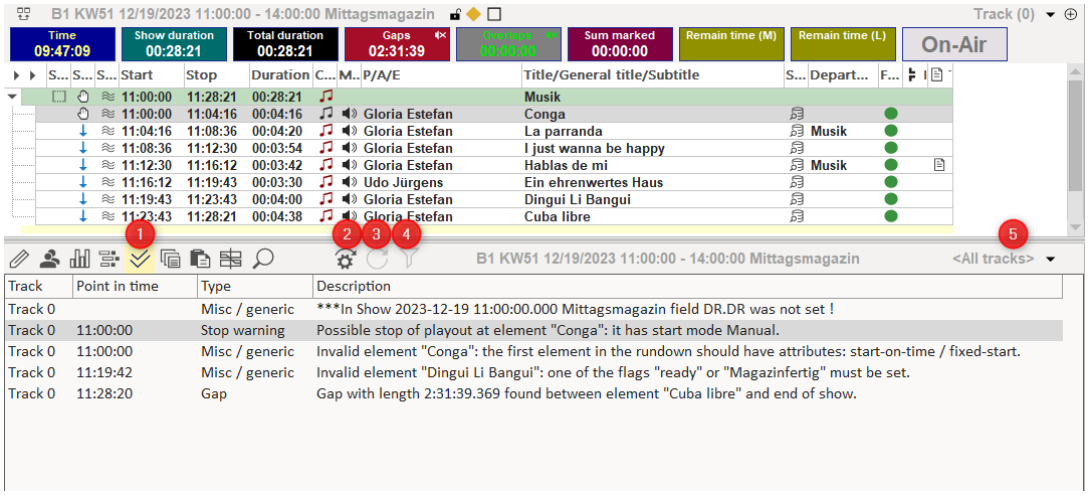
Since version 6.0 of the modules and with a WebSocket connection to BCS it is also possible for DigAIRange to receive play info messages from TurboPlayer via BCS. If a play info is available, it is preferred against the computed remain time.

Remaining time of current OnAir element (till MarkOut)

This is like the previous field, but the link-out is added to the stop time in order to yield the mark-out position in the timeline.

8.11 Consistency check view

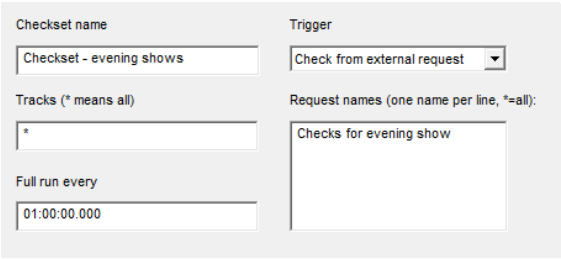
In the lower part of DigAIRange you can activate the view, which displays the results of one or more consistency checks. The content of this view is not populated by DigAIRange itself. You need BUS consistency check tasks running in the background. They must be configured to store their results in BCS ("Write results to BCS" in the general settings). This is not done in the regular BCS tree. BCS provides a special data space for storing these results and clients like DigAIRange can read out these results. Here is an example screenshot of the view:



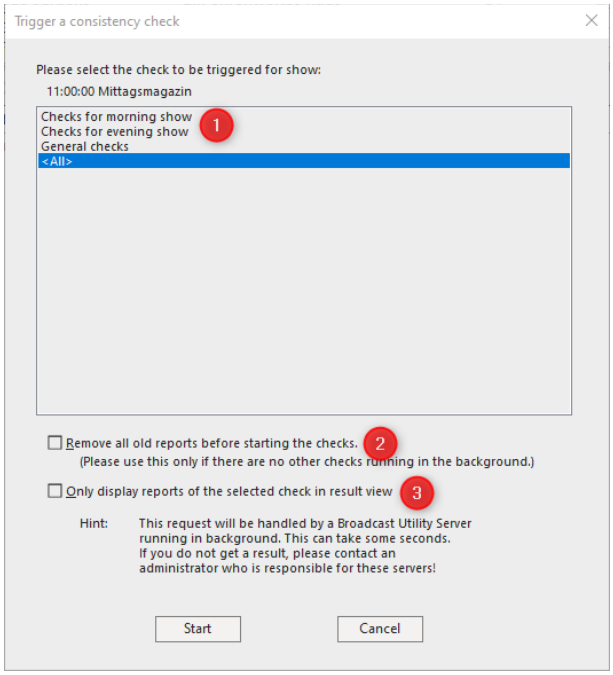
In the upper view you have to load a show and if you press button (1) you can see the results of consistency checks, which were already made, in the lower view. It depends on the configured checks of the BUS task, which criteria are checked and reported. The result view will display all reports, which are already available and for all tracks in the selected show. With button (5) you can filter for a specific track and by clicking into the header of columns you can sort the lines for a column.

If the checks in BUS are configured to run automatically and if the selected show is located within the monitored time range, it will normally only take one or two seconds until all checks were made and the results are available. These checks are repeated with every change of a show.

It is also possible to configure the checks to be externally triggered:



A checkset, which is configured in this way, will not start automatically, but wait for an external request with one of the specified request names. As you can see, the field “Request names” is a list, which means that the corresponding checksets can be triggered by multiple possible request names. In the screenshot only one request name is configured: “Checks for evening show”. The request can be triggered from DigAIRange with the buttons (2) or (3). If you press the button (2) a dialog will appear:



In list (1) you can see the available request names. BUT: DigAIRange does not read this list of request names from the BUS task settings. (This would not be realistic, because BUS instances are normally running on other computers, and it could be multiple ones. Therefore, scanning through all possible computer registries would be necessary to populate this list, which could take an incredible long time.) DigAIRange has its own registry value, in which the available request names can be listed. This is:

Digas\Planserver\<BCS name>\<program name>\AvailableConsistencyChecks

Sensibly, the request names in this parameter should be identical to the configured request names of the running BUS tasks. The item <All> is always added to the list. It will trigger all checksets, which are configured for an external trigger.

With checkbox (2) all existing reports for the selected show can be removed before new checks are started. This should only be used when the triggered checks are the ones, which will populate the list with new reports again. If there are additional (background) checks, which are not triggered, their results will get lost and will not be recreated by the triggered checks. A recreation might happen in the future (e.g. when the show is modified and a background check is monitoring the show) but at least for some time the reports of these other checks will be missing in the list.

With checkbox (3) DigAIRange will apply a filter to the list of reports so that only the reports of the triggered checks are being displayed. When this type of filter has been set, it can be activated or deactivated by pressing the button (4, the filter symbol) in the CC view of the main window.

Button (3) of the main view re-triggers the checks, which were triggered with the dialog the last time. This is identical to opening the dialog again and not modifying anything.

8.12 UI Schemes / Themes

Beginning with version 6.0 DigAIRange supports defining many colors of its GUI. This is called "Themes" in the GUI itself (e.g. in menu commands) and "UISchemes" in the registry. The user can change between defined UI schemes with the main menu command "View\Themes\...".

Before introducing the UI schemes, it was already possible to configure a lot of colors in DigAIRange. This made it quite complicated to introduce the new UI schemes, because some of the new settings were already available as an individual setting (typically of a user). In order to be backward-compatible individual settings are still preferred – if defined. If nothing is defined, DigAIRange will fall back to the corresponding color of the UI scheme, or – if nothing is defined there – to a standard color. You can also see parameters in the settings, which allow to define, whether the old colors or the colors of the UI scheme should be used. As an example, see the tab "Colors" of the general settings dialog. It allows to define class colors and also, whether these colors should be used or the corresponding colors from the UI scheme.

8.12.1 UI Schemes in the registry

DigAIRange reads the color settings of UI schemes from the DigaSystem registry and from an internal table in the following order:

1. User registry, Key DigAIRange\UISchemes\<name of UI scheme>
2. Local registry, Key DigAIRange\UISchemes\<name of UI scheme>
3. Global registry, Key DigAIRange\UISchemes\<name of UI scheme>
4. User registry, Key Common\UISchemes\<name of UI scheme>
5. Local registry, Key Common\UISchemes\<name of UI scheme>
6. Global registry, Key Common\UISchemes\<name of UI scheme>
7. Internal / hard-coded table

The steps 4-6 are not done for all possible values. See below in the list with all values, whether the value is read from the "Common" key, too.

For the last step (7) there are two internal UI schemes: "UIScheme_light" or "UIScheme_dark". When defining a new scheme, you can do it by defining only some of the desired values and for all other values make DigAIRange fall back to one of the internal colors. The fallback is possible to either the light or the dark default values. Define the registry value "DefaultUIScheme" below the registry key with your UIScheme. If the word "dark" appears in the text (case-insensitive) then DigAIRange falls back to the dark colors, otherwise to the light colors.

8.12.2 Available / default UI Schemes

Aside from defining UI schemes, it is also possible to configure, which of the defined schemes are available to a user and what scheme should be the default one. The available UI schemes can be configured in the registry value:

```
DigAIRange\UISchemes\UISchemes_available
```

This value is read from the registry: User, Local, Global (in this order), but **not** from the "Common" sub-key. In this value you can list the key names of UI schemes, which should be available, separated with a pipe character '|'. Example:

```
UIScheme_dark|UIScheme_light|UIScheme_medium
```

Each of the entries can have two parts, separated by an equal sign. In this case it means: "<registry key name>= <displayed name>". Without the "=", you have to define the registry key name and this is also the same string as it is displayed to the user.

It is also possible to configure the default UI scheme with the registry value:

```
DigAIRange\UISchemes\UIScheme_current
```

This value is read from the registry: User, Local, Global (in this order), but **not** from the "Common" sub-key. In this value you give the name of one of the available UI schemes. It will be used as the default when DigAIRange starts or when a user logs in. When the user selects a different UI scheme from the main menu, this UI scheme will become the new default one.

8.12.3 UI Scheme values

The following list enumerates all defined UIScheme values. The hint for "Common" shows, whether it is possible to define the value in the top-level key "Common", too. The data of each value must be defined with a string like:

```
RGB ( r, g, b )
```

or:

```
RGBA ( r, g, b, a )
```

The three values r, g and b are numbers between 0 and 255. The alpha value a must be a floating-point number between 0.0 and 1.0. A definition with alpha value is typically used for values, which define a color, which is used on top of another color. There the alpha value defines the transparency of the color. A value of 1.0 means: fully opaque. This is the same as only using an RGB value without alpha value. A value of 0.0 means: fully transparent. The defined color itself does no longer matter.

It is not possible to use the RGB(…) syntax when you don't want to define an alpha value. DigAIRange sometimes expects a value as an RGBA. This is visible in the hint "Alpha" below. If it says "yes", you have to use the RGBA syntax, otherwise the RGB syntax!

8.12.3.1 Colors for views, dialogs and grids

Color_Dialog_Bkgnd (Common: yes, Alpha: no)

Background color for dialogs (if not Color_PaneLeft or Color_PaneRight is used by DigAIRange). In dialogs with a tab control (like the mask or the settings dialog) most of the dialog is covered by the tabbed subdialogs, which have a background in color Color_PageBkgnd.

Color_GridSelected (Common: yes, Alpha: yes)

Background color for all selected lines within the grid windows if the window has the input focus. The specified color must have an alpha value, because the real color is computed as alpha blending of the specified color and the color Color_PaneRight.

Color_GridSelected_NoFocus (Common: yes, Alpha: yes)

Like Color_GridSelected but if the window does not have the input focus.

Color_GridGroupBkgnd (Common: yes, Alpha: no)

Background color for group lines in the rundown view.
This color is used for the grouping lines in the "More" tab of the mask, too.

Color_GridStoryBkgnd (Common: yes, Alpha: no)

Background color for EndpointStory lines in the rundown view.

Color_GridEndpointStoryBkgnd (Common: yes, Alpha: no)

Background color for story lines in the rundown view.

Color_GridGapBkgnd (Common: yes, Alpha: yes)

Background color for gaps in the rundown view.
Attention: the alpha value here is not applied as a real alpha value. It is only intended to allow disabling of the colorization of gaps. So, if alpha=1.0 the color is applied and if alpha=0.0 the gap is transparent, meaning it has the regular background color of the rundown view. Don’t use other alpha values! If you do not specify this value or make it unused by setting the alpha value to 0.0, the default is applied, which is a yellow color.

Color_GridOverlap (Common: yes, Alpha: no)

Color of the zig-zag line, which is painted in the rundown view between nodes, which have an overlap. By default, a bright red is used.

Color_GridSeparatorBkgnd (Common: yes, Alpha: yes)

Background color for elements of type: hour-separator in the rundown view.
Attention: the alpha value here is not applied as a real alpha value. It is only intended to allow disabling of the colorization of gaps. So, if alpha=1.0 the color is applied and if alpha=0.0 the gap is transparent, meaning it has the regular background color of the rundown view. Don’t use other alpha values! If you do not specify this value or make it unused by setting the alpha value to 0.0, the default is applied, which is an orange color.

Color_PaneLeft (Common: yes, Alpha:no)

Background color for the left tree view in the main window of DigAIRange. The color is also used as background color for all other grid and tree windows (in dialogs or drop-downs). For the left tree view of the main window this color is only used if no color for this view has been defined in the old user-specific parameters (settings dialog -> tab "Tree").

Color_PaneLeft_Caption (Common: yes, Alpha: no)

So far not used within DigAIRange.

Color_PaneLeft_GridLines (Common: yes, Alpha: no)

So far not used within DigAIRange.

Color_PaneRight (Common: yes, Alpha:no)

Background color for the right views in the main window of DigAIRange (show-views, multi-view). This color is not used in the rundown windows if a specific color was set in the user parameters for a specific node type (->settings dialog, tab "List view", group "Background colors").
Also used as background color for buttons in the left view of the main window if the "easy selection dialog" instead of the tree view is used.

Color_PaneRight_Caption (Common: yes, Alpha: no)

Background color for the tool / title bar of sub-windows, e.g: the navigation, time and filter bar above a rundown window, or the top part of the window hosting CFM/OTM, or the top part of the show overview, or the top part of the remote BCS/BUS-GUI window. This color is used if the window below the caption does not have the input focus. This color will not be applied if a user-specific color has been set in the parameters (settings dialog, tab "View", group "Title bar background").

Color_PaneRight_Caption_Focus (Common: yes, Alpha: no)

Background color for the title bar of the views on the right-hand side of the main window, the rundown windows and the multi-view. This color is used if the window below the caption has the input focus. If this color is not given, the color Color_PaneRight_Caption is used for a focused window, too. This color will not be applied if a user-specific color has been set in the parameters (settings dialog, tab "View", group "Title bar background").

Color_PaneRight_GridLines (Common: yes, Alpha: no)

Color of the grid lines of the rundown windows.

Color_PaneRight_HeadlineBkgnd (Common: yes, Alpha: no)

Background color for the headline (with the column names) of a rundown window. By default, this value is not defined in the internal UI schemes and therefore the color Color_PaneRight will be used.

Color_PaneRight_JinglesBkgnd (Common: yes, Alpha: no)

Background color for the rundown view when a jingle group is loaded. By default, this value is not defined and the color Color_PaneRight is used. This value can be overridden by a user-specific setting.

Color_PaneRight_PoolBkgnd (Common: yes, Alpha: no)

Background color for the rundown view when a general or a personal pool is loaded. By default, this value is not defined and the color Color_PaneRight is used. This value can be overridden by a user-specific setting.

Color_PaneRight_ShowpoolBkgnd (Common: yes, Alpha: no)

Background color for the rundown view when a show pool is loaded. By default, this value is not defined and the color Color_PaneRight is used. This value can be overridden by a user-specific setting.

Color_PaneRight_DaypoolBkgnd (Common: yes, Alpha: no)

Background color for the rundown view when a day pool is loaded. By default, this value is not defined and the color Color_PaneRight is used. This value can be overridden by a user-specific setting.

Color_PaneRight_ProtocolBkgnd (Common: yes, Alpha: no)
Background color for the rundown view when a protocol track is loaded. By default, this value is not defined and the color Color_PaneRight is used. This value can be overridden by a user-specific setting.

Color_PaneRight_SnapshotBkgnd (Common: yes, Alpha: no)
Background color for the rundown view when a snapshot track is loaded. By default, this value is not defined and the color Color_PaneRight is used. This value can be overridden by a user-specific setting.

Color_PaneRight_TemplateBkgnd (Common: yes, Alpha: no)
Background color for the rundown view when a template is loaded. By default, this value is not defined and the color Color_PaneRight is used. This value can be overridden by a user-specific setting.

Color_PaneRight_TrashBkgnd (Common: yes, Alpha: no)
Background color for the rundown view when a day trash is loaded. By default, this value is not defined and the color Color_PaneRight is used. This value can be overridden by a user-specific setting.

8.12.3.2 Text colors

Color_RTF_Bkgnd (Common: yes, Alpha: no)
Background color for RTF text controls (so far only within DigaRTF.ocx).

Color_Text_Standard (Common: yes, Alpha: no)
Used for text within many controls, especially: buttons, menus, list boxes, combo boxes, tab controls, static text in dialogs. This color is applied if the control is enabled.

Color_Text_Disabled (Common: yes, Alpha: no)
Used for text within many controls, especially: buttons, menus, list boxes, combo boxes, tab controls. This color is applied if the control is disabled.

Color_Text_Highlighted (Common: yes, Alpha: no)
Only used for text in the header of a tab control for hovering mouse.

Color_Text_Inverse (Common: yes, Alpha: no)
Like Color_Text_Standard, but automatically applied if the background on which the text should be drawn is the inverse of the standard background (=light if the standard background is dark and dark if the standard background is light).

Color_Text_InverseDimmed (Common: yes, Alpha: no)
Like Color_Text_Inverse but only used for text in the header of an inactive tab of a tab control.

Color_Text_InverseHighlighted (Common: yes, Alpha: no)
Like Color_Text_Inverse, used for text in the header of a tab control for hovering mouse.

Color_TextEdit_Bkgnd (Common: yes, Alpha: no)
Used for the background of edit controls if the control is enabled.

Color_TextEdit_ReadOnlyBkgnd (Common: yes, Alpha: no)
Used for the background of edit controls if the control is read-only.

8.12.3.3 Menu colors

Color_Menu_BkgndHighlighted (Common: yes, Alpha: no)
Background color for a highlighted (=selected) menu item.

Color_Menu_BkgndStandard (Common: yes, Alpha: no)
Background color for menus.

Color_Menu_CheckmarkBkgnd (Common: yes, Alpha: no)
Background color for menu items, which are checked.

Color_Menu_CheckmarkBkgndHighlighted (Common: yes, Alpha: no)
Background color for menu items, which are checked and highlighted (=selected).

8.12.3.4 Button colors

Color_ButtonBkgnd (Common: yes, Alpha: no)
Default color for background of buttons. It is not recommended to change this color, because the background color for buttons is normally chosen according to the background of the surrounding. This value is not applied to all types of buttons. E.g. toolbar buttons or buttons, which display a color settings are excluded. Nevertheless, this color is applied as background color for text buttons, too. (For extra colors for text buttons: see below.)

Color_ButtonBkgndChecked (Common: yes, Alpha: no)
Default color for button background in checked state.
This is the color, which is applied to (permanently) checked buttons, e.g. filter buttons in the filter bar of a rundown window or one of the content buttons in the title bar of the multi-view.
Hint: for the filter bar buttons the user can override this setting in the settings dialog, tab

Color_Button_Def_ShapeMaskHover (Common: yes, Alpha: yes)
Default color and opacity for hover shape mask (mostly alpha).

Color_Button_Def_SymbolChecked (Common: yes, Alpha: no)
Default color for button symbols in checked state.
Hint: this is also the color, which is applied to (permanently) checked buttons, e.g. filter buttons in the filter bar of a rundown window or one of the content buttons in the title bar of the multi-view.

Color_Button_Def_SymbolChecked_Deviating (Common: yes, Alpha: no)
Color when button's checked background color is defined as non-standard.

Color_Button_Def_SymbolCheckedDisabled (Common: yes, Alpha: no)
Default color for button symbols in checked+disabled state.

Color_Button_Def_SymbolNormal (Common: yes, Alpha: no)
Default color for button symbols. This color is also used for many icons: in the toolbar, in the tree view, in the rundown windows, in the title- / tool-bar of rundown windows,

Color_Button_DisabledSymbol (Common: yes, Alpha: yes)
Color and opacity for disabled symbol (mostly alpha).

Color_Button_ShapelessHover_Blend (Common: yes, Alpha: yes)
Only if no shape mask defined: Is blended over background for hover (contains alpha).

Color_Button_ShapelessHover_Blend_CaptionRight (Common: yes, Alpha: yes)
So far not used in DigAIRange.

Color_Button_ShapelessPushed_Blend (Common: yes, Alpha: yes)
Only if no shape mask defined: Is blended over background for pushed button (contains alpha).

Color_Button_ShapelessPushed_Blend_CaptionRight (Common: yes, Alpha: yes)
So far not used in DigAIRange.

Color_Button_ShapeNormal (Common: yes, Alpha: yes)
Color and opacity for normal shape.

Color_Button_Toolbar_ShapelessHover_Blend (Common: yes, Alpha: yes)
For toolbar buttons: Only if no shape mask defined: Is blended over background for hover (contains alpha).

Color_Button_Toolbar_ShapelessPushed_Blend (Common: yes, Alpha: yes)
For toolbar buttons: Only if no shape mask defined: Is blended over background for pushed button (contains alpha).

Color_TextButton_Border (Common: yes, Alpha: no)
The border color of text buttons (like e.g. the OK/Cancel buttons in dialogs).

Color_TextButton_Border_Focus (Common: yes, Alpha: no)
The border color of text buttons in case the button has the input focus.

Color_TextButton_Background_Disabled (Common: yes, Alpha: no)
The background color of a disabled text button. (Hint the regular background color is given by Color_ButtonBkgnd.)

Color_TextButton_Background_Hover (Common: yes, Alpha: no)
The background color of a text button while the mouse cursor is hovering on top of the button. (Hint the regular background color is given by Color_ButtonBkgnd.)

Color_TextButton_Background_Focus (Common: yes, Alpha: no)
The background color of a text button, which has the input focus. (Hint the regular background color is given by Color_ButtonBkgnd.)

Color_DefaultButton_Background (Common: yes, Alpha: no)
The background color of a text button, which is the default button of a dialog.

Color_DefaultButton_Background (Common: yes, Alpha: no)
The background color of a text button, which is the default button of a dialog.

Color_DefaultButton_Background_Focus (Common: yes, Alpha: no)
The background color of a text button, which is the default button of a dialog, while the button has the input focus and the dialog is the currently active window.

Color_DefaultButton_Background_Hover (Common: yes, Alpha: no)
The background color of a text button, which is the default button of a dialog, while the mouse cursor is hovering on top of the button.

8.12.3.5 Scrollbar colors

Hint: so far, it was not yet possible to replace all scrollbars by owner-drawn ones. Therefore, some controls are the original ones of Windows and their scrollbars do not follow the style of embellished scrollbars. This affects e.g. classical multi-line edit boxes in dialogs.

Color_Scrollbar_ArrowBackground (Common: yes, Alpha: no)
The background color for the area with the arrows (up, down, left, right).

Color_Scrollbar_Background (Common: yes, Alpha: no)
The color of the background of scrollbars.

Color_Scrollbar_Edge (Common: yes, Alpha: no)
The color of the edge around a scrollbar.

Color_Scrollbar_Thumb (Common: yes, Alpha: no)
The color of the thumb area, which can be moved with the mouse.

Color_Scrollbar_ThumbBottomEdge (Common: yes, Alpha: no)
The color of the bottom edge of the thumb area.

Color_Scrollbar_ThumbTopEdge (Common: yes, Alpha: no)
The color of the top edge of the thumb area.

8.12.3.6 Other colors

Color_FaderHandle (Common: yes, Alpha: no)
Color of fader handler in MultiRec.

Color_FaderSlitAbove (Common: yes, Alpha: no)
Used somewhere in MultiRec.

Color_FaderSlitBelow (Common: yes, Alpha: no)
Used somewhere in MultiRec.

Color_Line_BarBottom (Common: yes, Alpha: yes)
Color for the bottom edge line of multiple controls, like buttons, tab controls, toolbars, menus. The alpha value is used, because the specified color is blended into the color of the background of the control.

Color_Line_BarTop (Common: yes, Alpha: yes)
Color for the top edge line of multiple controls, like buttons, tab controls, toolbars, menus. The alpha value is used, because the specified color is blended into the color of the background of the control.

Color_Line_Divider (Common: yes, Alpha: no)
Only used within tab controls for a small divider line.

Color_Line_SunkenEdgeBottom (Common: yes, Alpha: no)
Like Color_Line_BarBottom, but for the control state "sunken".

Color_Line_SunkenEdgeTop (Common: yes, Alpha: no)
Like Color_Line_BarTop, but for the control state "sunken".

Color_MultiRec_Bkgnd (Common: yes, Alpha: no)
The background color for MultiRec. Recommendation (and default) is the same color as Color_PageBkgnd.

Color_MultiRec_TimeLine_Bkgnd (Common: yes, Alpha: no)
The background color for the waveform in MultiRec.

Color_MultiRec_TimeLine_Waveform (Common: yes, Alpha: no)

The color for the waveform in MultiRec.

Color_ShowStatus_Unchanged (Common: yes, Alpha: no)

Color to signal the status of a show “Unchanged”. It is used in the tree view, the show/day overview and the diamond symbol in a show header. This value is only applied if no service specific color is set in parameter Digas\PlanServer\<BCS name>\<service name>\DigAIRange\TreeColorUnchanged. In the internal UISchemes this value is not set and therefore a standard color like Color_Text_Standard is used.

Color_ShowStatus_Changed (Common: yes, Alpha: no)

Color to signal the status of a show “Changed”. It is used in the tree view, the show/day overview and the diamond symbol in a show header. This value is only applied if no service specific color is set in parameter Digas\PlanServer\<BCS name>\<service name>\DigAIRange\TreeColorChanged. In the internal UISchemes this value is not set and therefore a standard color like Color_Text_Standard is used.

Color_ShowStatus_Completed (Common: yes, Alpha: no)

Color to signal the status of a show “Completed” (=”Approved”). It is used in the tree view, the show/day overview and the diamond symbol in a show header. This value is only applied if no service specific color is set in parameter Digas\PlanServer\<BCS name>\<service name>\DigAIRange\TreeColorCompleted. In the internal UISchemes this value is not set and therefore a standard color like Color_Text_Standard is used.

Color_ShowStatus_Protected (Common: yes, Alpha: no)

Color to signal the status of a show “Protected”. It is used in the tree view, the show/day overview and the diamond symbol in a show header. This value is only applied if no service specific color is set in parameter Digas\PlanServer\<BCS name>\<service name>\DigAIRange\TreeColorProtected. In the internal UISchemes this value is not set and therefore a standard color like Color_Text_Standard is used.

Color_Signal_Green (Common: yes, Alpha: no)

Used for drawing some green symbols. These are: a green check symbol in some grid windows, the check icon on some buttons, the day status "Approved".

Color_Signal_Red (Common: yes, Alpha: no)

Used for drawing some red symbols. These are: a red cross in some grid windows, the lock symbol or text with information if a node is locked, the day status "Protected", the database flag "Invalid". This color is also used for the progress bar in the play time / remain time column of a rundown view.

Color_Signal_Yellow (Common: yes, Alpha: no)

Used for drawing some yellow symbols. These are: the icon for "Info preview" in the filter bar, the day status "Changed", the database flag "Archiving".

Color_Sphere_A (Common: yes, Alpha: no)

Background color for buttons and scroll bars.

Color_StatusBar (Common: yes, Alpha: no)

Background color of the status bar on bottom of main window.

Color_Tab_Inactive (Common: yes, Alpha: no)

Not used in DigAIRange. Instead, color Color_Tab_Popup_Inactive is used for this purpose, though this color has nothing to do with popups.

Color_Tab_Popup_Inactive (Common: yes, Alpha: no)

The background color for inactive items in tab controls (the items in the header, not the page with the tabbed content.)

Color_Tooltip_Bkgnd (Common: yes, Alpha: no)

The background color of tooltips in the rundown view or title tips in the tree view. By default, the Windows system color 24 “INFOBK” is used in light mode and the color RGB (175, 175, 0) is used in dark mode.

Color_Tooltip_Text (Common: yes, Alpha: no)

The text color of tooltips in the rundown view or title tips in the tree view. By default, the color of a tree item is used as text color for a title tip in the tree view and the colors “blue” and “black” are used for tooltips in the rundown view.

Color_VUMeterPhaseIndicator (Common: yes, Alpha: no)

So far not used in DigAIRange.

8.12.3.7 Toolbar symbol colors

Color_Button_Toolbargroup_View (Common: no, Alpha: no)

Color for the icons in the toolbar group "View". This includes the buttons: toggle selection view, toggle vertical splitter, toggle horizontal splitter. Setting an individual color overrides the group color.

Color_Button_Toolbargroup_General (Common: no, Alpha: no)

Color for the icons in the toolbar group "General". This includes the buttons: login, print list, print list via browser, delete, fulltext search, day overview, show overview. Setting an individual color overrides the group color.

Color_Button_Toolbargroup_UndoRedo (Common: no, Alpha: no)

Color for the icons in the toolbar group "UndoRedo". This includes the buttons: undo, redo. Setting an individual color overrides the group color.

Color_Button_Toolbargroup_Edit (Common: no, Alpha: no)

Color for the icons in the toolbar group "Edit". This includes the buttons: play, Multitrack, EasyTrack, SingleTrack, text editor, story editor, OTM, DBM. Setting an individual color overrides the group color.

Color_Button_Toolbar_ToggleSelectionView (Common: no, Alpha: no)

Color for the toolbar icon "Toggle selection view".

Color_Button_Toolbar_ToggleVerticalSplitter (Common: no, Alpha: no)

Color for the toolbar icon "Toggle vertical splitter".

Color_Button_Toolbar_ToggleHorizontalSplitter (Common: no, Alpha: no)

Color for the toolbar icon "Toggle horizontal splitter".

Color_Button_Toolbar_Login (Common: no, Alpha: no)

Color for the toolbar icon "Login".

Color_Button_Toolbar_Undo (Common: no, Alpha: no)

Color for the toolbar icon "Undo".

Color_Button_Toolbar_Redo (Common: no, Alpha: no)

Color for the toolbar icon "Redo".

Color_Button_Toolbar_PrintList (Common: no, Alpha: no)

Color for the toolbar icon "Print list".

Color_Button_Toolbar_PrintListBrowser (Common: no, Alpha: no)

Color for the toolbar icon "Print list via browser".

Color_Button_Toolbar_Delete (Common: no, Alpha: no)

Color for the toolbar icon "Delete".

Color_Button_Toolbar_Play (Common: no, Alpha: no)
Color for the toolbar icon "Play".

Color_Button_Toolbar_MultiTrack (Common: no, Alpha: no)
Color for the toolbar icon "MultiTrack".

Color_Button_Toolbar_EasyTrack (Common: no, Alpha: no)
Color for the toolbar icon "EasyTrack".

Color_Button_Toolbar_SingleTrack (Common: no, Alpha: no)
Color for the toolbar icon "SingleTrack".

Color_Button_Toolbar_TextEditor (Common: no, Alpha: no)
Color for the toolbar icon "Text editor".

Color_Button_Toolbar_DigaText (Common: no, Alpha: no)
Color for the toolbar icon "DigaText".

Color_Button_Toolbar_StoryEditor (Common: no, Alpha: no)
Color for the toolbar icon "Story editor".

Color_Button_Toolbar_OTM (Common: no, Alpha: no)
Color for the toolbar icon "OTM".

Color_Button_Toolbar_DBM (Common: no, Alpha: no)
Color for the toolbar icon "DBM".

Color_Button_Toolbar_FullTextSearch (Common: no, Alpha: no)
Color for the toolbar icon "Fulltext search".

Color_Button_Toolbar_DayOverview (Common: no, Alpha: no)
Color for the toolbar icon "Day overview".

Color_Button_Toolbar_ShowOverview (Common: no, Alpha: no)
Color for the toolbar icon "Show overview".

8.12.3.8 Class colors

Class colors are normally defined in the global registry in top-level key "ClassSettings". If you are logged in as "Admin" these settings are available in the settings dialog on tab "Colors". There you can also find a set of parameters to define, whether the class colors of the UI schemes should be preferred.

Choosing class colors is difficult, when your users work with light and dark UI schemes. On one hand, the color of some important classes is typically fixed station-wide (e.g. "our commercials are always bright green") and serve for having a good overview when looking onto a rundown. On the other hand, having the same color for a light and a dark UI scheme is not always possible – depending on the color, of course. With some colors it makes more sense to define different versions for different UI schemes.

The following color for classes can be defined: (for all: Common: yes, Alpha: no)

Color_Symbol_Class_Audio
Color_Symbol_Class_BatchList
Color_Symbol_Class_Cart
Color_Symbol_Class_Command
Color_Symbol_Class_Commercial
Color_Symbol_Class_Control
Color_Symbol_Class_DigaCutMTProject
Color_Symbol_Class_DigaCutSTProject
Color_Symbol_Class_Graphic
Color_Symbol_Class_Info
Color_Symbol_Class_Line
Color_Symbol_Class_Live
Color_Symbol_Class_Magazine
Color_Symbol_Class_Music
Color_Symbol_Class_News
Color_Symbol_Class_Project
Color_Symbol_Class_Promotion
Color_Symbol_Class_SequoiaProject
Color_Symbol_Class_Story
Color_Symbol_Class_Text
Color_Symbol_Class_Undefined
Color_Symbol_Class_Video
Color_Symbol_Class_VideoProject
Color_Symbol_Class_ZipArchive

If class-specific colors are neither specified in the ClassSettings nor in the UIScheme, then the color Color_Button_Def_SymbolNormal form the UIScheme is used.

9 FULL-TEXT SEARCH WITHIN BROADCASTSYSTEM

This chapter describes the new full-text search system, which is available since version 6.2 of BUS and BCS and version 2.0 of BCSSearch.ocx and ElasticIndex.dll. In contrast to the old system, which was based on dtSearch, the new system is based on ElasticSearch.

One big part of the system – the one, which is responsible for indexing the BCS content – is BUS index task. This task is described in detail in chapter 7.19 Index task. The index task generates a full-text index for a whole BCS tree and all data in it and then updates it with any changes by monitoring all BCS notifications.

Full-text search for clients is currently realized via ActiveX control BcsSearch.ocx for DigAIRange and BCSSearchService for any Web/custom clients to run queries on ElasticSearch via REST and Web Sockets so that you can play around with your own frontends (e.g. a web page). In order to run the index task and using BCSSearch.ocx, you need the ElasticIndex.dll.

9.1 Overview / Modules

The full text search within DigaBroadcastSystem is handled by the following modules: BUS(64).exe, ElasticIndex(64).dll, BCSSearch.ocx and BCSSearchService.exe.

9.1.1 BUS.exe / BUS64.exe

The index task of BUS is responsible for indexing the content of a BCS server. See chapter 7.19 Index task above for a detailed description.

9.1.2 ElasticIndex.dll / ElasticIndex64.dll

This is an internal module, which is used for communication with ElasticSearch, for both indexing and searching.

9.1.3 BCSSearch.ocx

This ActiveX control is the user interface for the full text search system which is displayed within DigAIRange. Like any other ActiveX control, it must be registered on every computer with regsvr32.exe. Requirements: for the communication with ElasticSearch the ElasticIndex.dll is needed. For reading settings the DigaSQL.dll is needed. All dlls should be located in the DigaSystem program directory or in the Windows system directory (not recommended).

9.1.4 BCSSearchService.exe

Provides REST & Web Socket communication for indexing & searching so that you can play around with your own frontends (e.g. a web page). For the communication with Elastic Search the ElasticIndex.dll is needed. For the communication with the BroadcastServer the IBCS.DLL is needed and must be registered. All dlls should be located in the DigaSystem program directory or in the Windows system directory (not recommended). See external documentation about installation & search request language for details.

9.2 Installation

9.2.1 Starting with ElasticSearch

ElasticSearch is a free and quite powerful and flexible engine for indexing text documents. It is available for Windows and for Linux and it can run as command line program or as service (daemon). You can download it on the home page of ElasticSearch:

https://www.elastic.co/

or the download page:

https://www.elastic.co/de/downloads/elasticsearch

The first steps for the installation can be found here:

https://www.elastic.co/guide/en/elasticsearch/reference/current/zip-windows.html

- For Windows you will get a zip file. This needs to be extracted to a location from which you want to run it.
- ElasticSearch contains a Java runtime from OpenJDK. You might want to use your own version of Java. Then you need to install it. Otherwise, you can go with the one from ElasticSearch.
- Start a command line and navigate at this location to the sub-folder “elasticsearch-X.Y.Z\bin”.
- Execute the command “elasticsearch”. At the first start this will set-up everything needed, will set a default configuration, and will try to start the ElasticSearch server. Hint: ElasticSearch uses a lot of memory Something like 8 GByte is quite normal. The standard port used by ElasticSearch is port 9200. This needs to be confirmed in the Firewall question, which pops-up. In the settings (see below) this port can be changed.
- By default, ElasticSearch sets the configuration to only allow secure communication via https and with authentication. It will print the password for the default user “elastic” on the command line. This password is generated anew with every start, and it is very annoying to always copy it and re-enter it in the configuration of clients like the BUS index task. For playing around with ElasticSearch and indexing it is probably better, to first start without security, certificates and without https. Therefore:
- Stop ElasticSearch by pressing Ctr-C in the command line and confirming with “Y”.
- Navigate with the Explorer to the sub-folder “elasticsearch-X.Y.Z\config” and open the file “elasticsearch.yml” with a text editor. Go down to the section “BEGIN SECURITY AUTO CONFIGURATION”.
- You should then change all entries, which make ElasticSearch to use security. For version 8.7 this would be:
  - xpack.security.enabled: false
  - xpack.security.enrollment.enabled: false
  - xpack.security.http.ssl: enabled: false
  - xpack.security.transport.ssl: enabled: false
  - All other entries below “# Enable encryption” should be commented out.
- Save the file and restart ElasticSearch. In case of the smallest error ElasticSearch will not start and you need to carefully check the configuration file and/or study the documentation on the home page of ElasticSearch!
- There is also a section “Paths” with the entry “path.data” in it. By default, this is a sub-directory of the ElasticSearch installation directory. In case this is not located on a solid-state disk or if you want to change the storage location of the index files, you should uncomment this value by removing the ‘#’ and then enter a sensible location. It is strongly recommended to choose a location on an SSD.
- Later, when you have finished trying out ElasticSearch, you will probably want to install it as a Windows service (Linux daemon). For a productive use it is recommended to use an extra server for ElasticSearch. One instance should be enough, even if you have multiple services and BCS servers. The content of one BCS goes into one “index” of ElasticSearch. In the unlikely case that you have many search requests per second you can think about running ElasticSearch in a cluster. You can also think about re-activating security. In this case you should create permanent users with permanent passwords.

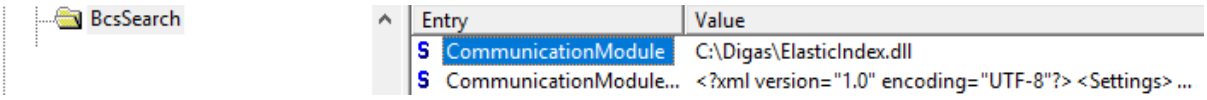
9.2.2 ElasticIndex.dll

- Copy the ElasticIndex.dll into the DigaSystem directory. If you want to run BUS as a 64-bit variant, you need to copy ElasticIndex64.dll into the DigaSystem64 directory.
- Copy one of the provided default mapping files into the DigaSystem(64) directory. See chapter 7.19 Index task above for a description of mapping files.
- In order to get the protocol, which is being written by the ElasticIndex.dll, you can activate the log destination “ElasticIndex” in the top-level key “Protocol” in the registry. Errors and all data being sent and received on the REST interface to ElasticSearch are logged.

9.2.3 BCSSearch.ocx

- A local DigaSystem base installation with a DigaSQL.dll and a setup for the DigaSystem registry is a prerequisite.
- Install ElasticIndex.dll (see above).
- Copy BCSSearch.ocx into the DigaSystem directory.
- Register BCSSearch.ocx with regsvr32 (this needs administrator rights).
- Only in case of doubt: the BCSSearchService.exe is **not** needed for BCSSearch.ocx.

The configuration for this module can be found in the top-level registry key “BCSSearch”. You need to configure the filename for the “communication module”. This module is responsible to make calls to the full-text search server (ElasticSearch). So far, only the 32-bit variant of the ElasticIndex.dll can be used:



The second parameter “CommunicationModuleConfiguration” can be used to specify all needed parameters for ElasticIndex.dll. You should not build the XML string yourself. Instead, navigate in Admin.exe to the registry settings of your BUS index task. Copy the content of the value “CommunicationModuleConfiguration”.

Using a single configuration file like this is only suitable if you have a single ElasticSearch server and a single index for the BCS content in it. If you have individual servers or indexes for each of your BCS servers, do not define this parameter. Instead, define the necessary communication parameters in the subkey “ElasticSearch” of each BCS key in the registry (->”Digas\PlanServer\<BCS name>”). This can then be different for each BCS, of course.

In the combination with DigAIRange (and perhaps in the future with DBM) BCSSearch.ocx provides a function to search for a specific element (or story, endpoint-story, group), meaning: where in all schedules does a specific element appear? For this feature it is an interesting question which properties of an element are the ones, which can be used for executing the search. If an element is originally coming from the database, it has a metadata field “DBRef”, consisting of the data source name, the table name and the reference number. This field is a very good candidate for the data to search for. Therefore, it is the default, too. In some cases, searching for the DBRef is not appropriate. This is the case, if an element is not from the database and does not have the DBRef set, of course. But there might be other cases, in which a different search could be done. For example, it could be desired to search for the MusicID in case of a music element. Therefore, BCSSearch.ocx has the possibility to specify a macro, which provides a search expression for any given element. This macro can be configured in this registry parameter:

`BCSSearch\SearchForElementMacro`

If the macro text is too long for entering it directly to the registry, it is recommended to put it into a text file and to reference the file with this parameter:

`BCSSearch\SearchForElementMacroFile`

The syntax for this macro is the usual one (like the TurboPlayer macros) and which is explained in chapter 7.1.10 (Macros and filter expressions). The main function, which must always be present must be named “**SearchExpression**” and it must return the search expression to be used. The macro is called with the metadata of the element, which can be retrieved by either accessing it with the function Data (<fieldname>) or with a local variable: %<fieldname>. (You need to use the field names from BCS system, not from the database.) The tagname of the searched node is %Tagname and its ID is %ID (or %LevelID for only the ID of the last level). If the program (=service) is known, it can be queried with %Program. This is the case if the function is called form DigAIRange for an element within a certain program or when called from the database when the metadata field “Program” is filled.

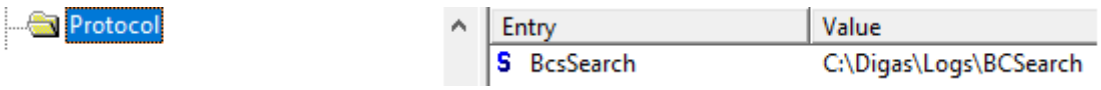
Here is an example for a quite complex macro:

```
function SearchExpression ( )
{
    %se = ""
    if (%Tagname $= "Group" || %Tagname $= "Story" || %Tagname $= "EndpointStory")
    {
        %se = strcat ( "Title: ", \{{"\}}, %Title, \{{"\}} )
    }
    else if ( %Class $= "Music" )
    {
        %se = UseMusicID ( )
    }
    else if ( %Class $= Cart )
    {
        %se = UseTitleFirstPart ( )
    }
    else
    {
        %se = UseDBRef ( )
    }
    %se = replacestr ( %se, "\", "\\" )
    return %se
}

function UseDBRef ( )
{
    return strcat ( "DBRef: ", \{{"\}}, %DBRef, \{{"\}} )
}

function UseMusicID ( )
{
    return strcat ( "Music_MusicID: ", \{{"\}}, Data ( Music_MusicID ), \{{"\}} )
}

function UseTitleFirstPart ( )
{
    %buf = %Title
    %brackIndex = strchr ( %buf, "[" )
    if ( %brackIndex >= 0 )
    {
        %buf = substr ( %buf, %brackIndex, left )
        %buf = trimstr ( %buf, right )
    }
    %buf = "Title: " + \{{"\}} + %buf + \{{*"\}}
    return %buf
}
```

You can see that the macro decides how to generate the search expression depending on the tagname and the class of the node. For non-elements a search for the title is done. The first possibility for an element (also the default of BCSSearch.ocx) uses the DBRef. For a music element the field “Music_MusicID” is being used for searching. The example shows a special handling when a cart element should be searched. It searches for the title, but first removes anything behind a left bracket ‘[‘ (including the bracket itself and then trimming the string). The idea is that the title might contain something like a branding version or e.g. info about variants of a title enclosed in brackets. This is first cut off, to find all the variants, which appear somewhere in the schedule.

Please note the usage of the intrinsic replacestr function to replace a possibly present single backslash by two backslashes. This is an escaping, which is required by ElasticSearch.

BCSSearch.ocx can write a log with some information. To enable it, define key “Protocol\BcsSearch” like any other log in DigaSystem:



9.3 Search Expressions

This chapter describes search expressions as they can be used with ElasticSearch. Internally, the so-called “Query string query” is used. You can find detailed information on the documentation pages of ElasticSearch: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html#query-string-syntax

9.3.1 Words

The simplest query is made by specifying one or multiple words, like:

merkel macron

This will find all nodes, which contain either “merkel” or “macron”. Capitalization does not matter, because all data is stored as lower case in ElasticSearch and searches are done case-insensitive.

If a specified word must appear in the text you need to prefix a plus, like:

merkel macron +draghi

If a specified word must not appear in the text, prefix a minus:

Angela -Merkel

9.3.2 Phrases

Use quotation marks to indicate a phrase. You can use a phrase anywhere in a search request. Example:

apple “fruit salad”

9.3.3 Field names

You can search for text in specific fields by prefixing a search text with the field name and a colon. Therefore, you need to know the exact name of the field, case-sensitive! A list of all regular fields can be found in BCSTechManual, chapter 2. For some common fields there is a helper dialog, which can be opened by pressing the “…” button right of the box for the search text. In this dialog you can enter a search text for a specific field and if you close the dialog, the box for the search text will be filled accordingly.

Here is an example how to search for a specific author:

News_Author:prosche

You can also combine the search in a specific field with logical operators:

Title:(Merkel AND Draghi)

9.3.4 Wildcards (* and ?)

A search word can contain the wildcard characters ***** and ?. A ? in a word matches any single character, and a ***** matches any number of characters. The wildcard characters can be in any position in a word. For example:

appl*

would match apple, application, etc.

cipl

would match principle, participle, etc.

appl?

would match apply and apple but not apples.

ap*ed

would match applied, approved, etc.

Using the ***** wildcard character near the beginning of a word will slow searches significantly.

9.3.5 Fuzzy search

You can run fuzzy queries using the ~ operator:

quikc~ brwn~ foks~

The query uses the Damerau-Levenshtein distance to find all terms with a maximum of two changes, where a change is the insertion, deletion or substitution of a single character, or transposition of two adjacent characters.

The default edit distance is 2, but an edit distance of 1 should be sufficient to catch 80% of all human misspellings. It can be specified as:

quikc~1

The main view of BCSSearch has an option “Fuzzy search”, too. You can select “no / exact”, “low” or “high”. The option “low” corresponds to “~1” and “high” corresponds to “~2”. If you select “low” or “high”, BCSSearch will append the corresponding ~ operator to each word in the search expression automatically. This works well for search expressions, which are not overly complicated – including logical expressions. In case the automatism seems to fail, you should switch the fuzzy search option to “no / exact” and specify the ~ operator as you need it.

Important: **do not** mix fuzzy search and wildcards. ElasticSearch cannot handle both at the same time and will probably decide to use only one of them.

9.3.6 Boolean and logical operators

By default, all terms are optional, as long as one term matches. A search for foo bar baz will find any document that contains one or more of foo or bar or baz

The preferred operators are + (this term must be present) and - (this term must not be present). All other terms are optional. For example, this query:

quick brown +fox -news

states that:

- fox must be present
- news must not be present
- quick and brown are optional - their presence increases the relevance

The familiar boolean operators AND / &&, OR / || and NOT / ! are also supported but beware that they do not honor the usual precedence rules, so parentheses should be used whenever multiple operators are used together. For instance, the previous query could be rewritten as:

((quick AND fox) OR (brown AND fox) OR fox) AND NOT news

ElasticSearch is a little bit picky with search expressions. It requires that the words AND, OR and NOT are written in upper-case and that the !-operator directly prefixes the word without spaces. BCSSearch facilitates this. It automatically converts the logical words to upper-case and removes any space behind the !-operator. In case you encounter problems with the automatic improvement of the search expression by BCSSearch, switch off the fuzzy search option, use the operators &&, || and !, and don’t insert spaces behind the ! operator. If you do that, BCSSearch will not modify the search expression any longer.

9.3.7 Regular expressions

Regular expression patterns can be embedded in the query string by wrapping them in forward-slashes ("/") and can be used for the global or field specific search:

Title:/joh?n(ath[oa]n)/

Regular expressions are quire powerful and complex. We cannot describe them here in detail. If you are interested in these details, please have a look into this web page:

https://www.elastic.co/guide/en/elasticsearch/reference/current/regexp-syntax.html

9.3.8 Ranges

Ranges can be specified for date, numeric or string fields. Inclusive ranges are specified with square brackets [min TO max] and exclusive ranges with curly braces {min TO max}.

All elements with a duration between 30 and 60 seconds:

Time_Duration:[30000 TO 60000]

All elements with a duration greater than 30:

Time_Duration:[30000 TO *]

Alternatively, you can write:

Time_Duration:(>=30000 AND <=60000)

Time_Duration:>=30000

If you use curly braces { / } instead of brackets [/] the specified values are not part of the interval (=exclusive).

Hint: The parsing of ranges can be complex and error prone

9.3.9 Proximity searches

While a phrase query (e.g. "john smith") expects all of the terms in exactly the same order, a proximity query allows the specified words to be further apart or in a different order. In the same way that fuzzy queries can specify a maximum edit distance for characters in a word, a proximity search allows us to specify a maximum edit distance of words in a phrase:

"fox quick"~5

The closer the text in a field is to the original order specified in the query string, the more relevant that document is considered to be. When compared to the above example query, the phrase "quick fox" would be considered more relevant than "quick brown fox".

9.3.10 Reserved characters

If you need to use any of the characters which function as operators in your query itself (and not as operators), then you should escape them with a leading backslash. For instance, to search for (1+1)=2, you would need to write your query as *\(1\+1\)\=2*

The reserved characters are: + - = && || > < ! ( ) { } [ ] ^ " ~ * ? : \ /

Failing to escape these special characters correctly could lead to a syntax error which prevents your query from running.

Note. < and > can't be escaped at all. The only way to prevent them from attempting to create a range query is to remove them from the query string entirely.

10 REUSING BROADCAST CONTENT

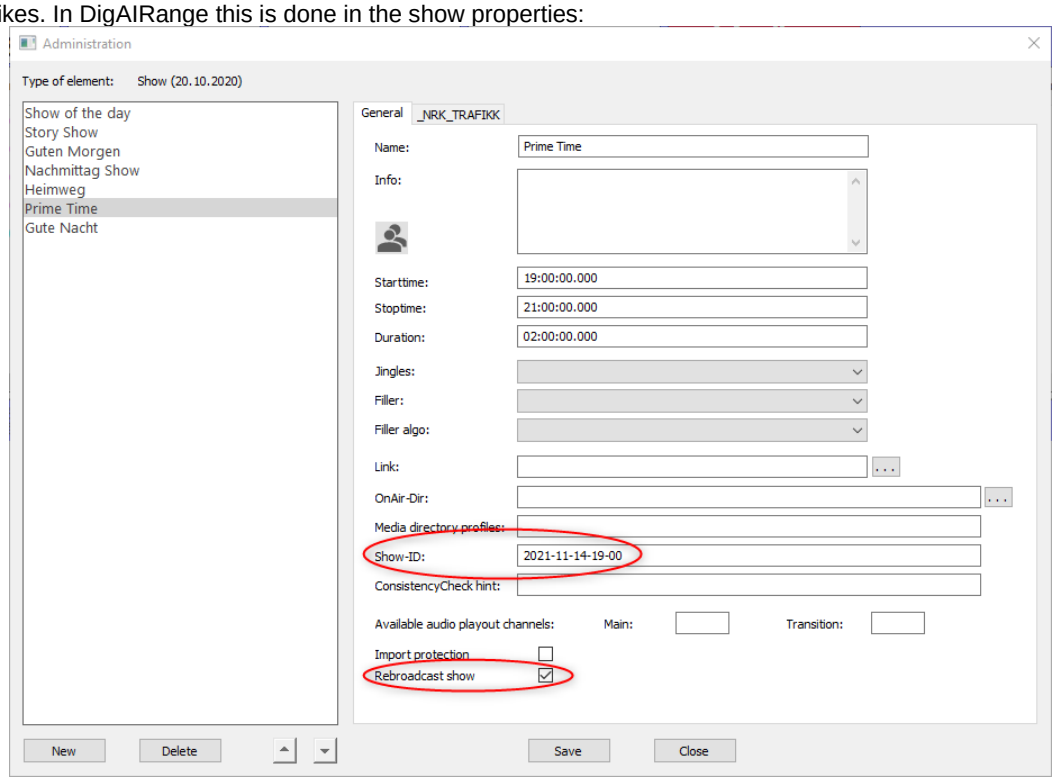
10.1 What is re-use ?

In radio broadcasting it happens that you want to broadcast a whole show or parts of it more than once. Typically, something is sent on the main service during the day and it should be used to fill a second minor service or to fill the same service during the night operation and typically further broadcastings should run automated. In DigaSystem we call this re-use or re-broadcast. In case the show originally run fully automated, it will be possible to do re-broadcastings automatically. In case the original show was run by a presenter there will be live parts in the rundown. Therefore, these live parts have to be replaced by audio parts before the show can be re-used in automatic mode. DigaBroadcastSystem has some features which support these workflows.

Please be aware that the re-broadcast task of BUS is completely separated from the re-use workflow with DigAIRange - though you can use both feature sets for similar problems.

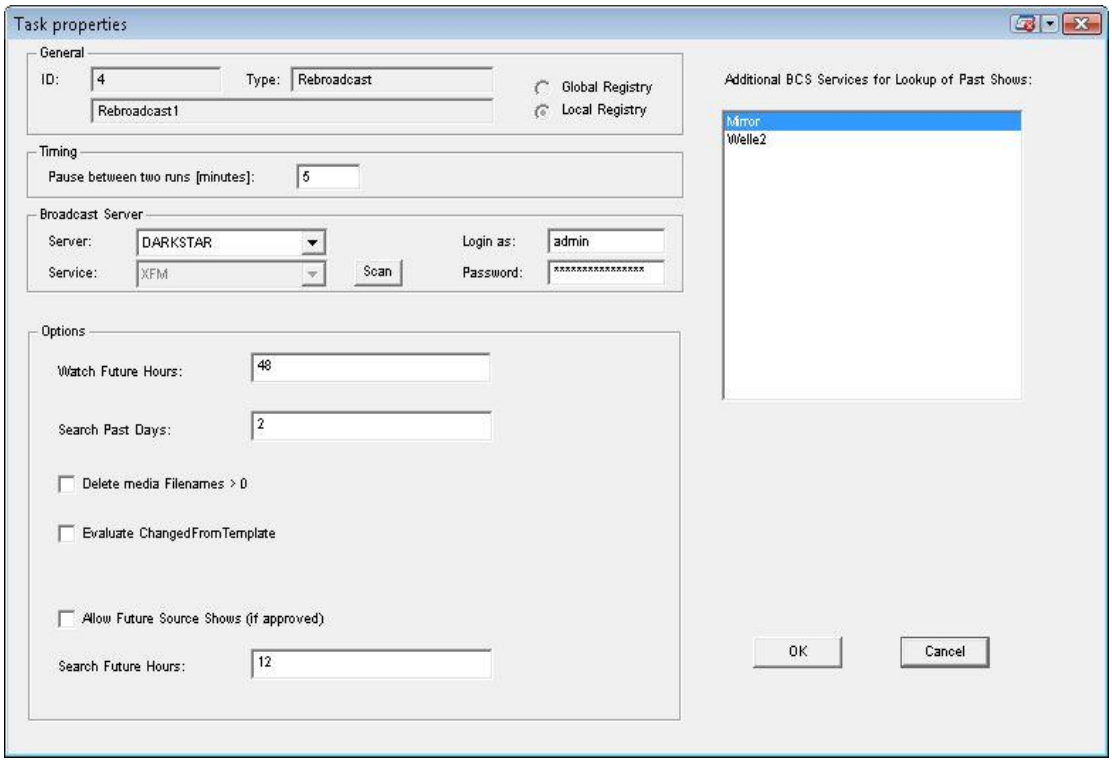
10.2 Re-broadcast task of BUS

Our multi-purpose background server BUS has one task which is called "Rebroadcast". This task can be used for re-scheduling a fully automatic show. What does this task do ? It runs permanently in the background and whenever a show was completed (the end time was reached and all elements in it were sent or skipped) the task checks future shows whether they are intended as a rebroadcasting of the completed show. If the task finds such shows it copies the whole material to the show in the future and adopts the content to make it sendable again. For example, the tasks resets the send state and adopts the times to the destination show. The task identifies the destination shows by two things: first a flag "Rebroadcast show" must be set in the properties of the show. And second the source and the destination show must have the same show ID. The show ID can be specified by the administrator as he likes. In DigAIRange this is done in the show properties:



We recommend to use the start timestamp of the original show as show ID. All rebroadcasting shows must have the same show ID and in addition the flag "Rebroadcast show" must be set.

The BUS task can be configured with the following dialog:

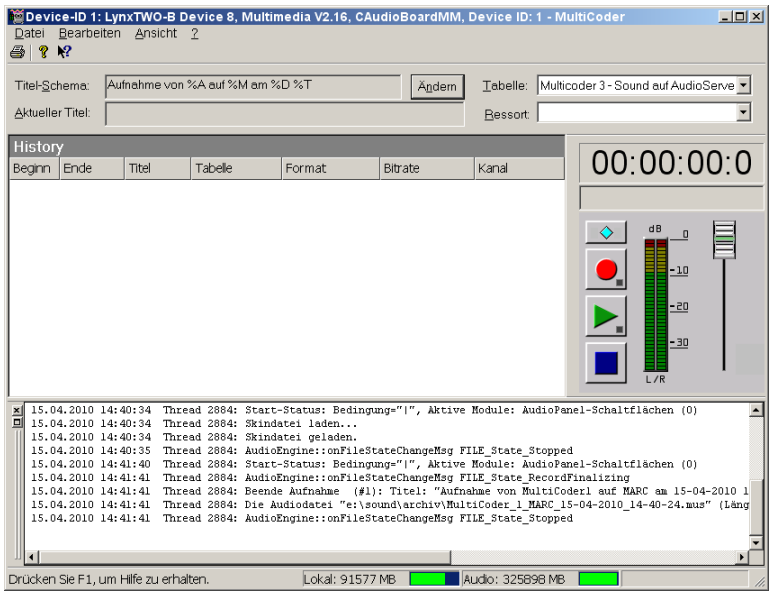


There is not much to configure here. Mainly you have to define the BCS/service to operate on, additional services (which allows to copy shows from one service to another one) and the future/past interval. The intervals are necessary to avoid that the task has to scan for possible source and destination shows endlessly to the past and to the future. See the BUS chapter for more information about the individual settings.

10.3 Recording of live elements

For shows which have live parts it is necessary to record these live parts. This makes it possible to replace placeholders for the live parts in the rundown with real audio material later. It depends on the type of service which audio signal should be recorded. For a pure news or classical service which has gaps between the elements (or at least no crossfades) the on-air signal can be recorded. If the elements overlap it is necessary to record the live signal individually. The mixer console must be able to build a second sum signal from individual mixer sources, e.g. the microphones or other external devices. Such a signal must not contain the audio which is played from hard disk.

For the recordings we recommend our tool MultiCoder:



Starting and stopping the recordings can be done in three different ways. (1) MultiCoder can record continuously - meaning: the whole show. (2) It can react on a fader contact - meaning: the recording is done as long as at least one microphone fader is open. (3) TurboPlayer can start/stop MultiCoder to record only the live parts. Each new recording will create a new element in the DigaSystem database. More details about MultiCoder can be found in its manual. More details about controlling MultiCoder via TurboPlayer can be found in the TurboPlayer-TechManual.

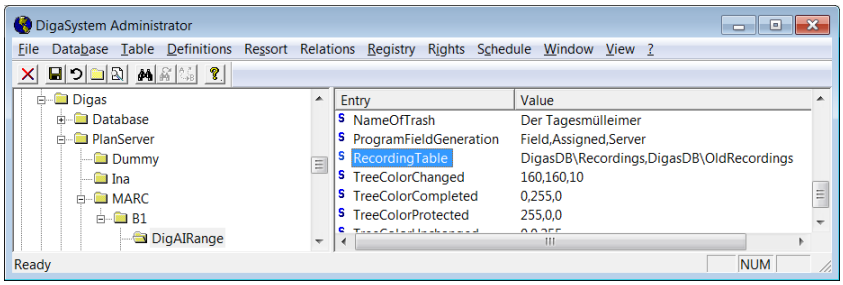
10.4 Re-Use workflow in DigAIRange

In our scheduling program DigAIRange it is possible to prepare a sent show for its re-use. The workflow is like this:

1) You copy elements from the source show to the destination show. If the whole show should be re-used you can copy the whole show in the tree view. Or you can select individual parts in the show view and drag them into the destination show.

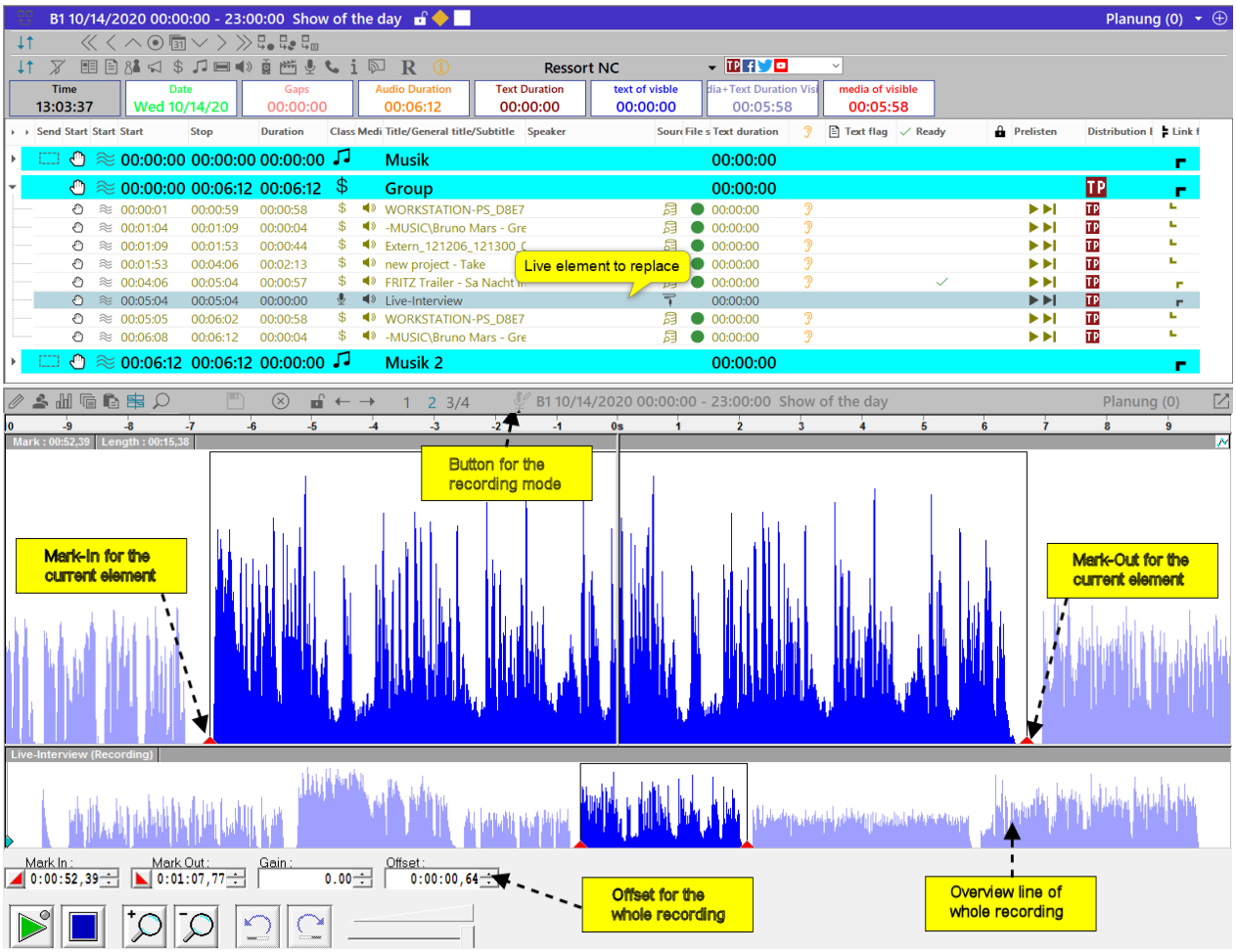
2) You open the CrossfadeMixer in DigAIRange and switch it to recording mode (see picture on step (4) below).

3) You drag the first entry you want to replace with the recording into CrossfadeMixer. DigAIRange will automatically look into the database to find the right recording and will load it into CrossfadeMixer. How does DigAIRange know which database element it should load ? First of all you have to tell DigAIRange in which tables it should look for the element. This is done in the registry:



In addition you can filter the database elements by the parameter "RecordingPrefix" to achieve that only elements with a specific starting text in the title are usable. DigAIRange scans the specified tables for a recording which encloses the time of the treated rundown element. The start and stop time of the treated element is stored in special fields by TurboPlayer and these times are kept when the element is copied to a new show and reactivated. Therefore DigAIRange can find the right database element. If no database recording for the searched time is found an error is displayed, if multiple recordings are found it is undefined which one will be used.

4) Now you can adjust the mark values in CrossfadeMixer:



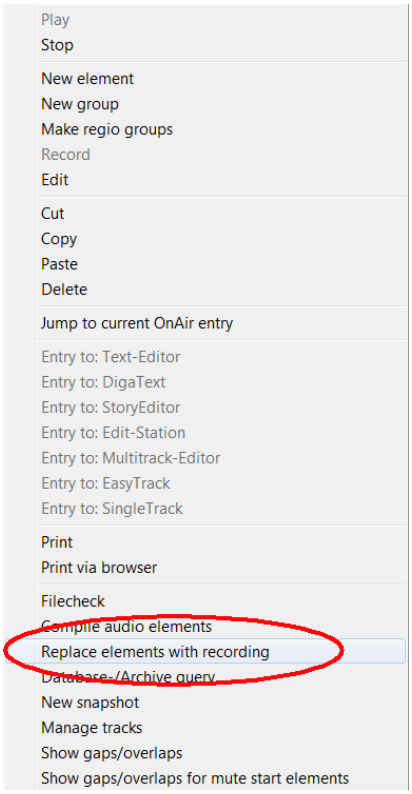
DigAIRange will place the markers to the best guess it has. But for technical reasons it is possible that the correct position is missed a little bit. Therefore, you have the possibility to move both markers until they pick out the desired piece from the recording. A typical problem is an offset of the two system times on the TurboPlayer and the MultiCoder computer. This results in an offset between the times in the rundown data and the recording itself. Fortunately, this offset remains constant for the whole recording. Therefore, you can adjust the offset value with the first element you pick from the recording. For all further elements from the

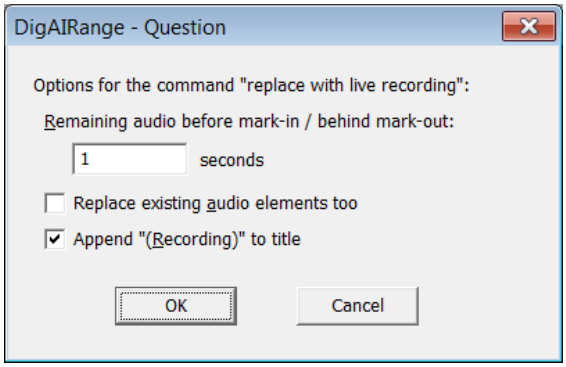
same recording it is likely that you need not make any more adjustments, because the offset alone will correct the marker positions. (Technical hint: the offset is stored in the DB field "Location" as "RecOffset").

5) Now you press the save button. This will not replace the placeholder in the rundown with the recording. Only the changed metadata (markers and offset) will be written to the rundown data.

6) You continue with every element in the rundown you want to replace. You can omit this work if you trust that the correction of the offset value for the first element is sufficient. We recommend to check every live element nevertheless.

7) Next you select the range of elements you want to replace and you call a DigAIRange function which will replace every treated element with the marked piece from the recording:





DigAIRange will step through all selected elements and will replace them with pieces from the recording. You have some options:

- DigAIRange can make the piece it cuts from the recording bigger than the interval between the two markers. The mark-in and mark-out value DigAIRange computes for the new element will be like they were set in the steps above. But there will be some silent audio around the markers. This can be useful for a later correction.
- Normally DigAIRange will treat only external elements. Internal elements can be replaced too. This is only useful if your recording contains the internal elements too. Depending on the way how the recording was done a quality loss might occur.
- DigAIRange can append the string "(Recording)" to each replaced element to identify them easily.

If you are finished you have a show which only consists of internal audio elements and therefore can be broadcasted fully automatic. As all the steps above are done within DigAIRange in the scheduled show you can also make use of all scheduling capabilities. This means: of course it is possible to move or exchange elements, to modify start attributes, to make nice crossfades, … whatever you need to do for scheduling. Typical examples are: you want to replace a news block with some music elements for the night show. Or someone told the current time in a presentation which therefore must be exchanged. As you always treat individual elements in this workflow you can do all these modifications which makes this solution quite flexible.

10.5 Metadata track

The metadata track is a feature, which is available since version 5.5 of the DigaBroadcastSystem. It allows another workflow for reusing broadcast content. The feature is intended to be used when you want to play a live recording of a show later. Let's assume you have one-hour shows and you are recording them in the background into one-hour audio files. Now you can schedule these recordings in a later show, of course. However, you face the problem that the one-hour audio does not contain any information about the original elements, which were played. So if you want to feed an RDS encoder or put to your web page what element is currently playing or if you need to report what has been played for computing royalties, this is not possible.

But it is possible to use the multi-track concept of the BroadcastSystem. On the main track 0, you can put the one-hour recording. On an additional track (e.g. number 1) you could put live elements, which represent the originally played elements. "Live" means: there is no audio behind these elements, they are only going to be played virtually by TurboPlayer. Nevertheless, they can contain some (or even all) of the metadata of the original elements. That's the reason why such a track is called a "metadata track".

Fortunately, you do not have to fill the metadata track manually. There is a BUS task, which can handle it for you, the "Metadata scheduler" task. It is described below but also in the corresponding chapter about BUS above. In addition, you need a feature within TurboPlayer which is called "Embedded start". It is needed because the live elements have to be started at the time when the original elements were played within the live recording. Therefore, the Metadata scheduler will enrich the metadata of the live recording with references and start/stop information for the corresponding live elements in the metadata track. TurboPlayer can evaluate this information and can trigger the start of the live elements accordingly.

Now let's have a walk through the workflow and all the modules and individual features which contribute to make the workflow possible.

10.5.1 Subclips

Subclips are a feature of the DigaSystem database. You can assign multiple subclips to each file of a database item and each subclip can have metadata like a title, a comment, a start position or duration. Subclips are used to store different sort of information, e.g. different scenes in videos, cut markers of an editor or beat markers. Therefore, each subclip has a number field "Layer" which allows differentiating the types of subclips. (Hint: the field "Type" of a subclip, which was intended to be used for this purpose, is currently not in use because the DPE system does not support this field.) Within this workflow the subclips are used to store metadata of the original elements played during a live recording. There is a DPE workflow called **"AsRunLogImport"**, which can fill the subclips by evaluating the BCS protocol of a playing show.

You need 5.5 versions of the BroadcastSystem modules in order to get the subclip information from the database to the BroadcatSystem. Older versions do not transfer the subclips from DBE/DBX to BCS-XML. Only the subclips of the main file are converted. If you have a new version, you can see the subclips in the "More" submask of DigAIRange:

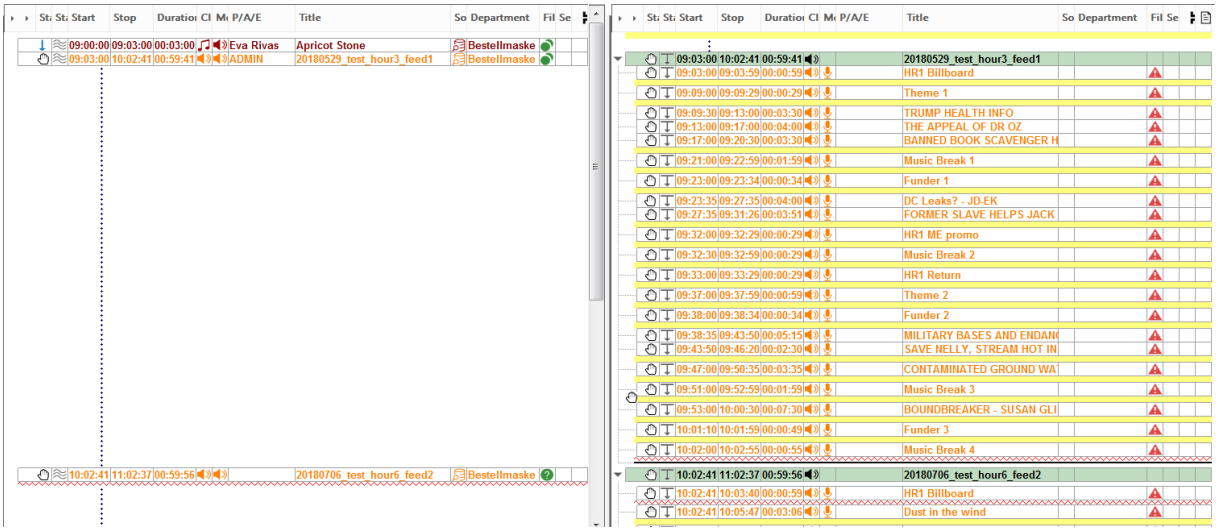
Field	Value
Subclips	
Clip.1.Comment	{"showdate":"20170905","feedtime":"1600","showtitle":"atc","subcliptitle":"_open3h1f1","itemtype":"OPENRETURN"}
Clip.1.Duration	00:00:59.000
Clip.1.Layer	2
Clip.1.Start	01:00:00.000
Clip.1.Title	_open3h1f1
Clip.2.Comment	{"showdate":"20170905","feedtime":"1600","showtitle":"atc","subcliptitle":"kinobe_sundog"}
Clip.2.Duration	00:00:28.494
Clip.2.Layer	3
Clip.2.Start	00:29:30.021
Clip.2.Title	Kinobe Sundog
Clip.3.Comment	{"showdate":"20170905","feedtime":"1600","showtitle":"atc","subcliptitle":"_funder3h1f1","itemtype":"FUNDER"}
Clip.3.Duration	00:00:49.000
Clip.3.Layer	2
Clip.3.Start	00:58:10.000
Clip.3.Title	_funder3h1f1

As you can see, each subclip has the essential information about its start and its duration. The optional field "Layer" can be used to pick out the subclips, which are relevant for this workflow. As the subclip information within the database is not extensible, it is necessary to miss-use the comment field in order to store arbitrary additional metadata. Therefore, the comment contains a JSON string with some fields. The BUS task "Metadata scheduler" is able to evaluate the comment field and convert the field names into BCS field names. In the example above you could configure that the JSON field "subcliptitle" is being used as "Title" for the created live elements. In this case, the field "Title" of the subclip itself is not used and therefore is optional.

10.5.2 Metadata scheduler

The Metadata scheduler is a BUS task, which evaluates the information for a live recording and creates metadata elements in a different track. The task cannot only evaluate the subclip information of an element, it can also perform a direct lookup of the BCS protocol data of the time when the live recording was done. This task with all its options is not described here but in the chapter “Broadcast Utility Server / Metadata scheduler” above.

If you run the Metadata scheduler, it will permanently monitor the current and addition future shows. If it detects a live recording (through a configurable filter expression), it creates live elements in another, configurable track. You can use the sync view of DigAIRange to visualize the main and the metadata track side-by-side. For the example above this could look like this:



You can see that the Metadata scheduler has created many live elements in the second track. Each represents an originally played element. The scheduler has assigned manual / fixed start attributes to each of the elements. For the TurboPlayer start this does not matter. An embedded start is handled independently of the start attributes. But the fixed starts make it clearly visible at which time the live element should start. If a fixed start does not fit your needs, there is an option for the BUS task to make it create floating elements. The task automatically switches to floating elements if a computed start time is outside of the show borders. If there are any changes in the metadata of the recording element (e.g. start time, mark-in value or the subclips) the Metadata scheduler will automatically update the live elements so that they reflect the current data of the recording.

It is even possible to play a recording, which is still being recorded (read-while-write). During the ongoing recording, the DPE workflow creates new subclips and the Metadata scheduler creates new live elements from the new subclips. There are two drawbacks: First, the Metadata scheduler reads the subclip information from the BCS but the DPE workflow writes the information into a DigaSystem database. So far, there is no active database lookup of the task. Therefore, it is necessary to perform the database to BCS synchronization with a different task, the AutoReplace task. (Hint: in future versions a direct database synchronization might be implemented.) The second drawback is that a read-while-write workflow must be set up carefully. Especially in BCS-protocol-lookup-mode, the task relies on precise information when the live recording was originally done. Therefore, metadata fields like the real times, the record times, the database create time and the duration and the mark values are evaluated. This is information, which is not very stable as long as an element is just being recorded or it might be modified by many operations and by user interactions. If this information is wrong or is lost, the BUS task will pick the wrong protocol elements or it might not find any at all. Take special care about the file state “Recording”. This file state should be kept if you re-schedule an ongoing live recording, because the BUS task will then pick all protocol elements up to the current time. It does not matter what duration or stop time was originally stored for the recording. On the other hand, you must also assure that the file state “Recording” is later being reset for the re-scheduled element, too. This can be achieved with the BUS task “FileCheck”, for example. Be aware that the information needed by the task can get easily lost, even if you are not working with recordings. For example, if you copy a live recording from one DigaSystem table to another one, a new create time might be assigned to the copied element and therefore the original recording start time gets lost.

One hint about the mark-in: if you modify the mark-in of the recording element all live elements which are in front of the mark-in position will be removed from the metadata track. Or if you schedule a new recording with mark-in these live elements will not even be created. This is different for the mark-out, because the mark-out is not evaluated by the Metadata scheduler. The reason is that TurboPlayer can be switched to ignore-mark-out mode. Then the recording element will be played behind the scheduled mark-out and the live elements after the mark-out will be needed. As it is not known in advance whether ignore-mark-out is active or not, the task creates the live elements behind the mark-out in any case.

Embedded start

The “EmbeddedStart” is a global mode of TurboPlayer. It must be active to make TurboPlayer automatically start the live elements on the metadata track.

Here is how embedded starts work:

- The Metadata scheduler assigns a newly generated GUID to each live element. It is stored in the field “EmbeddedStartID”.
- The same GUID is stored in the corresponding subclip of the recording element in the field “Clip.<n>.EmbeddedStartID”.
- When loading the recording element TurboPlayer will convert each subclip of the configured layers (-> parameter “TurboPlayer\SubclipLayersToStart”) into a control at the correct position within the recording. Normally, you do not see these controls but if you create a dump you will see it in the output for the RundownKernelLists. For example:

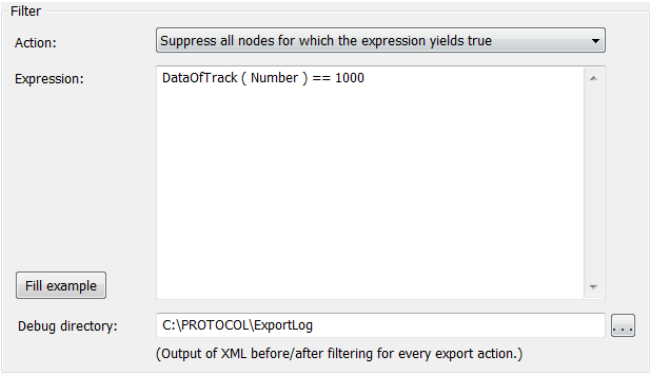
```
*******************************************************************************
2018-07-10 14:09:58.953 Dump of RundownKernelLists:
*******************************************************************************
Dump of RundownKernel - ShowList:
...
2018-07-10 14:09:58.955 ----------- RundownKernel - ShowList Element 1 -----------
...
    Control(1):
      Operation = TP_StartEmbedded
      Title = Dust in the wind/5e5e907d-7fe4-45e8-b8ae-a823c01fff94
      Reference = MarkIn
      Offset = 30000
      Parameter = *,"5e5e907d-7fe4-45e8-b8ae-a823c01fff94"
      Position = 30000
      Timer event = 0
      Execution time =
      ControlID = 2
      Executed = No
```

- When the recording element starts, TurboPlayer generates timer events for each control at the computed position.
- If you use the mode “Lookup of metadata in BCS protocol” of the metadata scheduler task, the task does not create subclips, it directly writes controls which can then be executed by TurboPlayer. Nevertheless, it is necessary to activate the embedded start mode.
- When the timer event is being triggered and the EmbeddedStart mode is active TurboPlayer will execute the macro command TP_StartEmbedded (*, <EmbeddedStartID>). Hint: this macro command is also available as a regular macro command. TurboPlayer will look through the other rundowns whether an element with the specified EmbeddedStartID can be found. If yes, this element is started. If no element is found, this is silently ignored. Nevertheless, you can use the two internal event triggers “EmbeddedStartHit” and “EmbeddedStartMiss” for these two cases.

It might be possible that the recording elements are not started in the sequence in which they were scheduled. This can be a problem if many live elements were generated for each recording element: The command TP_StartEmbedded works only for elements within the preload range of the kernel. The number of preload elements is normally dependent on the number of available MultiPlayer slots and you might not be able to configure a very high number of preloaded elements for the metadata stack. But as live elements need not be preloaded by MultiPlayer anyhow, you can circumvent the slot limitation. You can set the parameter “TurboPlayer\CheckSlots” to FALSE. In this case TurboPlayer will no longer check whether there are enough MultiPlayer slots available for the configured preloaded elements. You must be careful with this setting, of course. It can only be used, if you are sure that it’s a lot of live elements which must be handled and not audio (or video, picture) elements.

10.5.3 Export

The final topic for completing the workflow is to generate an export, which does not contain the recording element, but instead the live elements. This can be achieved with the settings of the export in DigAIRange or in BUS export task:



The “Action” and the filter expression are described in chapter “Broadcast Utility Server / Export Task”. The example shown would suppress the output of any nodes of track 1000 – the regular protocol track. If you have only recording elements in this track, the filter can work as intended. If you have a mix of regular elements and live recordings on the track, the filter must be adopted to filter out only the recordings.

This example also assumes that the receiving export module and/or the receiver of the data can handle multiple tracks – here track 1000 and the protocol track for the metadata track, e.g. 1001. If this is not the case, it might be necessary to combine the protocol data into a single track. This is not done during the export, this must already be handled by TurboPlayer. You can configure the used protocol track for each rundown list and therefore you can let TurboPlayer write the protocol for the main and the metadata track into the same protocol track. In this case the filter above must be adopted, of course.

If you wonder about the “Debug directory” – this setting is only present in the dialog of DigAIRange. It allows you to test filter expressions whether they really filter the intended elements. You can configure a directory and DigAIRange will write the BCS-XML before and after filtering into this directory. Each time an export two of these files are being written with a unique timestamp.

11 REGIONALIZATION AND REMOTING

11.1 Introduction

There are two main types of regionalization which are supported by DigaBroadcastSystem. The first type is the regionalized playout from one location. That means you have only one location with one studio but there are multiple outputs from this studio. Each output is being sent in a regional part of your service area. Typically, all outputs are identical most part of the time, but there are small time windows in which the content is different for each output, e.g. during regional news or commercial breaks. The second type is the playout from different locations. Typically, most of the time a service is being sent from a main location, but there are time windows during which the service is split and each regional studio is servicing its local area. The term "remoting" refers to this type of regionalization, because many issues are about remote links, remote data and file handling and remote control.

In DigaBroadcastSystem these two types are supported and this chapter describes how to configure the system for regionalization. This manual describes the general concept, the representation in the data, file handling and BCS / BUS / DigAIRange configuration. It does not contain details about TurboPlayer, please refer to the TurboPlayer TechManual for more information.

11.2 Data handling

11.2.1 Shows vs. Tracks

Each day within a service is separated into multiple shows, which have fixed start and end times. Shows can be used for remoting.

Each show contains multiple tracks. If you do not use regionalization you have most probably just two tracks for each show: the main schedule track and the protocol track. However, multiple schedule tracks can be created (Schedule tracks have a track number < 1000). This can be used to create regional tracks related to the main schedule track. Tracks can be used for regionalized playout and for remoting.

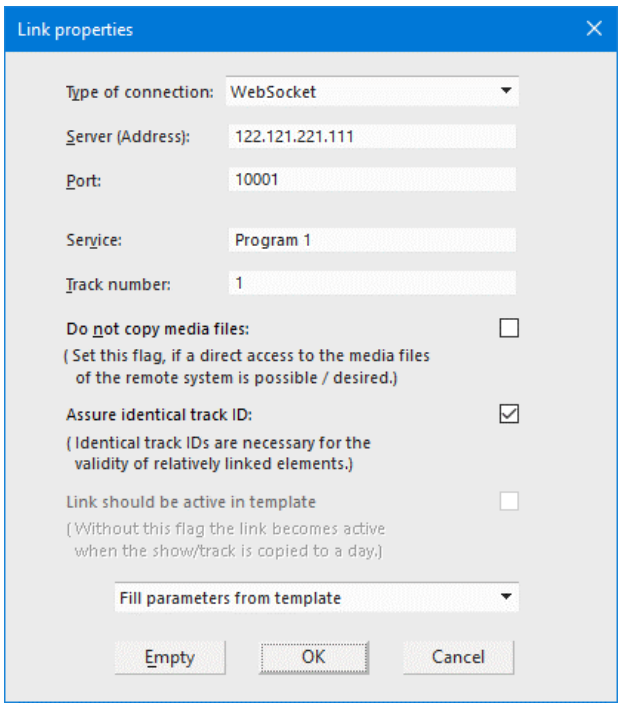
11.2.2 Remote links

If you have multiple locations it would be quite easy to get “remote” access by treating the whole system as a large, single network. That means each location has its own file server and BroadcastServer and each client has direct access to all servers. As all clients use TCP/IP for communication you should not have any problem to set up such a topology. This approach has some drawbacks: (1) The performance might be poor because many clients access data and files via a single WAN line. (2) The security can be a problem because it is not always desirable to open a local network for all remote clients. (3) And the reliability will be degraded if the on-air playout is done with files from remote locations. Therefore, the idea is to make all required data and files locally available in each location. File handling is done by BUS and is described later.

Remote data handling is done by BCS with a feature called "remote links". For each show or track you can set up a remote link to a show or track on a different BCS. If such a link exists, the data in this show / track will be identical to the data on the linked BCS. There is one BCS on which the show / track does not have a remote link, this is called the "host" BCS. The show / track on one or multiple BCS have a link to the host, these are called the "slave" BCS. If any client wants to access a show / track with a remote link, a slave BCS connects to the host BCS and relays any information between the client and the host. You can have read-only access on the slave system. In this case the data of the host is mirrored to the slave and the slave BCS will pass on any notifications about data changes of the host to its own clients.

The advantages compared to a large, single network topology are clear: (1) The data is transferred only once via WAN, though there may be many local clients accessing the data. (2) The network of the host system need be opened for the remote BCS only. (3) And the XML data will be locally present. If the WAN is interrupted you can go on working with the local data - though write access is not possible unless you delete the remote link. Generally, you can decide with a global BCS parameter whether writing of remote data should be allowed or not.

For the administration of remote links you use DigAIRange. If you click on the "…" button behind the link field of the manage dialog of either the shows or the tracks, the following dialog appears:



- **Type of connection** is one of: "SeplProtocol", "WebSocket" or "Secure WebSocket". SeplProtocol is the old protocol used by BCS versions before 6.x and/or by a 32-bit BCS. With a version 6.x or newer of BCS you can / and with a 64-bit variant of BCS you must use one of the WebSocket types.
- **Server** is the DNS name or the IP address of the host BCS. It is NOT one of the registry keys below Digas\PlanServer, because BCS does not evaluate this key.
- **Port** is the port number of the host BCS – default is 10000.
  If the remote link should point to a master/buddy cluster, you enter only the IP address and the port number of the master BCS. The information for the corresponding buddy BCS is specified in the DigaSystem registry in key BroadcastServer\<Config>\RemoteBuddyData (also see parameter description for BCS above). You need to create a registry value which has the address and port of the master BCS as value name and the buddy address and port as data of the value. In the example above it could be something like: “122.121.221.111:10000=122.121.221.112:10000”.
  It is also necessary to enter the login credentials into the registry, into the key: BroadcastServer\<Config>\RemoteLoginData. See the chapter about the BCS parameters above for more information.
- **Service** is the service (program) you want to connect to. It need not be identical to the service name of the own system. Attention: Please enter the name of the service, not the ID!
- **Track number** is only available if you call the dialog for a track. It is the track number of the host track. It need not be identical to the track number of the local system.
- **Do not copy media files:** This flag is described below in the chapter about remote file handling. Generally spoken: set this flag if you do not want the media files to be copied from the host to the slave system.
- **Assure identical track ID:** This flag is only available if you call the dialog for a track. If set, BCS will assure that the sub ID (a part of the XML field <ID>) for the linked track is identical on the host and on the slave system. This is needed for relative links of groups and elements (see below) because such a relative link consists of a relative ID to the linked element / group. (If the track IDs were not identical, the relative ID on the host and on the slave system would be different.)
- **Link should be active in template:** This flag is only available for day templates. By default, a show or track within the day templates will not be linked to the remote system. This is only done when the template is inserted in day in the normal timeline. If you want the link to be already active in the templates check this option.
- The drop-down field contains the link templates. You can define such templates in the DigaSystem registry in the key DigAIRange\ShowLinkTemplates or TrackLinkTemplates, respectively. The subkeys contain values for each field in the dialog. This makes it much easier to set a new link without entering all parameters each time.
- The button "Empty" can be used to clear all fields. This removes the existing link.

You will have probably noticed that you cannot specify a destination show / track for the remote link. The reason is that you cannot link to any show on the remote system you want. As the data on the slave system is only a copy of the host data, the time information is identical and therefore the times of the show (track) must be the same too. That means: the start time, stop time and duration of the show on both systems must be identical! In order that the slave BCS can find the right show automatically, you need not specify the destination show. In this way, it is also possible to set links in the show / day templates already: when the BCS uses them for an empty day, the remote links become automatically active.

Perhaps the question arises which user account the local BCS will use to connect to the remote server. This account cannot be configured on a per-link base because the local BCS will connect only once to the remote system and therefore the account is identical for all links to a specific host system. You can administer the used accounts in the registry in the key "BroadcastServer\...\RemoteLoginData".

Another word about the rights: As you can see, the slave BCS always uses a single DigaSystem user account to connect to the host BCS. Therefore, the host BCS can see only a single user and will perform all right checks for this user. Typically, you will need a user account with enough rights (e.g. Admin) if you have the write access for remote shows / tracks enabled. In contrast, a user account with only read rights is sufficient if write access is disabled. In addition, there is still the slave BCS which can perform right checks. These are additional right checks based on the local (the slave system) user accounts. For any operation the necessary rights on both systems must have been granted.

11.2.3 Relative starts

Here the term "relative start" refers to groups or elements with the start type "relative" and with a relative ID / GUID. Please note that there is also the start mode which can be relative, but this mode can be used together with the start type "offset floating" too, e.g. for drop-ins. As always, the start type defines the behaviour of the BCS during scheduling, whereas the start mode defines the behaviour of the on-air client during playout. To make the distinction clear here is a small table:

		Start mode: Relative	Start mode: Manual
		The on-air application will automatically start the element at the computed start time.	The element must be started manually.
Start type: Relative	BCS will keep the start time identical to the start time of the **linked** element (or add the specified offset).	Used for links among tracks.	Almost never used because usually regio elements should be started automatically together with the main element.
Start type: Offset floating	BCS will keep the start time identical to the start time of the **previous** main element or add the specified offset.	Used for voice-overs / drop-ins within the same track.	Sometimes used, if the presenter wants to start a drop-in himself.

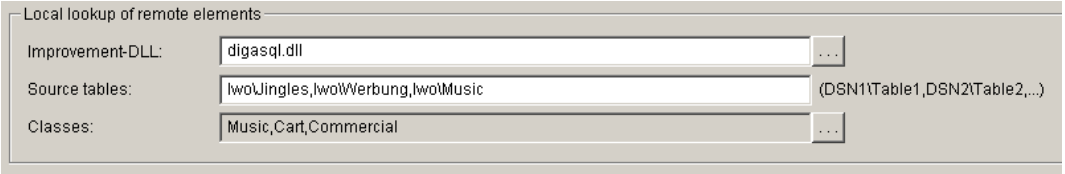
Elements or groups with start type relative are linked to an element or a group in a different track. Therefore, two XML tags are used. The first one is the field <Time_ReferenceNode> which typically looks like this: " ../../00000002/0000006A/00000065". This is a so-called relative ID. Beginning from the current node it specifies the path to the linked node. The special ID ".." means: go one level up. (The example is a link of a relative element in track 1 to a main element in a group in track 0. So the path is: up to track, up to show, down to track with ID 2, down to group with ID 6A, then use element with ID 65.) Relative IDs are mainly used by BCS if it aligns elements among multiple tracks. They were implemented long before relative starts were implemented in TurboPlayer. When this was done, a new link became necessary, unfortunately. The problem is that relative starts must work among multiple TurboPlayer, each of them having loaded a single track only. Therefore, a new identifier for each node has been introduced, which is universally unique: the field <GUID>. This Guid is used to make a second link from the relative to the main element with the field <Time_ReferenceGUID>. (Hint: if you have old versions of the applications, they might not support the GUIDs. Then you will not be able to use relative starts as intended.)

11.3 File handling

11.3.1 BUS

It is only one part of the job to copy the metadata to each location. But generally, it is just as important to copy the media files. This task is handled by BUS with a little support from the BCS. The BUS file check copies the files whereas the BCS makes a translation of file-describing metadata.

For the file check task there is no – or very little – configuration necessary. Aside from the general option "Ignore remote shows / tracks" which completely disables the treatment of remote nodes, there is only one small block of options at the bottom of the configuration dialog:



These options need not be filled for copying remote files. But there is a feature, which needs these parameters: Let's assume that you have a DigaSystem database in both locations and some of the tables are already aligned by another process (e.g. DigAlign). If an element from one of these tables is inserted into a show, BUS would copy the media files to any other remote location linking to the show. But this is not necessary, because the same element already exists in the database of the remote systems. Therefore, an optimization exists: For each specified class the BUS at the remote location uses the improvement DLL to look into the specified tables to find out, whether the element already exists locally. It looks for database entries with the same MusicID, Motive, or MediumName (=CDID). In addition, it checks the size and the first kByte of the file to ensure that the local entry is really identical to the host element. If the search is successful, BUS enters the local entry directly and avoids to copy the element.

Aside from this optimization no configuration of BUS is necessary. BUS can see from the metadata if there are remote links on show or track level and it can see from the element data if a file copy from a remote system must be performed. There are some facts you should bear in mind:

- The BCS and all normal clients do not need file access to the remote system, but BUS does. Therefore, you must (1) open your fileservers for all remote BUS computers and (2) use pathnames – probably with UNC naming – which are visible from any remote location.
- Copying of remote files is always done in advance of copying local files. Typically, the behaviour is like this: Someone enters an element in the host system. The element appears nearly at once at the remote system, but the metadata will say the element is remote and no local filename is available. BUS copies the media files from the host system to the local system and sets the file state to existing. From this time the files are visible for all local clients and BUS continues to make more local copies (if configured).
- Copying of remote files is always a "get" and not a "push" action. It does not matter whether BUS runs in a slave or in a host system. It is always the local BUS, which fetches files from a remote system and inserts them locally. Therefore, it is necessary that a BUS file check runs in every location where files are needed locally.
- BUS starts to copy remote files only if an element on the remote system has at least file state “Accessible”. Elements with lower file states like “Existing”, “Recording” or “Virtual” are ignored. Therefore, it is necessary to run a BUS file check on any system, which should act as a source for media files, too.
- There is an important limitation. Whereas metadata changes of an element are distributed nearly immediately by the BCS to all remote locations, this is not true for the media files. There can be a lot of these files and each file can be large. It takes time to copy these files to the remote systems, especially if BUS is very busy. Therefore, it is possible that an element on a remote system shows already the correct metadata, but the media files are not yet available (or they are old ones), till BUS has treated the element.

11.3.2 Data translation

A very important part of the file handling is the data translation of the file metadata, which is performed by the BCS. This is necessary to achieve two things: the first one is transparency for all other clients except for BUS. That means all metadata describing files must retain its meaning in a distributed system. Therefore, new fields are needed to store any remote information. And secondly BUS should be able to know what action it has to perform without scanning the media files, because this would generate unnecessary network traffic.

Here is an explanation for the new fields, introduced with remoting:

- Remote_Filename<n>, Remote_Presentation<n>: These are the links to the files on the remote system. They are automatically created by BCS when it moves data from host to slave.
- RemoteFileState: The file state of the remote system. This field is automatically created by BCS. The field FileState was extended by a new possible value: "Remote", meaning that an element is only available on the remote system.
- FCTS: This is the "file change timestamp". The field is written by each client, which inserts, updates or replaces a media file. It is NOT written by BUS when only a copy of an already existing file is made. This field is not translated by BCS, therefore it is identical in all locations.
- Loc_FCTS: The "local file change timestamp". A normal client never writes this field. It is written by BUS when it makes the first local copy of a remote file. With the two FCTS fields BUS can see from the metadata only in which location the newer file exists and whether a copy must be performed.

Here are the translation rules when data is transferred between host and slave:

Transfer Host -> Slave:

FileState	->	RemoteFileState
Filename…	->	RemoteFilename…
RemoteFilename…		thrown away
RemoteFileState		thrown away
Loc…		thrown away
FCTS	->	FCTS

Transfer Slave -> Host:

FileState		thrown away
Filename…		thrown away
RemoteFilename…	->	Filename…
RemoteFileState	->	FileState
Loc…		thrown away
FCTS	->	FCTS

Exception: not a single RemoteFilename is present. Then there is a special mapping:

Filename…	->	RemoteFilename…

The exception is needed in case a client on the slave system wants to replace one of the media files. The client writes the path as usual into a filename field. In addition, it deletes all remote-filename specifications. Due to the special rule the written filename will appear as single file reference on the host system (in this case the data in the host will be the one with a remote file link only). Afterwards the BUS in the host system can fetch the specified file from the slave system and the file will be present in both locations.

Perhaps you have noticed in the XML field description that there is another file part, the IIFile part. II means "initial inserter". This file part is used by BCS internally when a new element is created on a slave system. From the moment, the element has been created in both BroadcastServers, this part is obsolete.

Please note that this data translation is only performed if the field <RemoteFileAccess> for the remote link of the show or track is 0 or not present. If the flag is set, it means: all clients have direct access to the files of the remote system. Therefore no translation is performed at all and BUS does not perform any remote file copying. This flag can be set in the dialog with the option "Do not copy media files".

11.4 Regionalization with EndpointStories

11.4.1 Introduction

Since version 6.1 of BroadcastSystem there is another way of doing regionalized playout. It is intended for “type 1”, meaning: the playout of different audio streams for different regions but from one single location.

The general idea is that an EndpointStory works as a container for elements for different regions. There is no background process, which creates the regional information or which distributes the elements to individual regionalized tracks. Everything is in one unit. The user only sees one EndpointStory and whenever he moves it around or deletes it, the elements for all regions are automatically included in the operations without any extra effort.

The regions you want to serve must be defined in the DigaSystem registry in key: Common\DistributionEndPoints. There you can define a name for each region and an image with a position within the list of regions. Hint: this key is also used to define non-linear endpoints, like e.g. Twitter. The big difference is that such a social media endpoint normally should not get a playout duration in the rundown (parameter TimeCalculation=false) and should not be played out by TurboPlayer (parameter Playout=false). For endpoints, which represent a region, the contrary is true. Regionalized elements should get a playout duration (parameter TimeCalculation=true) and a TurboPlayer should play them (parameter Playout=true).

The next chapter describes the affected modules and concepts and gives hints, where you can find more information.

11.4.2 DatabaseManager / ContentManager

A DigaSystem database supports groups with different types. A group acts as a container of database items, which must be located in the same table as the group. One possible group type is the EndpointStory. So far, DBM does not make a big deal out of EndpointStories. They are handled like other group types. So, you can create an EndpointStory, select it and open the member view in the lower half of the window. Then you can add items to the EndpointsStory with drag&drop operations into the member view. There is no special implementation, which supports the user to see, which regions are existing and which are already filled, or to see, whether the items for each region have approximately the same duration. A special tool for this type of regionalized pre-planning is in work and will probably be added to DBM later.

One very important hint: the metadata field of an element DAVID/DistributionEndpoints is used to store the information, for which region the element is intended. If you put an element into an EndpointStory the information about the region is still part of the element. If you put an element into multiple EndpointStories, the database only stores that the element is part of multiple EndpointStories. No copies of the element are made. Therefore, if you change the field EndpointStory for such an element it will be changed in all EndpointStories in which the element is a member! Therefore: don’t do that! Normally, an element is produced for exactly one region and it is not at all reasonable to play this specially produced element in a different region. Therefore, this database restriction should not be a problem. By the way: in BroadcastSystem the situation is different. When you copy an element from one EndpointStory (or Group or Story) to a different one, a real copy is made. Therefore, you could change the DistributionEndpoint of an element without affecting other elements. Nevertheless, though being possible, it is still not reasonable to do this.

ContentManager does not yet support EndpointStories.

The conversion routines, which convert between DBE/DBX and BCS-XML will convert a database group of type “EndpointStory” into a BCS node with tagname “EndpointStory”. All children will be copied. Therefore, pay attention: if you copy BCS EndpointStories, which contain “the same elements” back to the database, the elements will be present multiple times in the database, because there is no such thing as “the same elements” in BroadcastSystem. The children of different EndpointStories are all different elements.

11.4.3 BroadcastServer, SyncStart and neutral elements

BCS will handle EndpointStories in a special way. The very first element for one region will always get the start mode SyncStart, the start type Floating and the start time of the first neutral element. A neutral element has an empty DistributionEndpoint or “TurboPlayer” (which has the same meaning). Neutral elements are intended to be played on the general, non-regionalized stream or they can be used for regions, which do not have any special element for their own.

With the SyncStart mode it is achieved that when the first neutral element starts, all regionalized elements start, too. The SyncStart mode needs one metadata field, which has the same content for the master and for all slave elements. Therefore, you must configure such a field in the DigaSystem registry parameter BroadcastSystem\SyncStartField. You can use any arbitrary existing regular or custom field or you can define a new field name. It is recommended to use the field name “SyncStartID” or “SSID”. BCS will the automatically assign the same SyncStartID for the first neutral element and the first element of each region. The value being used is the value of the field “GUID” of the parent EndpointStory.

Typically, EndpointStories are kept simple and only one neutral element and one element for each region is scheduled. Nevertheless, it is possible to build up mini-rundowns for each region. You can do this by putting multiple neutral elements and/or multiple elements for each region into one EndpointStory. So far, the only supported start attributes are start mode Sequenced and start type Floating for these elements. BCS will automatically assign these attributes. If you do this, we strongly recommend to first put all neutral elements into the EndpointStory, then alle elements for region 1, then for region 2 and so on. Otherwise, you will probably lose the overview.

Time computation is special for EndpointStories. You will see that BCS assigns the same start time for all elements, which are started synchronized. Afterwards, the sequence of elements for one region forms a mini-rundown and the stop- and start times are added up accordingly (taking link values into account, too). The EndpointStory itself gets a duration, which is the maximum duration of all these mini-rundowns.

11.4.4 Import / DigaSQL.dll

The import dlls and the import routines handle EndpointStories like regular stories. This means: they are treated as a single element and are either imported completely or not at all. They can also replace single elements in the destination and vice versa. A difference is made for the improve step, which is being executed by DigaSQL.dll. This step is the database lookup in order to get all needed metadata. Here it could happen that an EndpointStory and all its children have the same ID (e.g. MusicID). In DigaSQL.dll versions before 3.20.3493.0 the dll did stop with the database search when the very first item was found. In the described case with multiple elements with the same ID this is probably undesired. Therefore, since version 3.20.3493.0 the DigaSQL.dll first assembles all items which can be found in all configured lookup tables. If there is exactly one EndpointStory in the found items, it is picked and the dll forgets about individual items, which were found, too. The improvement does not fail here because multiple matches were made. As a result, the improvement step of the import will always pick the EndpointStory from the database, even if individual elements with the same ID are found.

11.4.5 DigAIRange

DigAIRange displays and handles EndpointStories like regular stories. The background color of the EndpointStory line is different. By default, it has a light violet color, but this can be configured in the settings and in the UISchemes with the value: Color_GridEndpointStoryBkgnd.

It is recommended to add the column for DistributionEndpoints so that the user can see (and perhaps modify) the region, for which an element is scheduled. Here it helps greatly to have good and distinguishable icons defined for each region. Users will also need the combo box for the DistributionEndpoints in the filter bar on top of the rundown in order to filter the view – or even more important: to enable to see all available endpoints in case they are missing some in the rundown view. Hint: if the filter box for DistributionEndpoints is hidden, DigAIRange displays all endpoints, because otherwise the users might have an empty rundown or might miss some content and would not be able to change the filter in a way so that they can see everything.

Here is an example screenshot of two EndpointStories (the second one with two elements per region):

S...	Distribution Endpoints	St...	St...	Start	Stop	Duration	Class	P/A/E	Title/General title/Subtitle
	TP			11:00:00	11:19:42	00:19:42			Music 1
	TP			11:19:42	11:19:57	00:00:15			Regio leg 1
	TP			11:19:42	11:19:57	00:00:15	$	ADMIN	ARIEL REGULAR/PROCTER
				11:19:42	11:19:57	00:00:15	$	ADMIN	BUENA VISTA HOME VIDEO
				11:19:42	11:19:57	00:00:15	$	ADMIN	VPower/Shell & DEA Oil Gm 1
				11:19:42	11:19:57	00:00:15	$	ADMIN	BUKO- MD FOODS DEUTSCHL.
	TP			11:20:00	11:20:40	00:00:40			Regio leg 2
	TP			11:20:00	11:20:20	00:00:20	$	REKLAME	Kiwi, vinner av mai matbørsen
	TP			11:20:20	11:20:40	00:00:20	$	REKLAME	Spar, first price - juice og fiskepinner
				11:20:00	11:20:20	00:00:20	$	ADMIN	Celebrations / MASTERFOOD
				11:20:20	11:20:40	00:00:20	$	ADMIN	Fürst von Metternich/HENK
				11:20:00	11:20:20	00:00:20	$	ADMIN	Liga Points/WARSTEINER BR
				11:20:20	11:20:40	00:00:20	$	ADMIN	Kinder Friends/FERRERO
				11:20:00	11:20:20	00:00:20	$	ADMIN	TV-Digital/SPRINGER VERLA
				11:20:20	11:20:40	00:00:20	$	ADMIN	PIRELLI REIFEN
	TP			11:20:40	11:48:45	00:28:05			Music 2

11.4.6 TurboPlayer

The main work of handling regionalized playout is being done by TurboPlayer. Please have a look into chapter 5.3 of TurboPlayerTechManual for more information.

12 LOGS

12.1 Basics / DigaSQL logging

In some of the modules, log output is implemented with a direct file writing. This affects some of the TurboPlayer IO modules and the two services, which are written in C#/.NET: BCSService and TurboPlayerService. For these modules the configuration of the log output is done within the settings of these modules. This is explained in extra documents, which are not part of this manual.

Here we describe the log output via DigaSQL.dll, which is used by nearly all C++-based modules, especially by the four big ones: BCS, BUS, DigAIRange and TurboPlayer. Regarding this type of logging, the first point of reference is in the parameter documentation (DigParam.rtf) for key “Protocol”. This key is the central point for the configuration.

The basic idea of logging via DigaSQL is that a developer of a module simply calls a function (“DSLog”) and does not care about the details, especially when and where the log output is written to. The administrator on the other hand, can configure, in which log output he is interested and where to the output should be written. The following concepts apply:

- This system uses something called “destination”. This is a keyword, which is being specified by the developer when calling the log function. This can be used to realize something like different log levels, but more often it is used to differentiate between areas of interest or sort of information. The administrator need to know, which keywords are being used by the modules, because these keywords must be used as parameter name in key “Protocol”. The destinations, which are used by BroadcastSystem are described below, but you can find others in the parameter documentation (DigParam.rtf). If a specific destination is not defined in the key “Protocol”, the corresponding log is not being created. If you want to temporarily disable a certain destination, change the parameter name so that it starts with an underscore ‘_’ as first character. These parameters will be ignored by the dll.
- As administrator you do not specify the full filename, which is then used. Instead, you specify only the path and the first part of the filename. The DigaSQL.dll automatically appends “MMDD.log” to the filename, whereas “MM” represents the current month and “DD” represents the current day. If the destination folder does not exist, it will automatically be created. This means: one log file per destination and per day will be written. If files become too big, you can split them up – see below. In contrast, combining multiple days into a single file is not possible.
- When specifying filenames (first part) you can make use of some placeholders:
  - %s the station name or computer name
  - %c the station name or computer name
  - %u the current DigaSystem username. If no user is logged in, the artificial name "DUMMY_USER" will be used instead.
- When specifying filenames (first part) it is strongly recommended to make use of the registry variables. This is independent of the log system and works for every value in the registry. Here it is useful to use a registry variable like “%LogDir%”. The variable needs to be defined in the top-level key “Variables”. Whenever you specify the directory for a log file, you should then use this variable. This allows changing the log directory by modifying only the content of the variable, but you need not touch a lot of destinations values.
- Instead of specifying the beginning of a filename, it is also possible to specify a section-strings with additional configuration values. The following sections are supported:
  - [FILE] is the partial filename as if you did not use a section string
  - [SIZE] is the maximum size of the logfile. If new log data would exceed the file size, the DLL will rename the file (append an underscore and a continuous number) and restart with an empty one. You can append a k(Byte), M(Byte), or G(Byte) to the size, e.g. "[SIZE]10 MByte".
  - [CYCLE] is a flag which can be 1/Yes/TRUE or 0/No/FALSE. If set, the DLL will restart to write at the beginning of the file if the maximum size has been reached. (A size must always be specified.) That means there will be only a single logfile with a maximum size which is always "re-used".
- It is possible to specify the same value (path and first part of filename) for more than one destination. This is even possible for multiple computers. As a result, the output for more than one destination and/or from more than one computer will be combined into a single file. This works, because the DigaSQL.dll caches the log data and writes it with a background thread into files. More specific, the dll opens a log file exclusively, writes the cached log output to the file and then closes the file again. This allows a different instance of the dll (on the same or on a different computer) to write to the file. A disadvantage is that you

will find data for one destination/from one computer in blocks of random size in the resulting file, meaning: the content is no longer sorted chronologically. If you use a log viewer like LogView.exe, which sorts the lines by time, this is mitigated. The term “mitigated” is used, because a log can contain multiple lines with the same time in milliseconds. Sorting within these lines is not possible and you have to be careful when interpreting such a file with interleaved content.

- There is a special parameter “BreakLines”, which does not define a log destination. Instead, it is possible to define, whether the DigaSQL.dll should insert line breaks for long lines. The parameter is a bool parameter, so you can specify the usual values: “TRUE”, “FALSE”, “Yes”, “No”, 1, 0. You can also specify the value “Default”, which is the default value, too. In this case the DigaSQL.dll will use the parameter regarding line breaks, which was given by the developer calling the DSLog function. If line breaks are active, the DigaSQL.dll will try to break lines after 72-88 characters and will try to break at spaces.
- There is a special parameter “KeepDays”, which does not define a log destination. Instead, this parameter specifies the number of days log files are kept. After the configured number of days, the DigaSQL.dll automatically deletes log files. Default value is 30. If you want to keep log files for a longer time you should copy or move them to a different location. Hint: copying, moving or deleting log files is even possible for the files, which are currently in use. This is possible because the dll closes the file after each write operation.
- The files written by the DigaSQL.dll are (still) formatted as ANSI text. Neither Unicode, nor UTF-8 is used. ANSI was the format initially used, many years ago, and this was never changed. This gives smaller files than Unicode but special characters like mutated vowels might not be represented correctly. As the language being used for log text is nearly always English and only some details like element titles might contain these characters, this is normally no problem.
- The DigaSQL.dll distinguishes regular lines and error lines. An error line is still regular text, but as identifying feature the dll appends the special character A0 (hex) / 160 (decimal) to an error line, which is a “non-breakable space”. In normal text editors like Notepad, you will not see any difference to other lines. Therefore, you have to use an editor or viewer, which can make these error lines visible. Our LogView.exe does it automatically and displays error lines in red. Other log viewers (like LogExpert) might be configurable so that the non-breakable space is recognized, and the error text is displayed differently.
- We want to mention one nasty detail. If you want to compare (or even combine) multiple log outputs, you might notice discrepancies in the time being used by different modules. Each module can print the time it wants at the beginning of each log line. This might sound strange, because there is only one real time, but there can be different reasons. The biggest discrepancy occurs with the logs of MultiPlayer (though DigaSQL is not used in this case). MultiPlayer prints UTC times. So, you always need to know the current time zone and whether time light saving time was active when the log was written to convert these times into local times and to compare these times with other logs. And TurboPlayer uses a special “multi-media timer” internally, and these times appear in the output of all logs, too. It is configurable, how much maximum offset from the real time is allowed for this special timer. By default, it is 500ms. Whenever you compare times of the log files from these areas, you need to be careful and check, whether applying an offset between two logs is necessary. Sometimes, you can see such an offset directly by comparing the times of a solitary event, like the start of an element, when the logs are not mixed with other stuff.

12.2 BCS Logs

BroadcastServer does not support different log destinations. There is only one single destination, which is:

BCS-Log

BCS uses different log levels, which can be set with the two parameters:

BroadcastServer\<config>\General\ProtocolLevel

BroadcastServer\<config>\General\ScreenProtocolLevel

The first parameter defines the log level for writing to the log file, whereas the second parameter defines the log level for the output to the log view in the lower half of the main screen of BCS.

6 log levels are supported:

0) This deactivates log output.
1) The “normal” log output as it is recommended in regular operations when no extra logging is needed.
2) This extends level 1, but only with very few details. Therefore, it is not recommended to use this level.
3) Debug level. This level enables a lot of log output about client requests but also about background operations, like e.g. file reading and saving. This level (or a higher one) should always be activated if you encounter problems or errors. It is also recommended to use this level when starting with a fresh installation. You can then monitor the size and the content of the log files for a while. If the size of the files becomes too big with this level and if there are no problems or errors visible in the log then you can decrease log level.

4) Extended debug level. This activates additional information, especially: individual file operations, distribution of play state / time messages, regular memory status information and a log line for each received or sent WebSocket message (but without content).
5) Maximum level. This includes all possible log output. The additions to level 4 are especially: the data of all WebSocket messages (in principle this includes all sent and received data, but in order to not make the log explode, the log is cut off after some kBytes), logging about locks being used by BCS for preventing simultaneous access to the manager data. This log level should only be activated for special debugging purposes.

In case you start an instance of BroadcastServer with a different configuration name than “Default”, then the log destination is different – it contains the name of the configuration:

BCS-<config>-Log

There is one additional registry parameter, which can be used to switch on a specific type of debug logging:

BroadcastServer\<config>\General\FieldsToLogOnInsert

This is a comma-separated list of BCS field names (technical names, i.e. the names as they show up in the XML tags), which are logged in the BCS protocol when a client inserts a new node. This is done when individual nodes are inserted, or a block insert is done. It is not logged for a whole tree insert.

There is one more registry parameter, which is independent of the regular logging:

BroadcastServer\<config>\General\WriteMemoryDumpBeforeRestart

If this parameter is set to TRUE, a full memory dump of the BCS process is written to the current working directory before BCS needs to do a restart because a serious internal error was detected. (In case BCS runs as a Windows service, the dump is written to the Windows system directory.) To do this, a version 2.6.0.0 of Restart2.exe or newer and ProcDump.exe (from the SysInternals suite of Microsoft) is required. For BCS64.exe use ProcDump64.exe to create the dump files. These programs must reside in the working directory, in which BCS is located.

12.3 BUS Logs

BroadcastUtilityServer does not support different log destinations. There is only one single destination, which is:

BUS-Log

BUS in general and all tasks write to this single log. The output of different tasks can be distinguished, because each line has the ID and the name of the task at its start. Nevertheless, if you want to have the output of only a single task in a log, then you must install an extra BUS instance for this task.

When the name of the BUS instance / configuration is not the default one (“BUS”), then the destination name of the log is configurable in the general settings dialog of BUS – in Protocol / Destination. You can configure the destination name as you like, but it is recommended to use something like:

BUS-<config>-Log

BUS does not support multiple log levels. Nevertheless, you can activate three additional debug areas:

- Each task has an individual “DebugLog” setting. You can activate this output with the context menu of the task view of BUS for the rightest most column.
- In the general settings dialog you can activate “Include debug messages”. This parameter does not only determine the default when creating new tasks, it also activates extended log output in general, which is mostly in the area of BCS connections and especially for activating the output of the data of WebSocket messages (the data is cut off after some kBytes). For the latter it is necessary that the next parameter is active, too.
- In the general settings dialog you can activate “Log WebSocket messages from/to BCS”. A line for each sent or received message will be printed. This does not include the data of these messages. If you want to include the data you need to activate the previous parameter, too.

Please have in mind that some BUS tasks make use of dlls (like import or export task). These dlls typically have their own logging and it is configurable in the settings dialogs. In most of the cases, this sort of log output does not use DigaSQL logging but instead is done by directly writing to log files.

12.4 DigAIRange Logs

DigAIRange supports the following log destinations:

DigAIRange-Init	This protocol logs the initialization process of DigAIRange during startup. This can be useful if the program does not start.
DigAIRange-Error	All errors from DigAIRange. This log contains all the error messages, which are displayed to the user of DigAIRange in message boxes.
DigAIRange-DDE	All DDE commands which are sent or received by DigAIRange. The DDE interface is used for the communication with other desktop programs, like DBM and the editors. Despite of the name, a communication via COM automation, WM_COPYDATA or named pipes is logged, too.
DigAIRange-COM	This protocol logs the data about the communication with BCS servers of DigAIRange. It is useful for debugging communication problems. For each sent or received WebSocket message a line is printed. This does not include the data of these messages. If you are interested in the data, you have to activate the log destination "DigAIRange-Debug", too.
DigAIRange-Debug	This destination represents some information, which might be helpful for examining problems. Logging for this destination has only been implemented in very few cases, in which problems did arise in the past, but one thing is worth to be mentioned: it activates logging around saving data from the mask. It is especially helpful if you encounter the problem that DigAIRange displays the question “Content has changed – do you want to save?” though the user did not change the data. Then typically an auto-correction of invalid data did happen. In the log you can find out which data was modified.
DigaSQL-Security	This log is used by DigAIRange for exactly one purpose: whenever a login to a BCS succeeded or failed one line is written.

A remark regarding log output in case of multiple instances of DigAIRange: it is possible to start DigAIRange multiple times when using the command line switch -mi. This gives different names of the running instances: “DigAIRange2”, “DigAIRange3”, etc. Nevertheless, this does not change the log destinations. They always stay the same. Therefore, you will get the output of different instances in one file.

12.5 TurboPlayer Logs

First, a general remark: there is only one log destination, which uses “TurboPlayer” in its name. This is only the log of the GUI. The engine has multiple log destinations, and they all start with “BCE”. This is due to the historical name “Broadcast Control Engine”, which was used in the first years, before the project was named “TurboPlayer”. The old name still sticks to the destination names.

TurboPlayer supports the following log destinations:

TurboPlayer	This is the log destination for the desktop (C++) GUI. It does not matter, whether the GUI is running within the same process as the engine or as a standalone process – this log is always separate from the engine logs. The generated log contains user actions and a lot of information about the internal operations of the GUI.
BCE-TechLog	These three destinations represent the main logs of the TurboPlayer engine – about technical information, extended technical information and errors. For historical reasons, these destinations were separated, but during time it turned out that it’s best, to combine these three destinations into a single log file. (This is strongly recommended nowadays.)
BCE-TechLogEx	
BCE-Error	
BCS-PlayLog	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-TreeOp	This is an extra protocol, containing only information about BCS tree operations (inserts, updates, deletes of elements or higher nodes). This is a very important log for analyzing errors, especially in regards to BCS communication, and it is strongly recommended to activate this log! If you activate the output of all sent/received WebSocket messages (see below) then these messages are printed into this log, too.
BCE-IO-<module name>	There is an extra log destination for each loaded IO module. 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. Here "module name" means one of the registry keys below "TurboPlayer\IO". In most cases this type of log need not be activated, because the IO modules can write their own logs by writing directly to a file (which can be configured in the settings dialogs of these modules). Normally, it should be preferred to use the direct logging of a module, if it provides this feature.
BCE-Messages	When you have read TurboPlayerTechManual you will know that TurboPlayer uses thread-messages for the communication of the main internal modules. These messages can be transferred in-process, via memory mapped files on the same computer and via TCP/IP between different computers. In any of these cases it is possible to write each sent or received thread-message into this protocol. This is a very powerful feature for debugging, but without limitation of the written messages, this log would explode after short time. Therefore, you must also select the messages IDs of interest in the parameter "TurboPlayer\Logging\MessageIDs" (see below). If you run TurboPlayer GUI separated on a different computer, and if you are interested in thread-messages, which are sent between engine and GUI, then it is necessary to activate this log and the message IDs on both computers. Only then you will see both sides of the communication and whether messages get lost.
BCE-Dump	This is a very special type of log. It is not written regularly / automatically, you have to request the creation explicitly by executing the corresponding macro function or by triggering the generation in the GUI. This log then contains a snapshot of the internal engine state. This includes a lot of information, general, line- and channel-states, variables and the loaded elements (metadata and additional internal states). This can be super-helpful for debugging. As this log does not contain time-sorted lines, it is best viewed by a simple text editor, not by a log viewer.

There are 3 additional registry parameters, which can be relevant for logging:

- **`TurboPlayer\Logging\MessageIDs`**
  This parameter is a comma-separated list of all message IDs you want to log into the protocol of type "BCE-Messages" (see above). 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.
- **`TurboPlayer\Logging\MessageClasses`**
  This is an old parameter, which has been replaced by the MessageIDs parameter. It allowed to activate log output for certain areas of interest, which are: Init, TreeOp, Load, Play, Time, Regular, IO, Key, Event, All. This parameter is still active, but it is recommended to use the more specific MessageIDs parameter.
- **`TurboPlayer\Logging\DebugLog`**
  This parameter is a comma-separated list of possible log extensions for debugging purposes. The following keywords are defined for this parameter:

BCSCommunication	enables the output of all sent and received WebSocket messages to/from BCS into the log destination "BCS-TreeOp".
Filler	enables extended information from the filler routine into the log destination "BCE-TechLogEx".
FileAccess	enables extended information about every file access - but so far this keyword is only evaluated by the TurboPlayer GUI.
BroadcastMessages	enables extended information about BCS broadcast messages into the log destination "BCE-TechLogEx".
PreciseTime	enables additional information into the log destination "BCE-TechLogEx" when a start or a stop command for an element is being sent to MultiPlayer: in addition to the normal output of the time used by TurboPlayer (Multimedia timer) it prints the current timestamp of the Windows function *GetSystemTimePreciseAsFileTime.* This can be helpful when examining the precision when executing start or stop commands.
Timers	adds output to the BCS-TechLogEx destination around setting or removing start- and stop timers for elements (available since version 6.5).

12.6 BCSSearch and ElasticIndex

The two modules BCSSearch.ocx and ElasticIndex.dll are used for the full-text search. BCSSearch.ocx is only used within DigAIRange and is the GUI, which is visible in the lower window of DigAIRange for executing search requests. It uses ElasticIndex.dll to communicate with an ElasticSearch server, which hosts the full-text index. This index is built up in BUS Index task with the help of ElasticIndex.dll.

When using the modules for searching, it is recommended to activate both logs and to keep them active. Full-text searches are not done very often, and the amount of logged data is not too big. When using ElasticIndex.dll for the BUS Index task it is only recommended to active the log at the beginning or when you encounter problems. This is very helpful, because the ElasticIndex log contains the data, which is being sent with REST commands to ElasticSearch server and which data is being answered. When the system runs without problems you can switch this log off, because otherwise building up an index can create an enormous amount of log output.

Both modules use DigaSQL.dll logging and the corresponding log destinations are:

`BCSSearch`

`ElasticIndex`

12.7 MultiRec and OnAirTrackMixer

These two modules are ActiveX controls, which are used by DigAIRange and by TurboPlayer and they do not read settings themselves. Instead, the host application must provide all parameters for them. This applies to the log settings, too. Therefore, the location of the log settings depends on the application.

12.7.1 In DigAIRange

For MultiRec you can configure a log output with this registry parameter:

DigAIRange\MultiRec\Logging

DigaSQL logging is not used, and you have to always specify a section string with individual parameters, which are:

- [DIRECTORY] The output directory (without terminating backslash).
- [FILENAME] The filename without extension, e.g. “MultiRec”. Like DigaSQL.dll, MultiRec appends the current day and month, a possible counter if the file is split, and the extension .LOG.
- [TYPE] Works like the DigaSQL destinations. You can activate individual areas of interest, as there are: GLOBAL|EVENTS|METHODS|FILE|ACCESS|DEBUG
  Use the pipe character like in the example to separate the areas.
- [DELETEOLD] TRUE or FALSE. Activates automatic deletion of old log files.
- [DELETEDAYS] A number. The number of days how long old log files should be kept.

There is a second instance of MultiRec, which is used within OnAIR TrackMixer (or the old Crossfade Mixer). For it no other parameter is necessary. The mentioned parameter “Logging” is used for this instance of MultiRec, too.

The log output for OnAIR TrackMixer (or the old Crossfade Mixer) can be configured with this registry parameter:

DigAIRange\Crossfade\Logging

Here you specify a filename with a directory and extension. OTM will take this filename and will append “-YYYYMMDD” to it, while keeping the extension. Hint: DigaSQL logging is not used.

12.7.2 In TurboPlayer

For MultiRec you can configure a log output with this registry parameter:

TurboPlayer\MultiRec\Logging

The format is identical to DigAIRange. If you run a different configuration than “TurboPlayer”, you need to adapt the first level of the key name accordingly.

For the instance of MultiRec, which is used within OnAIR TrackMixer (or the old Crossfade Mixer) you can configure the log output with this registry parameter:

TurboPlayer\MultiRec\Logging_CFM_MultiRec

The log output for OnAIR TrackMixer (or the old Crossfade Mixer) can be configured in one of these two registry parameters:

Protocol\OnAIRTrackMixer

Protocol\CrossfadeMixer

Interestingly, though being located in top-level key “Protocol”, these two entries are not part of DigaSQL logging. Nevertheless, you can use the same format as usual: the directory and the beginning of a filename, like “OTM”. “-YYYYMMDD.LOG” will automatically be appended. A section string format cannot be used in this case.

12.8 DigaSQL Logs

The DigaSQL.dll itself can write to some log destinations. The ones, which are of interest in regard to BroadcastSystem are:

DigaSQL-Error	Reports about errors within the DigaSQL.dll. This might be relevant in case of some errors occurring within BroadcastSystem modules.
DigaSQL-DbOp DigaSQL-SQLCommand	These two logs contain information about executed database operations. As there are only a few places in BroadcastSystem, which execute database operations (e.g. improvement during import, AutoReplace task in BUS or saving a recording in DigAIRange, MiniDBM operations in TurboPlayer), this log is of limited use.
DigaSQL-Improve	This is a special log, which is only relevant in one specific step in BroadcastSystem: when an import is done (in DigAIRange or in BUS) and a database lookup/improvement is active, the DigaSQL.dll can log all activities around the improvement of an element. This includes information about the searched tables, which lookups are made, which elements are found and whether found elements are picked for the improvement. You can evaluate this log in case of problems around the improvement step during import, especially if the lookup does not find referenced database elements and an improvement does not occur.
DigaSQL-Registry	This log contains all read and write operations to the DigaSystem registry. Be aware that this log can become very big, because programs, which start up, typically read thousands of parameters and all are logged. Nevertheless, this log can help if you want to find out which parameter with which value a program reads or writes. After such an evaluation this log should always be deactivated again.
DigAS-StackDump	This is a very important log in case of application crashes. All C++ core modules of BroadcastSystem can write to this log in case of unhandled exceptions, caught crashes or full application crashes. Due to the nature of the situation when this log is being written (typically, the application is just “dying” and internal structures might be corrupt and unreadable) there is no guarantee that useful information is written. But from experience we can say that around 50% of these logs are useful. Each report about an incident contains general information, the content of registers, exception information, a list of all loaded modules and as the name says a stack dump. It is generated when a Windows exception occurs and in some other cases where a stack dump provides useful information for debugging. The quality of information can be improved if you have a “program database file” (*.pdb) for the main application and/or dlls (these files must reside in the same directory as the executable and/or the dlls) and the Windows dll "DBGHELP.DLL" in your Windows (system) directory. Nowadays, this dll is normally part of the regular Windows files and need not be installed manually. On 64-bit systems the dll exists twice (with exactly the same name), once in the System32 directory and once in the SysWOW64 directory. When the crash happens in C#/.NET code this type of stack-dump does not help, unfortunately. In this case it is always necessary to have MiniDump files (see below).

Due to its importance in case of crashes, here is an example of a DigAS stack-dump:

```
*****************************************************************
DigaSystem  Stack dump
Windows:    Windows Server 2008
Station:    FH-KLTP010
Program:    DAVID Systems TurboPlayer 5.9_Pro_0x00030090
User:       KLTP010
Time:       2020-11-23 21:04:53.143
Process ID: 00001CBC (7356)
Thread ID:  00000554 (1364)

Windows exception E06D7363: Microsoft C++ exception
Address: 74E4C5AF 01:0000B5AF C:\Windows\syswow64\KERNELBASE.dll

Registers:
EAX:0018F8F8 ECX:00000003
EBX:0018F99C EDX:00000000
ESI:00B2D080 EDI:00C069CC
CS:EIP:0023:74E4C5AF
SS:ESP:002B:0018F8F8  EBP:0018F948
DS:002B ES:002B  FS:0053  GS:002B
Flags:00000212

Loaded modules:
C:\Program Files (x86)\DigaSystem\TurboPlayer.exe   Version: 5.9.2354.0 LinkTime: 2020-04-
29 17:35:00 BaseAddress: 00400000
C:\Windows\SysWOW64\ntdll.dll   Version: 6.1.7600.16385 (win7_rtm.090713-1255) LinkTime:
2019-04-19 03:51:58 BaseAddress: 771C0000
.....

Intel call stack:
Address  Frame    Logical addr  Module
74E4C5AF 0018F948 0001:0000B5AF C:\Windows\syswow64\KERNELBASE.dll
00958BE2 0018F98C 0001:00557BE2 C:\Program Files (x86)\DigaSystem\TurboPlayer.exe
00955536 0018F9A8 0001:00554536 C:\Program Files (x86)\DigaSystem\TurboPlayer.exe
....
771F97D5 0018FFEC 0001:000297D5 C:\Windows\SysWOW64\ntdll.dll

Symbol call stack:
Address  Frame    Logical addr   Symbol [File/Module]
74E4C5AF 0018F948 0001:0000B5AF  RaiseException+58 [C:\Windows\syswow64\KERNELBASE.dll]
                                 Params: E06D7363 00000001 00000003 0018F980
00958BE2 0018F98C 0001:00557BE2  <symbol not found> [C:\Program Files (x86)\DigaSystem\
                                 TurboPlayer.exe] Params: 0018F99C 00C069CC 00B2ADB0
.....
771F97D5 0018FFEC 0001:000297D5  RtlInitializeExceptionChain+36 [C:\Windows\SysWOW64\
                                 ntdll.dll] Params: 00953628 7EFDE000 00000000 78746341
End of DigaSystem stack dump
*****************************************************************
```

Even if you are no developer, you can read out useful information and make a basic check, whether the stack dump is valid.

1. The first block lists information about the operating system, the computer name, the crashed process and the time of the crash.
2. The second block tells us what exception occurred. In this case it was a MS C++ exception. There might be some additional information, which is normally only interesting for the developer of the module.
3. These are the CPU register values when the crashed occurred. Only interesting for the developer and only for some very special analyzing techniques.
4. The block "Loaded modules" lists all loaded modules (the main application and all loaded libraries). This is interesting for non-developers, too, because here you can find the version numbers and the build date/time of all modules.
5. The Intel call stack can quickly be checked by a non-developer. If it is super short (e.g. containing only a single or 2-3 lines), it is defect and is not usable. The same is the case if the addresses are obviously

nonsense (e.g. only the address 00000000). One additional thing you can read out as non-developer are the affected modules. Typically, the main application always occurs somewhere in the stack. But sometimes you have also other modules, e.g. MultiRec or DigaSQL. This is a strong hint that these modules might be responsible for / involved in the crash.

6. The Symbol call stack is normally only a copy of the Intel call stack with some different detail information. For a non-developer it does not help more than the Intel call stack. It is only interesting, if only one of the two stacks seems to be defect, while the other seems to be o.k. In this case it is worth trying an analysis of the stack dump.

One hint about files with the extension **SXL/UXL**: These files are similar to a DigaSQL stack dump and might be generated by one of the submodules (dlls, ocx). They do not contain the list of loaded modules and are therefore inferior. Writing these files was implemented earlier than the implementation of the DigaSQL stack dump writing and it competes with the DigaSQL writing. Therefore, you might get these files with some (old) modules if the problem occurs within them. In case of a crash, it is useful to look into the mentioned directories, whether an SXL or UXL file is present.

12.9 Log examples

12.9.1 BCS client update - log level 1

Here is an example of a BCS log with default log level 1. It shows a client (DigAIRange), which first performs a login, then modifies the title of a single element “Girls just want to have fun” and then logs out and disconnects again:

```
2025-02-12 10:14:40.430 [21648] Accepted new IP connection 3 of peer 10.1.1.18:64367.
2025-02-12 10:14:40.430 [21648] Connection 3 to web socket client 10.1.1.18:64367 established.
2025-02-12 10:14:40.433 [21648] 10.1.1.18:64367 : (R-LIO) Received login request
2025-02-12 10:14:40.434 [21648] 10.1.1.18:64367 : User "MARKUS" at WS "MARC24" : LOGIN; Method =
                                HMAC, Connection-Nr = 3, Application protocol version = 0x010B
2025-02-12 10:14:40.434 [21648] 10.1.1.18:64367 : Unique ID for connection number 3 =
                                "MARC24-1\DigAIRange"
2025-02-12 10:14:51.671 [21648] 10.1.1.18:64367 : User "MARKUS" at WS "MARC24" : UPDATE DATA
                                (ID="00000000/00000067/000007E9/00000002/0000000C/0000006C/
                                00000002/00000064/00000068"); Node Info: Tag = "Element"; Title =
                                "Girls just want to have more Fun" ...
                                in Show "Tagessendung" on 2025-02-12
2025-02-12 10:14:55.559 [21648] 10.1.1.18:64367 : (R-LIO) Received logout request
2025-02-12 10:14:55.559 [21648] 10.1.1.18:64367 : User "MARKUS" at WS "MARC24" : LOGOUT
2025-02-12 10:14:55.559 [21648] Connection 3 to WebSocket client 10.1.1.18:64367 was closed.
                                Reason/error: Der E/A-Vorgang wurde wegen eines Threadendes oder
                                einer Anwendungsanforderung abgebrochen
2025-02-12 10:14:55.559 [21648] WebSocket connection 3 from address 10.1.1.18:64367 disconnected.
```

The number [21648] represents the thread ID of the thread, which handles the request of the client. This can help if other client requests are handled interleaved. But a thread is not assigned to a certain client. There is a thread pool for handling client requests, and it is random, which thread is picked for each request.

You can see that the IP address and the local port number of the client is printed in all the lines, which are related to a client’s request. This can help to identify or filter for connected lines.

If you step through the log, you can see that the client established a connection, performed a successful login as user “MARKUS”, requested to update an element, which is specified with its full ID, performed a logout and then disconnected. What you do not see is additional information, like the sent or received WebSocket messages.

12.9.2 BCS client update - log level 5

If you activate log level 5, you will get much more information. We do not print it completely here, because the log spans multiple pages only for the same few steps as above. We can only encourage you to try it out yourself and then examine the BCS log. Here only some highlights:

```
2025-02-12 14:22:34.484 [21648] Received text message from connection 4 / client 10.1.1.18:51559.
                        Message=<?xml version="1.0" encoding="UTF-8"?>
                        <LoginLogoutRequest>
                        <TransactionNo>79</TransactionNo>
                        <Login>true</Login>
                        <SelfLogin>false</SelfLogin>
                        <BuddyLogin>false</BuddyLogin>
                        <ADLogin>false</ADLogin>
                        <User>MARKUS</User>
                        <AuthenticationType>HMAC</AuthenticationType>
                        <Authentication>#HMAC-SHA256:uIZurqOISKDYrkPOx+
                             QLC9z3ktl3QObMyil4OLNjBhk=</Authentication>
                        <WorkstationID>MARC24</WorkstationID>
                        <ConnectionGUID>MARC24-1\DigAIRange</ConnectionGUID>
                        <ProtocolVersion>267</ProtocolVersion>
                        <BCSAppType>Showplanner</BCSAppType>
                        <ProtocolType>SPC</ProtocolType>
                        <RemoteShowWriteSupport>true</RemoteShowWriteSupport>
                        <ExtendedLockInfo>true</ExtendedLockInfo>
                        <GetBuddyInfo>false</GetBuddyInfo>
                        <RemoteProcessID>0</RemoteProcessID>
                        <Language>English</Language>
                        </LoginLogoutRequest>
2025-02-12 14:22:34.484 [21648] 10.1.1.18:51559 : (R-LIO) Received login request
2025-02-12 14:22:34.485 [21648] 10.1.1.18:51559 : User "MARKUS" at WS "MARC24" : LOGIN; Method =
                        HMAC,  Connection-Nr = 4, Application protocol version = 0x010B
2025-02-12 14:22:34.485 [21648] 10.1.1.18:51559 : Unique ID for connection number 4 =
                        "MARC24-1\DigAIRange"
2025-02-12 14:22:34.485 [21648] 10.1.1.18:51559 : Reading rights for new connection 4 ...
2025-02-12 14:22:34.526 [21648] 10.1.1.18:51559 : ... reading rights for new connection 4 done!
2025-02-12 14:22:34.526 [21648] 10.1.1.18:51559 : (R-LIO) ... sending answer 1
2025-02-12 14:22:34.526 [21648] Sending compressed text message 5 to connection 4 / client
                        10.1.1.18:51559. Message=<?xml version="1.0" encoding="UTF-8"?>
                        <LoginAnswer>
                        <OpCode>1</OpCode>
                        <TransactionNo>79</TransactionNo>
                        <Result>1</Result>
                        <ServerIsMaster>true</ServerIsMaster>
                        <IsPreferredMaster>false</IsPreferredMaster>
                        <ServerProtocolVersion>267</ServerProtocolVersion>
                        <ServerID>Nchen:19760</ServerID>
                        <TreeWriteCount>0</TreeWriteCount>
                        <StartAttributeInheritance>Regular</StartAttributeInheritance>
                        </LoginAnswer>
2025-02-12 14:22:34.526 [22328] Message 5 was successfully written to connection 4 / client
                        10.1.1.18.51559.
```

This shows a login request of DigAIRange. You can see details, like the fact that no password is sent, but only a SHA256 hash value; or the answer of BCS with some of its current properties. If you are interested in more details of such a request and the corresponding answer, please check BCSProgrammersManual. It explains the messages and all possible parameters in detail.

Later in the log you might see other operations of DigAIRange, like requesting the root node, then a day node, then possibly a show/track and then DigAIRange might request notifications for loaded shows/tracks. This is how the update of a single element like in the example above does now look like:

```
2025-02-12 14:22:49.404 [22328] Received text message from connection 4 / client
                        10.1.1.18:51559. Message=<?xml version="1.0" encoding="UTF-8"?>
                        <GetNodeRequest>
                        <TransactionNo>96</TransactionNo>
                        <NodeID>00000000/00000067/000007E9/00000002/0000000C/0000006C/
                                00000002/00000064/00000068</NodeID>
                        <Type>Node</Type>
                        <SubLevels>0</SubLevels>
                        </GetNodeRequest>
2025-02-12 14:22:49.404 [22328] 10.1.1.18:51559 : (R-GET) Receiving "Get Node" for ID =
                        "00000000/00000067/000007E9/00000002/0000000C/0000006C/00000002/
                         00000064/00000068" (0 levels, change time only=no, auto notify ="no") ...
```

```
2025-02-12 14:22:49.404 [22328] MRSWMultiItemLock[Tree]: Granted read lock for item
                         "00000000/00000067/000007E9/00000002/0000000C/0000006C/00000002/
                          00000064/00000068" to thread 22328. LockID=1512.
2025-02-12 14:22:49.404 [22328] 10.1.1.18:51559 :    ... node info: Tag = "Element";
                        Title = "Girls just want to have more Fun"
2025-02-12 14:22:49.404 [22328] 10.1.1.18:51559 : (R-GET) ... sending answer 1
2025-02-12 14:22:49.404 [22328] Sending compressed text message 22 to connection 4 /
                        client 10.1.1.18:51559. Message=<?xml version="1. 0" encoding="UTF-8"?>
                        <GetNodeAnswer>
                        <OpCode>1</OpCode>
                        <TransactionNo>96</TransactionNo>
                        <Result>1</Result>
                        </GetNodeAnswer>
                        <<<<DATA>>>>
                        <Element><ID>00000000/00000067/000007E9/00000002/0000000C/0000006C/
                            00000002/00000064/00000068</ID>
                        ……
                        </Element>
2025-02-12 14:22:49.404 [22328] MRSWMultiItemLock[Tree]: Released read lock with ID 1512
                        of thread 22328.
2025-02-12 14:22:49.404 [21648] Message 22 was successfully written to connection 4 /
                        client 10.1.1.18.51559.
2025-02-12 14:22:49.405 [21648] Received text message from connection 4 / client
                        10.1.1.18:51559. Message=<?xml version="1.0" encoding="UTF-8"?>
                        <UpdateNodeRequest>
                        <TransactionNo>97</TransactionNo>
                        <TransactionID>DA/MARC24-1/11328/39</TransactionID>
                        <UpdateBlock>false</UpdateBlock>
                        <UpdateDataOnly>true</UpdateDataOnly>
                        <UseOnlySpecifiedData>true</UseOnlySpecifiedData>
                        </UpdateNodeRequest>
                        <<<<DATA>>>>
                        <Element><ID>00000000/00000067/000007E9/00000002/0000000C/0000006C/
                            00000002/00000064/00000068</ID><Title>Girls just want to have less
                            Fun</Title>
                        </Element>
2025-02-12 14:22:49.405 [21648] 10.1.1.18:51559 : (R-UPD) Receiving "Update Node" (Data) for ID =
                        "00000000/00000067/000007E9/00000002/0000000C/0000006C/00000002/
                         00000064/00000068" Flags = UpdateDataOnly,UseOnlySpecifiedData
                         TID = DA/MARC24-1/11328/39 ...
2025-02-12 14:22:49.405 [21648] MRSWMultiItemLock[Tree]: Granted write lock for item
                        "00000000/00000067/000007E9/00000002/0000000C/0000006C/00000002/
                         00000064/00000068" to thread 21648. LockID=1513.
2025-02-12 14:22:49.405 [21648] 10.1.1.18:51559 :    ... node info for ID =
                        "00000000/00000067/000007E9/00000002/0000000C/0000006C/00000002/
                         00000064/00000068": Tag = "Element";
                         Title = "Girls just want to have more Fun"
2025-02-12 14:22:49.405 [21648] 10.1.1.18:51559 : User "MARKUS" at WS "MARC24" : UPDATE DATA
                        (ID="00000000/00000067/000007E9/00000002/0000000C/0000006C/00000002/
                         00000064/00000068"); Node Info: Tag = "Element";
                        Title = "Girls just want to have less Fun" ... in Show
                        "Tagessendung" on 2025-02-12
2025-02-12 14:22:49.405 [21648] 10.1.1.18:51559 :  .... Title = "Girls just want to have less Fun"
2025-02-12 14:22:49.406 [21648] MRSWMultiItemLock[Tree]: Released write lock with ID 1513
                        of thread 21648.
2025-02-12 14:22:49.406 [21648] 10.1.1.18:51559 : (R-UPD) ... sending answer 1
2025-02-12 14:22:49.406 [21648] Sending compressed text message 23 to connection 4 /
                        client 10.1.1.18:51559. Message=<?xml version="1.0" encoding="UTF-8"?>
                        <BCSAnswer>
                        <OpCode>1</OpCode>
                        <TransactionNo>97</TransactionNo>
                        <TransactionID>DA/MARC24-1/11328/39</TransactionID>
                        <Result>1</Result>
                        </BCSAnswer>
2025-02-12 14:22:49.406 [22328] Message 23 was successfully written to connection 4 /
                        client 10.1.1.18.51559.
2025-02-12 14:22:49.414 [21760] Sending compressed text message 24 to connection 4 /
                        client 10.1.1.18:51559. Message=<?xml version="1. 0" encoding="UTF-8"?>
                        <UpdateNodeDataNotification>
                        <TransactionID>DA/MARC24-1/11328/39</TransactionID>
                        <FixedElementAdjustmentOnShowTimeChange>true
                           </FixedElementAdjustmentOnShowTimeChange>
                        </UpdateNodeDataNotification>
```

```
                        <<<<DATA>>>>
                        <Element><ID>00000000/00000067/000007E9/00000002/0000000C/0000006C/
                                    00000002/00000064/00000068</ID>
                        ……
                        </Element>
2025-02-12 14:22:49.414 [22328] Message 24 was successfully written to connection 4 /
                        client 10.1.1.18.51559.
```

You can see that much more is logged that with log level 1, which includes the data of the received and sent WebSocket messages. If this data is small enough, it is written completely to the log. But if the data exceeds something around 8000 characters, the data is cut off, because otherwise the log would explode.

On the top of this example, you can see a GetNode request. This was not added by mistake; it is part of the update-operation of DigAIRange. Why that? DigAIRange has an undo manager, which resides in between of the business logic and the WebSocket layer. When the undo manager sees the update request for the changed title, it first asks BCS about the current metadata of the element, so that it is able to restore the original values in case of an undo.

In some lines you can see operations about the tree lock, which is protecting parts of the data tree against simultaneous access of clients. You can see a read-lock, which would allow other simultaneous read locks and later an exclusive write lock.

At the end of the example appears first the direct answer, which confirms the success of the update operation. Afterwards, one notification to a client is shown. This is the same client, as requested the update operation, but all other clients, which have requested notifications, will be informed, too (which is left out in this example).

12.9.3 TurboPlayer – playing a sequenced element

The following example is an excerpt of a TurboPlayer engine log (BCE-TechLog/TechLogEx). It shows all relevant lines for the playout of one music element: “Men! I feel like a Woman” from Shania Twain. The element has a sequenced start and a mark-out, which is shorter than the length of the take, so we will see start and stop timers at work. A TurboPlayer log of this type contains a lot of other lines. But in order to have a clear example we have picked all lines, which refer to the mentioned element.

The element appears the first time when it comes into the range of preloaded elements and TurboPlayer requests a preload by MultiPlayer:

```
2025-02-13 13:29:30.730 Sending request to load internal element "13:49:56.824 Shania Twain:
                        Men! I feel like a Woman" in rundown Show. TransactionID=10000029
2025-02-13 13:29:30.780 Element "13:49:56.824 Shania Twain: Men! I feel like a Woman" of rundown
                        Show loaded. TransactionID=10000029, new FileID=8.
```

Next time, the element appears when the previous element is started and it becomes the <next> element. You can also see that TurboPlayer sets itself two timers for starting the element – because it is sequenced and the start time can be computed when the previous element hast started playing. As there is enough time till the start (more than 3 seconds), TurboPlayer also adds a prepare-timer 3 seconds before the start:

```
2025-02-13 13:49:37.104 Next element of rundown Show is now "13:49:56.824 Shania Twain: Men! I feel
                        like a Woman" (00000000/00000067/000007E9/00000002/0000000D/00000067/
                        00000002/00000066)
2025-02-13 13:49:37.208 New relative start timer event 49 for element "13:49:56.824 Shania Twain:
                        Men! I feel like a Woman" in rundown Show at time 2025-02-13 13:49:56.333
                        with pre-start-op delay 100 ms and predicted delay 2 ms. Anchor=00000067.
2025-02-13 13:49:37.208 New prepare timer event 50 for element "13:49:56.824 Shania Twain: Men! I
                        feel like a Woman" in rundown Show channel -1 at time 2025-02-13
                        13:49:53.333.
```

When the prepare timer elapses, TurboPlayer prepares the element to the intended playout channel. This causes MultiPlayer to load the audio data to a buffer on the soundcard, so that the upcoming start can be executed as fast as possible:

```
2025-02-13 13:49:53.333 Preparing (timed) element "13:49:56.824 Shania Twain: Men! I feel like a
                        Woman" of rundown Show for channel Ch-2.
2025-02-13 13:49:53.338 Element "13:49:56.824 Shania Twain: Men! I feel like a Woman" of rundown
                        Show prepared to channel Ch-2. Transaction ID: 10000037
```

About 3 seconds later, the start timer elapses and TurboPlayer now triggers the start. But you can see that a start command is not yet sent to MultiPlayer. Reason is that so-called pre-start operations are executed first. This can be multiple things, but in the log you can see one of these operations: the fader of the intended line need to be opened. Therefore, TurboPlayer sends an open-fader command via IOManager to the mixer console and sets itself one more timer to wait for the answer of the console that the fader is open. Please note that this does not delay the start behind the intended start point, because the pre-start operations are always tiggered earlier than the computed start point. Typically, 100 ms earlier – but this is configurable. If the mixer console does not respond within this time,

then the start will indeed really be delayed and the transition will not sound as scheduled. But in our example, you can see that the console responded within 0 ms (faster than 1 ms).

```
2025-02-13 13:49:56.334 Start timer event 49 for element "13:49:56.824 Shania Twain: Men! I feel
                        like a Woman" of rundown Show removed. Reason: Starting element
2025-02-13 13:49:56.334 New additional/delayed start timer event 51 for element "13:49:56.824
                        Shania Twain: Men! I feel like a Woman" of rundown Show and channel 2
                        at time 2025-02-13 13:49:56.434. Reason: move fader.
2025-02-13 13:49:56.334 IOManager(MARC24) moving fader 2 in 0 ms to level 0.
2025-02-13 13:49:56.334 IO module "SimIO" instance "Simulated-IO" reports fader for line 2 was
                        moved to level 0.
```

When the additional/delayed start timer elapses, TurboPlayer does now really send the start command to MultiPlayer. In addition, TurboPlayer sets itself a stop timer for the element. This is only done, because the element has a mark-out, earlier than end-of-take. Otherwise, such a timer is not necessary and TurboPlayer could wait until MultiPlayer reports that the end of the element was reached.

```
2025-02-13 13:49:56.434 Start timer event 51 for element "13:49:56.424 Shania Twain: Men! I feel
                        like a Woman" of rundown Show removed. Reason: Starting element
2025-02-13 13:49:56.434 Sending start command for element "13:49:56.424 Shania Twain: Men! I feel
                        like a Woman" in rundown Show (FileID=8) on channel Ch-2.
                        TransactionID=10000038.
2025-02-13 13:49:56.438 New (delayed) stop timer event 53 for element "13:49:56.435 Shania Twain:
                        Men! I feel like a Woman" of rundown Show at time 2025-02-13 13:52:48.822
                        with predicted delay 1 ms.
2025-02-13 13:49:56.438 Element "13:49:56.435 Shania Twain: Men! I feel like a Woman" FileID=8
                        Rundown=Show started at 2025-02-13 13:49:56.435 on channel Ch-2 /
                        line Ltg-2.
```

After some minutes of playing the element, the mark-out point is reached and the stop timer elapses. This causes TurboPlayer to send the stop command to MultiPlayer:

```
2025-02-13 13:52:48.822 Stop timer event 53 for element "13:49:56.435 Shania Twain: Men! I feel
                        like a Woman" of rundown Show removed. Reason: Stop element called
2025-02-13 13:52:48.822 Sending stop command for element "13:49:56.435 Shania Twain: Men! I feel
                        like a Woman" of rundown Show (FileID=8) on channel Ch-2.
                        TransactionID=10000045.
2025-02-13 13:52:48.827 Element "13:49:56.435 Shania Twain: Men! I feel like a Woman" FileID=8
                        Rundown=Show played for 00:02:52.390 ms,
                        stopped at 2025-02-13 13:52:48.824.
```

Then something interesting is happening. TurboPlayer moves the played element to a rundown of type “Deleted”. This happens, because TurboPlayer sets the element to send state “Sent” at BCS and this in turn made TreeManager to decide that this element is no longer needed and to remove it from the list of elements, which are sent to RundownKernel. The kernel should now remove the element from its lists, but as the element is still preloaded in MultiPlayer, it is still necessary to keep the metadata of the element somewhere in order to handle it correctly. This happens by moving the element to this special list. Shortly afterwards, a background tasks, which is responsible for the cleanup of elements, detects that the element can now be unloaded and sends the corresponding command to MultiPlayer. You can also see that the fader is closed a little time after the stop of the element:

```
2025-02-13 13:52:48.884 Moved element "13:49:56.435 Shania Twain: Men! I feel like a Woman" to
                        rundown list "Deleted" because element dropped out of original rundown
                        Show.
2025-02-13 13:52:48.902 IOManager(MARC24) moving fader 2 in 0 ms to level -99 (=closed).
2025-02-13 13:52:48.902 IO module "SimIO" instance "Simulated-IO" reports fader for line 2
                        was moved to level -99.
2025-02-13 13:52:49.936 Sending request to unload internal element "13:49:56.435 Shania Twain: Men!
                        I feel like a Woman" in rundown Deleted (originally: Show), FileID=8,
                        TransactionID=10000046
2025-02-13 13:52:49.947 Element "13:49:56.435 Shania Twain: Men! I feel like a Woman" of rundown
                        Deleted (FileID=8, TransactionID=10000046) unloaded.
```

If you are wondering about the different times: The time printed before the title is coming from BCS. First, it is a scheduling time, which is completely inaccurate until the element was really started and a real start time was reported to BCS. And if you see a line at 13:49:56.438, which reports that the element was started at 13:49:56.435, then this is caused by the delay of the real start, measured by MultiPlayer, and the time, when this start report has been handled by TurboPlayer, which is the log time.

In case you want to see the corresponding operations in the MultiPlayer log, you should look for the FileID – here it is number 8. This is the way how preloaded elements are referenced. One additional hint regarding log files of MulitPlayer: you can see the commands, which are received from TurboPlayer or which are sent to TurboPlayer by looking for commands in special brackets (here an example for the play command):

- [?PLAY?] This is the play command from TurboPlayer to MultiPlayer. Additional data about the file ID, the channel or the start level are sent in additional sections.
- [=PLAY=] This is the direct answer from MultiPlayer to TurboPlayer. Here we have a similar concept like the notifications of BCS. The direct answer only tells the caller (TurboPlayer) that this command was correct and was accepted and – in this case – that the start is now going to happen.
- [!PLAY!] This is the notification from MultiPlayer to TurboPlayer, which is sent, when an element really has been started. Additional sections give more information. In case of a start the real start time is the most important information.

12.10 Process (“Mini”) Dump Files

A mini dump file is an invention by Microsoft. It contains a part of the memory of a process, which is being written to a file in a special format. Such a file can be opened by some Microsoft tools like DumpChk, WinDbg or VisualStudio and can be used for analyzing the state of a process at a certain point in time (in this case at the time of a crash).

"Mini" can be read literally. Only a very small part of the memory of the process is normally written. This part is chosen by Windows when writing the dump file. In case of a crash Windows picks only some information around the crashing thread and some local variables on the stack. This might be enough to analyze a problem, but very often it is not. Therefore, a bigger part of the memory of the process can be written. The biggest possible part is to write all the memory of a process to the dump file. This is still called a "Mini dump", because it uses the same file format, but it will take much longer to write such a file and the file will be much bigger. Typically, you can expect some ten seconds for writing and many 100 MBytes of file size. This might be undesirable, especially, because you cannot restart the crashing process as long as the dump is being written. On the other hand, the full dump contains a lot more information, which can help to find a problem. A possible recommendation is to start with a mini dump and if the resulting dump does not contain enough information, try with a full dump in the next step.

Though it is possible to create MiniDump files with the DigaSQL.dll, it is not recommended to do so. Reason is that our experience has shown that this nearly never works in cases of a crashing application. Therefore, two other means of generating a MiniDump are proposed:

12.10.1 MiniDump files via Windows Error Reporting (WER)

Windows has everything available to directly generate a dump file when an application crashes.

To prepare the generation of dump files open RegEdit and navigate to:

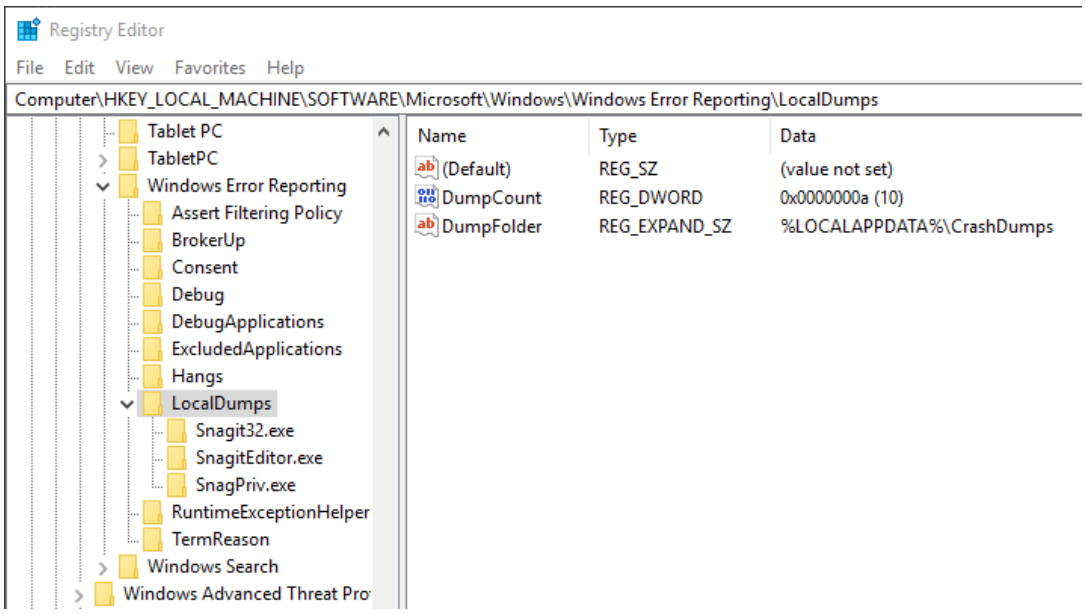
HKLM\Software\Microsoft\Windows\Windows Error Reporting

This key should always be available. The subkey "LocalDumps" might be available or not. If it is not available, create this subkey. Enter the following values:

- DumpFolder (REG_SZ): The folder where to put the create dump files.
- DumpCount (REG_DWORD): The number of dump files to keep before Windows deletes them automatically.
- DumpType (REG_DWORD): The type of the dumps being created.
  - 0: Custom dump (whatever that means...)
  - 1: Mini dump
  - 2: Full dump
- CustomDumpFlags (REG_DWORD): This value contains special flags, which control the behavior of the dump creation. Normally, don't set this value! Here are more details about the flags: https://docs.microsoft.com/en-us/windows/win32/api/minidumpapiset/ne-minidumpapiset-minidump_type

If the subkey "LocalDumps" was not configured, it is possible that Windows has placed the dump files somewhere else (only possibly...). A good try is the folder: C:\Users\<Username>\AppData\Local\CrashDumps. You can also search the harddisk for files with extension ".dmp".

The error reporting can also be configured with individual settings for each executable. You have to add a subkey with the name of the executable below "LocalDumps". Here is an example:



The value in "LocalDumps" are the general values, which apply to each application. Each of the possible values can appear in the per-application subkey, too.

If you want to have more information, here is a good start page from Microsoft:

https://docs.microsoft.com/en-us/windows/win32/wer/collecting-user-mode-dumps

12.10.2 MiniDump files via ProcDump

This is a tool from Mark Russinovich / SysInternals and is now owned by Microsoft themselves. You can download the tool from this page:
https://docs.microsoft.com/en-us/sysinternals/downloads/procdump
It is not necessary to download other tools from the SysInternals suite.
Then you need to run ProcDump permanently in a command line window. It must monitor the application and write a dump when the application crashes.
These command line options should be used:

- -e: Write a dump when the process encounters an unhandled exception.
- -ma: Write a dump file with all process memory. The dump file will be as big as the application just uses memory. This takes much longer, creates much bigger files, but analysis of the dump has much more possibilities to find the problem.
- -mm: Creates a mini dump only. Use alternatively to -ma if writing the full dump (which can take a minute or so) is not acceptable in a productive environment. Hint: as long as a dump file is being written, it is not possible to restart the application.
- -w: Wait for the specified process to launch if it's not running. Might be useful if you first start ProcDump and only afterwards the application.
- Process name or process ID

Example:

```
procdump -e -ma TurboPlayer
```

As it is not nice to have a permanent command line open (and the user possibly closing it), an alternative is to enter ProcDump as general "debugger". It will not do any real debugging, but will write a mini dump file whenever an application crashes. This is called a "postmortem debugger". The advantage is that you do not need to run a command line permanently.

You need to edit the Windows registry in key:

```
HKLM\Software\Microsoft\Windows NT\CurrentVersions\AeDebug
```

here set the value "Debugger" (REG_SZ) to the command line to execute, which is:

```
"C:\Program Files\Sysinternals\procdump.exe" /accepteula -ma %ld C:\Dumps
```

Please set the path "C:\Program Files\Sysinternals\" according to the correct location of your procdump.exe. The parameter %ld acts as placeholder for the crashing process and is replaced by Windows. The location "C:\Dumps" must exist and must be writeable with regular rights (best by each user of the computer). You can also first save the original value in order to restore it after debugging. Pay attention that ProcDump will create dumps whenever an application crashes. Each dump file can be very big. Therefore, monitor the dump directory!

Alternatively, you can call ProcDump with the option -i as an Administrator in order to install ProcDump as postmortem debugger.

12.10.3 MiniDump files via TaskManager

For completeness it should be mentioned, that you can use Windows TaskManager to create MiniDump files, too. Simply open TaskManager. Select the application for which you want to create the dump in the “process” tab and execute the context menu command “Create dump file”.

You only have to pay attention to one very nasty detail: if you want to create a dump file for a 32-bit application on a 64-bit Windows, you must use the 32-bit variant of TaskManager! By default, the 64-bit variant is started. So, you have to go to the Windows\SysWOW64 folder and execute TaskMgr.exe from this folder. You will not see a different in the GUI of both variants and you can only run one of them at a given time. If you go to the “Details” tab you can make the column “Platform” visible, which shows, whether an application is running as 64-bit or as 32-bit variant.

However, creating dump files via TaskManager is not helpful in case of a crash, because the function can only be called when an application is running. But it can be helpful in different situations, especially if an application is hanging and can no longer be operated via GUI. Then TaskManager is the easiest way to create a dump file.

12.11 Data to be provided in error reports

In case you want to send a report about a problem, an error or an application crash, you should check, whether the following information can be provided:

Description / analysis

You should always provide a text, which describes the problem. You can try to do an analysis yourself and send the result of it, but even if you cannot do an analysis, please think of these details in the report:

- What happened – expected result vs. real result.
- Was anything changed?
- Date and time of the incident(s).
- On which computer did the problem occur?
- Who had the problem? (DigaSystem user ID) Which operations did the user execute?
- In case multiple instances or tasks are configured or running: which configuration was the one, which had the problem or which BUS task had the problem?

Affected modules

Please provide a list of modules (exe, dll, ocx), which you think are part of the problem. Always give the exact version of these modules. If in doubt, better list too many modules than too few.

Log files

Log files are super-important for analyzing problems, which cannot be reproduced! Available log destinations are described above. Please be generous in the selection of log files! Whereas it is possible to send too few log files, it is not possible to send too many. So, in addition to the log files of the affected main module, think of loaded sub-modules, like IO modules in TurboPlayer or BCSSearch&ElasticIndex for a problem with full-text search. In many cases it is helpful to have the log files of BroadcastServer, because this is the central tool for storing all broadcast data and the operation of the clients can be visible in its logs. If you think that the root cause for a problem might be in the past, do not hesitate to add log files from a few days in the past, too. If a problem is (sometimes) reproducible, think about increasing the log level/output and to wait for a reproduction of the problem before sending the logs.

Parameters

Please do not send screenshot of setting dialogs! (Except if you have highlighted one specific setting.) Always send the registry parameter files! The local file of the affected computer(s) is always needed. The global file is only needed in some cases, but it does not hurt to send this file, too. User parameter files are normally not needed at all and can be sent only on special requests.

Rundowns

In most cases a problem or error is related to the data with a specific rundown. In order to analyze a problem, it is necessary to know the metadata in it. Please do not provide screenshots of a rundown, like a DigAIRange main window! Typically, the required information is in a not visible column or in the not visible area. You can provide a screenshot if you want to highlight a problem on a specific position, but then the rundown should still be added as a file.

The recommended way to generate a file to be sent is to have a DirectExport.dll configured as one of the exports in DigAIRange. Then you can easily drag&drop either a show or a day onto the corresponding export item in the tree view and create the file. As alternative you can browse with Windows Explorer through the directory where BCS stores its XML files. If you know the tree structure it is possible to find the right directory and extract the show file(s). Disadvantage is that this is more complicated and that you cannot generate a single file for a whole day. Therefore, the first alternative is recommended.

Please be generous when creating these files! Do not limit yourself to the single affected show. Always prefer to generate a file for the whole day. If you have the slightest doubt that the problem might have a root cause in the past, generate files for some days into the past, too!

If you want to provide the media files together with a rundown, you can use the FullExport.dll as export module. In this case you should only export a single show, because all media files will be copied to the output directory and then need to be sent. Typically, such an export is necessary if the problem is related to the media files itself or if the media files are necessary to reproduce a problem.

Macro / batch files

Macros are use in multiple places within BroadcastSystem, especially in TurboPlayer. These macros can be defined with registry parameters, but sometimes they are outsourced into extra files. If this was done and if you report a problem in TurboPlayer you should think about including the macro files, too. The same is true if you configured imports, which use external (vbs) batch files for preprocessing.

Special case: import problem

Normally, import problems can be reproduced without substantial obstacles. Reason is that importing is not time critical. When you have the same import and destination data and the same parameters, then the result will always be the same. An exception are problems with the database lookup. Reproducing them would mean to have the same database, which is unrealistic. So, in this case the focus would be more on analyzing.

Anyhow, for reproducing an import problem the items mentioned above are sufficient. We only can repeat them and add some details:

- Error report: Describe the expected and the real result. As normally multiple imports are configured, do not forget to describe, which import or BUS task is affected.
- Affected modules with version numbers. The import module must be part of it.
- Import data. This should be the original data, which is being read by the import module.
- Local parameter file.
- Destination rundown as it looks like before the import did run. Optionally, you can provide the destination rundown after the import, too, but normally this is not needed if the error description is clear.
- In case a macro or batch file is configured, this file is needed, too.

Head Office:

Erika-Mann-Str. 67
80636 Munich - Germany

Phone +49 89 540 139 0
Fax +49 89 540 139 50

office@davidsystems.com

Specifications and preliminary specifications are subject to change at any time without prior notice.
© 2025, DAVID Systems GmbH

Authors: Markus Hengstebeck
Karine Legrand
Andreas Parsch
Igor Barman
Georg Gruber
Carlos Oliveira
Sujata Sharma
Olga Seraya

File name: BCSTechManual.doc

Last revision: 27. February 2025